

Entwurf und Implementierung einer Terminal-Benutzeroberfläche zur Ver- waltung von PGP-Identitäten mit Hilfe von PGP-Schlüsseln

vorgelegt von

Moez Rjiba

EDV.Nr.: s837903

dem Fachbereich VI – Informatik und Medien
der Berliner Hochschule für Technik Berlin
vorgelegte Bachelorarbeit
zur Erlangung des akademischen Grades

Bachelor of Engineering (B.Eng.)

im Studiengang

Technische Informatik - Embedded Systems

Tag der Abgabe 5. Juni 2023



Betreuer

Prof. Dr. Christian Forler

Berliner Hochschule für Technik

Gutachter

Prof. Dr. rer. nat. Rüdiger Weis

Berliner Hochschule für Technik

Kurzfassung

Das digitale Zeitalter hat unsere Gesellschaft und das Leben des Einzelnen stark verändert. Von der persönlichen Kommunikation bis zu Finanztransaktionen, von der Gesundheitsversorgung bis zur Bildung – digitale Technologien haben die Art und Weise, wie wir leben, arbeiten und interagieren, revolutioniert. Mit der zunehmenden Verbreitung der digitalen Welt sind jedoch auch neue Komplexitäten und Schwachstellen entstanden. Datenmissbrauch, Identitätsdiebstahl und Eingriffe in die Privatsphäre sind in alarmierendem Maße alltäglich geworden, was zu einer verstärkten Besorgnis über die Sicherheit und den Schutz digitaler Daten führt.

In diesem komplizierten Umfeld bieten kryptographische Technologien wie Pretty Good Privacy (PGP) robuste Lösungen für die Sicherung der digitalen Kommunikation. Dennoch sind diese Technologien nicht frei von Herausforderungen, insbesondere was ihre Benutzerfreundlichkeit und Verwaltung betrifft. Die vorliegende Arbeit setzt an der Schnittstelle dieser Herausforderungen und Möglichkeiten an und versucht, eine benutzerfreundliche und effiziente Lösung für die Verwaltung von PGP-Schlüsseln zu finden.



Erklärung

Ich versichere, dass ich diese Abschlussarbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Datum

Unterschrift

Danksagung

Zunächst möchte ich meinem geschätzten Betreuer, Prof. Dr. Christian Forler, meine aufrichtige Dankbarkeit aussprechen. Seine unschätzbare Anleitung, seine kontinuierliche Unterstützung und seine fundierte Fachkenntnis haben diese Arbeit maßgeblich geprägt.

Ich schätze seine Bereitschaft, seine Weisheit weiterzugeben, sehr und danke ihm für die Zeit, die er sich genommen hat, um mir zu helfen, mein Verständnis zu verbessern und neue Perspektiven auf diesem Gebiet zu erkunden.

Außerdem wäre ich nachlässig, wenn ich nicht die unermüdliche Unterstützung meiner Eltern würdigen würde, deren Glaube an meine Fähigkeiten während meiner gesamten Reise eine beständige Quelle der Stärke und Motivation gewesen ist. Ihre Ermutigung und Liebe haben mir Halt und Orientierung gegeben, und dafür bin ich ihnen ewig dankbar.

Diese Arbeit ist ein Beweis für ihren gemeinsamen Einfluss, und dafür möchte ich ihnen meinen herzlichsten Dank aussprechen.

Inhaltsverzeichnis

1	Einführung	3
1.1	Motivation	3
1.2	Problemstellung	3
1.3	Ziel der Arbeit	3
1.4	Kapitelübersicht	4
2	Theoretische Grundlagen	5
2.1	Einführung in die PKI und Zertifikate	5
2.1.1	Public key Kryptographie	5
2.1.2	Digitale Signatur	5
2.1.3	Digitale Zertifikate	6
2.1.4	Public Key Infrastruktur	7
2.1.5	Zertifikatswiderruf (<i>Certificate Revocation</i>)	7
2.2	Einführung in PGP	8
2.2.1	PGP-Schlüsselverwaltung	8
2.2.1.1	Schlüsselgenerierung	8
2.2.1.2	Schlüsselverteilung	9
2.2.1.3	Schlüsselspeicherung	9
2.2.1.4	Schlüsselwiderruf	9
2.3	Rust, TUI und sequoia-openpgp	10
2.3.1	TUI-Tools und ihre Bedeutung	10
2.3.2	Verwandte Technologien und Tools	10
2.3.2.1	Rust	10
2.3.2.2	Tui-rs crate	11
2.3.2.3	Sequoia-openpgp crate	12
3	Konzept und Design	15
3.1	Anforderungsanalyse (<i>Systemanforderungen</i>)	15
3.2	Anwendungsfälle	16
3.3	Komponentenbeschreibung	18
4	Implementierung	23
4.1	Umsetzung der OpenPGP-Funktionen	23
4.1.1	Schlüssel-/Signatur Informationen abrufen:	24
4.1.2	Schlüsselpaar generieren:	25
4.1.3	Öffentlichen Schlüssel exportieren:	26
4.1.4	Passphrase bearbeiten:	26
4.1.5	Ablaufzeit bearbeiten:	27
4.1.6	Geteilter öffentlicher Schlüssel:	28
4.1.7	BenutzerID hinzufügen:	29
4.1.8	Widerrufen:	30

4.1.9	Die Implementierung des CertificateManager:	32
4.2	Umsetzung der Terminal-Benutzeroberfläche	33
4.2.1	Den Zustand einer Liste verwalten	34
4.2.2	Implementierung der App-Struktur	34
4.2.3	Komponenten der Benutzerschnittstelle	35
4.3	Verbindung der Komponenten	36
4.3.1	Wrapper Funktionen	36
4.3.2	Rendern der Schnittstelle	38
4.3.3	Tasten-Ereignisse	39
5	Ergebnisse	43
5.1	Das Hauptfenster	43
5.2	Die Benutzereingabe	44
5.3	Benutzer aufteilen	45
5.4	Schlüssel widerrufen	46
5.5	Hilfe-Fenster	47
6	Zusammenfassung und Ausblick	49
6.1	Zusammenfassung und Fazit	49
6.2	Zukünftige Entwicklungen	49
7	Anhang	51
7.1	Installationsanleitung	51
7.1.1	Voraussetzungen	51
7.1.2	Installation	51
7.1.3	Ausführung	52
7.1.4	Benutzerhandbuch	52
	Literatur- und Quellenverzeichnis	53

Listings

2.1	Result Enum[9]	10
2.2	Pattern Matching[10]	11
2.3	Closures in Rust[11]	11
4.1	get_key_details Funktion	24
4.2	get_signature_details Funktion	24
4.3	generate_key_pair Funktion	25
4.4	export_certificate Funktion	26
4.5	edit_password Funktion	27
4.6	edit_expiration_time Funktion	27
4.7	split_users Funktion	28
4.8	add_user Funktion	29
4.9	revoke_key Funktion	30
4.10	revoke_certificate Funktion	31
4.11	CertificateManager Implementierung	32
4.12	StatefullList Implementierung	34
4.13	App Struktur	34
4.14	Benutzer Eingabe	35
4.15	Erfolgs-/Fehlermeldungen popup	36
4.16	split_users_tui Wrapper Funktion	37
4.17	Implementierung der Hauptfunktion des Anwendungsläufers	38
4.18	Ereignisschleife in der Terminal-Benutzeroberfläche	39
4.19	Das Ereignis Tastendruck: Passwortänderung	39

Abbildungsverzeichnis

2.1	Public-Key-Kryptographie [3]	5
2.2	Digitale Signatur [1]	6
2.3	Web of Trust [8]	8
2.4	tui-rs popup beispiel [5]	12
3.1	Anwendungsfalldiagramm	16
3.2	Komponentendiagramm	19
3.3	Das App-Modul	19
3.4	Das Sequoia-OpenPGP-Modul	20
3.5	Das Utils-Modul	20
3.6	Das Widgets-Modul	21
4.1	Flußdiagramm für die Wrapper-Funktion <code>split_users_tui</code>	38
4.2	Flußdiagramm für das Ereignis Tastendruck: Passwortänderung	41
5.1	Hauptfenster der Schnittstelle	43
5.2	Parameter für die Erzeugung von Schlüsselpaaren	44
5.3	Ergebnis der Erzeugung von Schlüsselpaaren	45
5.4	Ergebnis der Aufteilung des öffentlichen Schlüssels	46
5.5	Ergebnis des Zertifikatswiderrufs	47
5.6	Hilfe-fenster der Schnittstelle	48

Abkürzungsverzeichnis

PGP	Pretty Good Privacy
CA	Certificate Authority
TUI	Terminal User Interface
GUI	Graphical User Interface
CLI	Command Line Interface
API	Application Programming Interface
CRL	Certificate Revocation List
OCSP	Online Certificate Status Protocol

Kapitel 1

Einführung

1.1 Motivation

Kryptographie ist die Grundlage der sicheren digitalen Kommunikation und schützt die Privatsphäre und Integrität unserer Daten. Unter den unzähligen kryptographischen Protokollen, die es gibt, zeichnet sich Pretty Good Privacy (PGP) durch seine Robustheit und Vielseitigkeit aus. Die Leistungsfähigkeit von PGP geht jedoch mit einer gewissen Komplexität einher.

Diese Komplexität, zusammen mit dem betrieblichen Aufwand und der steilen Lernkurve, die mit der PGP-Schlüsselverwaltung verbunden ist, schreckt die Benutzer oft ab, wodurch eine Lücke zwischen dem Potenzial von PGP und seiner Anwendung entsteht. Der Grund für die vorliegende Arbeit liegt genau in dieser Problematik.

Er zielt darauf ab, die Leistungsfähigkeit von PGP auf eine benutzerfreundliche Art und Weise nutzbar zu machen, indem er eine einfachere Schlüsselverwaltung ermöglicht und die Nutzung verschiedener Funktionen für den Endbenutzer leichter macht.

1.2 Problemstellung

Diese Arbeit befasst sich mit zwei miteinander verknüpften Problemen. Das erste Problem dreht sich um die Verwaltung von PGP-Schlüsseln, die Vorgänge wie das Überprüfen von Schlüssel-/Zertifikat/Signatur, das Erzeugen neuer Schlüsselpaare, das Extrahieren von Zertifikaten aus Schlüsseln, das Ändern von Passwörtern, das Hinzufügen von Benutzern zu einem Schlüssel und das Widerrufen eines Schlüssels umfasst.

Die Durchführung dieser Vorgänge erfordert ein detailliertes Verständnis von PGP und eine gewisse Vertrautheit mit den Befehlszeilenschnittstellen, was bei durchschnittlichen Benutzern nicht unbedingt üblich ist. Das zweite Problem betrifft die Verwaltung von gemeinsam genutzten PGP-Schlüsseln in Umgebungen mit mehreren Benutzern. Dieses Problem ergibt sich aus der Notwendigkeit, mehreren Benutzern die gemeinsame Nutzung desselben privaten Schlüssels zu ermöglichen, ohne dass für jeden Benutzer ein separater privater Schlüssel generiert werden muss, um so den betrieblichen Aufwand zu verringern.

1.3 Ziel der Arbeit

Das Hauptziel dieser Arbeit ist es, eine Lösung vorzuschlagen und zu entwickeln, die die PGP-Schlüsselverwaltung und die damit verbundenen Vorgänge auf eine benutzerfreundliche und intuitive Weise vereinfacht.

Bei der konzipierten Lösung handelt es sich um eine Terminal-Benutzeroberfläche (TUI), die die Kernfunktionen der PGP-Schlüsselverwaltung umfasst und somit eine zugängliche Schnittstelle für Benutzer bietet, ohne die Robustheit und Sicherheit von PGP zu beeinträchtigen.

Diese TUI soll die Lücke zwischen dem Potenzial und der Nutzung von PGP schließen, indem sie die technischen Fähigkeiten von PGP mit der Benutzerfreundlichkeit einer gut gestalteten Schnittstelle verbindet und so sichere digitale Kommunikation für eine breitere Nutzerbasis zugänglich macht.

Es ist wichtig zu beachten, dass die TUI zwar die am häufigsten verwendeten PGP-Funktionen abdeckt, aber noch nicht alle Funktionen von PGP implementiert. Die Erweiterung dieser Funktionen bleibt ein möglicher Bereich für zukünftige Entwicklungen, um die Funktionalität und Benutzerfreundlichkeit dieses Tools weiter zu verbessern.

1.4 Kapitelübersicht

Der Rest dieser Arbeit ist wie folgt gegliedert:

- Kapitel 2: „Theoretische Grundlagen“:
führt in die grundlegenden Konzepte ein, die zum Verständnis des Kontextes dieser Arbeit notwendig sind. Dazu gehören eine Einführung in die Public Key Infrastruktur (PKI) und Zertifikate, ein Überblick über den Widerruf und ein gründliches Verständnis von Pretty Good Privacy (PGP). Dieses Kapitel behandelt auch die Programmiersprache Rust, die Verwendung von Terminal User Interfaces (TUIs) und das OpenPGP-Protokoll. Es wird auch einen detaillierten Einblick in die spezifischen Technologien und Werkzeuge, nämlich `sequoia-openpgp` und `tui-rs` crates, geben, die bei der Entwicklung der in dieser Arbeit vorgeschlagenen Lösung entscheidend waren.
- Kapitel 3: „Konzept und Design“:
liefert eine detaillierte Anforderungsanalyse für das Projekt.

Das Anwendungsfalldiagramm und die Komponentenbeschreibung, die mit einem UML-Komponentendiagramm veranschaulicht wird, werden ebenfalls in diesem Kapitel vorgestellt.
- Kapitel 4: „Implementierung“:
befasst sich mit der detaillierten Implementierung der Komponenten und beleuchtet die Designentscheidungen und -Begründungen.
- Kapitel 5: „Ergebnisse“:
stellt die Anwendung vor und demonstriert ihre Effizienz und Benutzerfreundlichkeit.
- Kapitel 6: „Zusammenfassung und Ausblick“:
schließt die Arbeit ab, indem es die Ergebnisse dieser Arbeit zusammenfasst und zukünftige Entwicklungen diskutiert, die auf dieser Grundlage aufbauen könnten.

Schließlich enthält der „Anhang“ eine Installationsanleitung, die Schritt für Schritt durch die Installation und Ausführung der PGP Manager TUI führt.

Kapitel 2

Theoretische Grundlagen

2.1 Einführung in die PKI und Zertifikate

2.1.1 Public key Kryptographie

Die Public-Key-Kryptographie, auch bekannt als asymmetrische Kryptographie, ist ein grundlegender Bestandteil der Struktur moderner digitaler Sicherheit. Bei dieser Verschlüsselungstechnik wird ein Schlüsselpaar verwendet: ein öffentlicher Schlüssel und ein privater Schlüssel. Das Besondere an diesem System ist, dass die Schlüssel mathematisch miteinander verknüpft sind, es aber rechnerisch nicht möglich ist, den privaten Schlüssel vom öffentlichen abzuleiten.

Der öffentliche Schlüssel ist, wie der Name schon sagt, offen verteilt und für jeden zugänglich. Er wird zum Verschlüsseln von Nachrichten oder zum Überprüfen digitaler Signaturen verwendet. Der private Schlüssel hingegen bleibt seinem Besitzer vorbehalten und wird zum Entschlüsseln von Nachrichten oder zum Signieren digitaler Dokumente verwendet.

Im Zusammenhang mit der Verschlüsselung von Nachrichten würde ein Absender den öffentlichen Schlüssel des Empfängers verwenden, um eine Nachricht zu verschlüsseln. Nach dem Empfang würde der Empfänger dann seinen privaten Schlüssel verwenden, um den Inhalt zu entschlüsseln und zu lesen. Bei digitalen Signaturen würde der Absender seinen privaten Schlüssel verwenden, um eine eindeutige Signatur für ein Dokument zu erstellen, und der Empfänger würde die Signatur mit dem öffentlichen Schlüssel des Absenders überprüfen. [6]

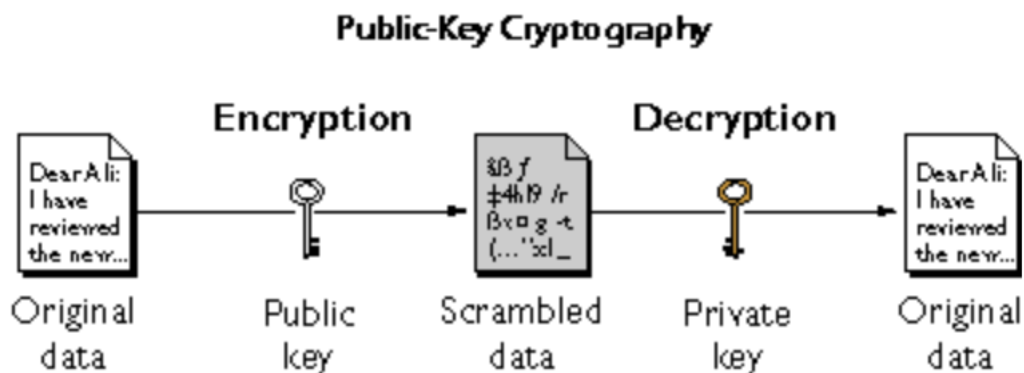


Abbildung 2.1: Public-Key-Kryptographie [3]

2.1.2 Digitale Signatur

Eine digitale Signatur ist ein von der Verschlüsselung abgeleiteter Mechanismus, der die Authentizität, Nichtabstreitbarkeit und Integrität von digitalen Daten gewährleistet. Der Prozess beginnt

damit, dass ein privater Schlüssel verwendet wird, um eine digitale Signatur zu erstellen, die mit einem bestimmten Datenteil verbunden ist, z.B. einer E-Mail oder einem digitalen Vertrag. Diese Signatur wird dann an die Daten selbst angehängt oder zusammen mit ihnen versandt. Beim Empfang der Daten verwendet der Empfänger den öffentlichen Schlüssel des Absenders, um die digitale Signatur zu überprüfen. Ist die Signatur gültig, kann der Empfänger sicher sein, dass die Daten während der Übertragung nicht manipuliert wurden (Integrität) und dass sie tatsächlich von der angegebenen Quelle stammen (Authentizität). Außerdem kann der Absender nicht leugnen, dass er die Nachricht gesendet hat (Nichtabstreitbarkeit).

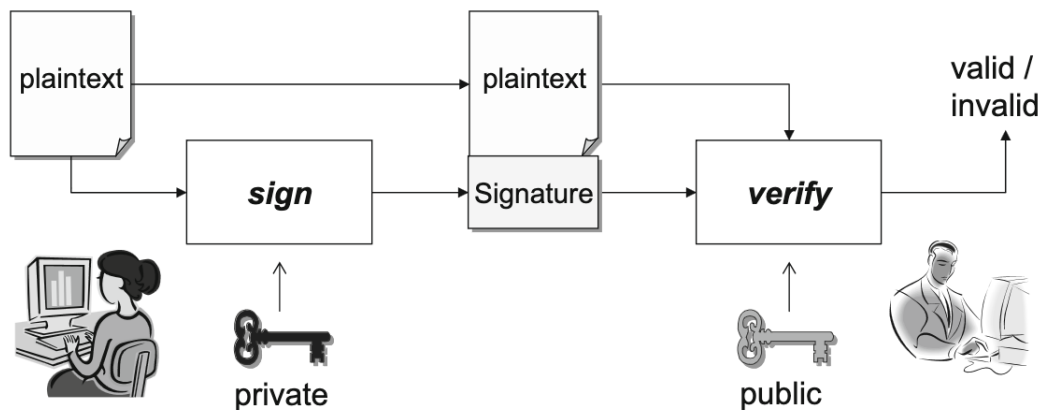


Abbildung 2.2: Digitale Signatur [1]

2.1.3 Digitale Zertifikate

„Eine der Hauptaufgaben einer Public-Key-Infrastruktur (PKI) besteht darin, die Authentizität öffentlicher Schlüssel nachzuweisen. Zertifikate sind ein wesentliches Instrument, um solche Überprüfungsprozesse zu erleichtern.“ [2]

Die Struktur und der Inhalt digitaler Zertifikate werden durch Normen wie X.509 geregelt, die in vielen Internetprotokollen zur Gewährleistung sicherer Transaktionen weit verbreitet sind. X.509 wurde von der International Telecommunication Union (ITU) entwickelt[16]. X.509-Zertifikate werden in vielen Internetprotokollen verwendet, darunter TLS/SSL, das die Grundlage für HTTPS, das sichere Protokoll für das Browsen im Internet, bildet. Sie werden auch für das Signieren und Verschlüsseln von E-Mails, das Signieren und Verschlüsseln von Daten zum Schutz der Integrität und Vertraulichkeit der Daten und für die Authentifizierung von sicheren VPNs verwendet. Ein X.509-Zertifikat enthält die folgenden Informationen:

1. Version: Die Version des verwendeten X.509-Standards.
2. Seriennummer: Eindeutiger Bezeichner für das Zertifikat.
3. Kennung des Signaturalgorithmus: Der von der Zertifizierungsstelle zum Signieren des Zertifikats verwendete Algorithmus.
4. Aussteller: Der Name der Einrichtung (der Zertifizierungsstelle), die das Zertifikat ausstellt und signiert hat.
5. Gültigkeitsdauer: Der Zeitraum, in dem das Zertifikat gültig ist, einschließlich eines Start- und Enddatums.

6. Betreff: Der Name der Entität, auf die das Zertifikat ausgestellt ist.
7. Informationen zum öffentlichen Schlüssel, des Betreffs: Dieser Abschnitt enthält den öffentlichen Schlüssel und den zur Erstellung des öffentlichen Schlüssels verwendeten Algorithmus.
8. Erweiterungen: Diese sind optional und können zusätzliche Informationen enthalten, wie z. B. die Verwendung des Schlüssels.
9. Unterschrift: Die Signatur der ausstellenden Zertifizierungsstelle, die aus einem Hash des Zertifikats und der Verschlüsselung dieses Hashs mit dem privaten Schlüssel der Zertifizierungsstelle gebildet wird.

2.1.4 Public Key Infrastruktur

Was ist eine Public-Key-Infrastruktur (PKI)?

Laut dem Bundesamt für Sicherheit in der Informationstechnik (BSI):

„Eine Public Key Infrastruktur (PKI, Infrastruktur für öffentliche Schlüssel) ist ein hierarchisches System zur Ausstellung, Verteilung und Prüfung von digitalen Zertifikaten. Die digitalen Zertifikate ermöglichen eine vertrauenswürdige Zuordnung von Entitäten zu ihren öffentlichen Schlüsseln. Eine PKI umfasst den Betrieb vertrauenswürdiger IT-Systeme, Prozesse und Richtlinien.“[14]

Basierend auf dieser Definition bietet eine Public Key Infrastruktur (PKI) den wesentlichen Rahmen für eine sichere Online-Datenübertragung und -Kommunikation. Im Kern beruht sie auf den Grundsätzen der asymmetrischen Kryptographie.

Im Rahmen der PKI spielen digitale Zertifikate, ähnlich wie digitale Reisepässe, eine zentrale Rolle. Sie dienen dazu, die Identität des Inhabers zu authentifizieren und eine Entität mit ihrem öffentlichen Schlüssel zu verbinden. Dies geschieht durch die Verknüpfung der identifizierenden Informationen der Einheit mit dem öffentlichen Schlüssel, wodurch sichergestellt wird, dass die Einheit wirklich der Eigentümer dieses Schlüssels ist. Die Ausstellung, Verwaltung und Überprüfung dieser digitalen Zertifikate wird von einer vertrauenswürdigen dritten Instanz, der sogenannten Zertifizierungsstelle (CA), überwacht.

2.1.5 Zertifikatswiderruf (*Certificate Revocation*)

Der Widerruf ist ein wichtiger Mechanismus im Bereich der digitalen Zertifikate und der Infrastrukturen für öffentliche Schlüssel. Es geht um die Situation, in der einem digitalen Zertifikat nicht mehr vertraut werden kann, d. h. es wird vor seinem festgelegten Ablaufdatum annulliert oder ungültig gemacht.

Eine solche Notwendigkeit kann aus verschiedenen Gründen entstehen, z. B. wenn der mit dem Zertifikat verbundene private Schlüssel kompromittiert wurde oder wenn das Zertifikat falsch oder böswillig ausgestellt wurde. Die Stelle, die die Ausstellung und den Widerruf von Zertifikaten verwaltet, die sogenannte Zertifizierungsstelle (CA), führt eine Aufzeichnung dieser widerrufenen Zertifikate in einer sogenannten Zertifikatswiderrufsliste (CRL)[4].

Durch den Widerruf wird sichergestellt, dass Systeme, die sich für eine sichere Kommunikation auf diese Zertifikate verlassen, von der Sperrung Kenntnis erhalten. Ein alternativer Mechanismus, das Online Certificate Status Protocol (OCSP), bietet einen Überprüfungsprozess in Echtzeit. Anstatt eine potenziell umfangreiche CRL herunterzuladen, kann ein System eine Anfrage an einen OCSP-Responder senden, um den Status eines bestimmten Zertifikats zu erfahren[4].

2.2 Einführung in PGP

Pretty Good Privacy (PGP), entwickelt von Phil Zimmermann, ist ein Computerprogramm zur Ver- und Entschlüsselung von Daten, das kryptographischen Datenschutz und Authentifizierung für die Datenkommunikation bietet. Das PGP-Modell stützt sich auf die gleichen Grundpfeiler der Public Key Infrastructure (PKI), nämlich die Verwendung digitaler Signaturen, digitaler Zertifikate und Public-Key-Kryptographie. PGP weicht jedoch vom zentralisierten PKI-Modell ab, indem es ein dezentrales Vertrauensmodell verwendet, das als "Web of Trust" bekannt ist.[8]

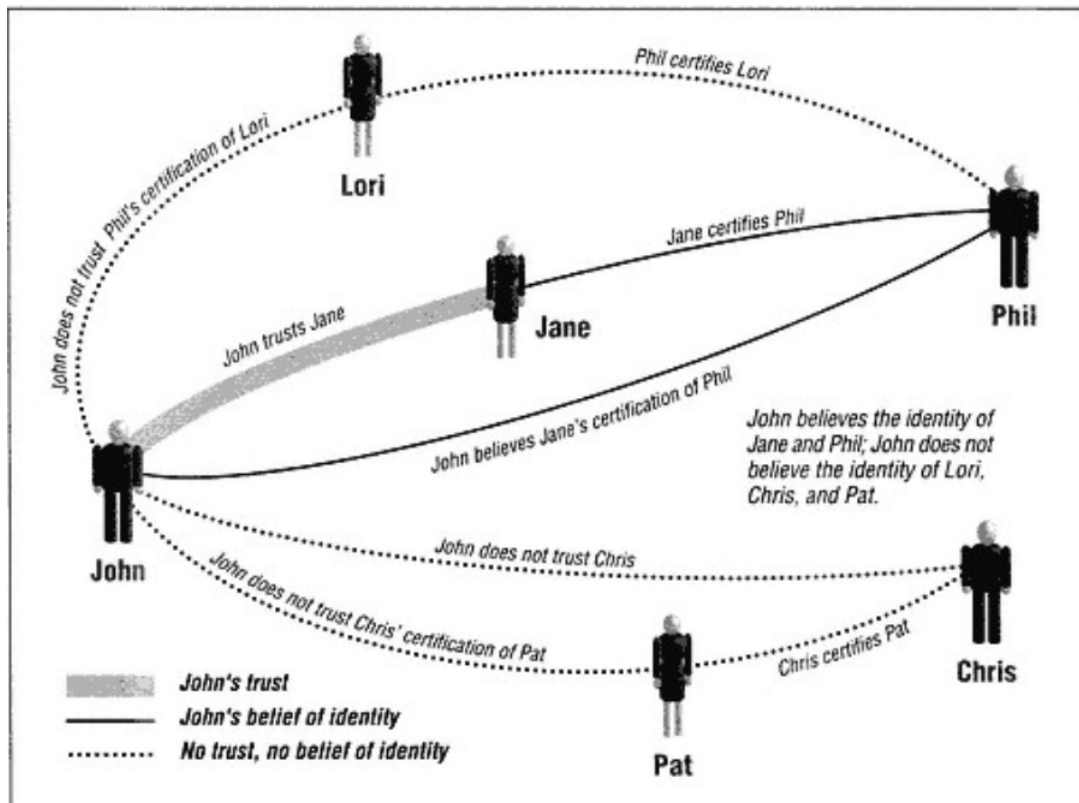


Abbildung 2.3: Web of Trust [8]

Jeder PGP-Benutzer unterhält einen Ring des Vertrauens, d.h. einen Satz öffentlicher Schlüssel von anderen Benutzern, die bis zu einem gewissen Grad vertrauenswürdig sind. Mit diesem Modell können die Benutzer selbst entscheiden, wem sie vertrauen, anstatt sich auf eine zentrale Zertifizierungsstelle (CA) zu verlassen, die vertrauenswürdige Zertifikate ausstellt.

2.2.1 PGP-Schlüsselverwaltung

Der Vorgang der Schlüsselverwaltung in PGP ist von zentraler Bedeutung für den Betrieb und die Sicherheit des Programms. Sie bezieht sich auf die systematische Erstellung, Verteilung, Speicherung und den Widerruf von Schlüsseln, die es den Benutzern ermöglichen, Daten sicher zu verschlüsseln, zu entschlüsseln, zu signieren und zu verifizieren. Im Folgenden werden die kritischen Elemente des PGP-Schlüsselverwaltungssystems beschrieben.

2.2.1.1 Schlüsselgenerierung

Der Prozess der Schlüsselverwaltung beginnt mit der Erzeugung eines Schlüsselpaares. Bei der Generierung wählt der Benutzer einen Verschlüsselungsalgorithmus und eine Schlüssellänge

aus, oft gefolgt von der Angabe einer Benutzer-ID (normalerweise eine E-Mail-Adresse) und einer starken Passphrase zum Schutz des privaten Schlüssels.

Dieses Schlüsselpaar bildet die PGP-Identität des Benutzers. Der öffentliche Schlüssel wird anderen zur Verfügung gestellt, um Nachrichten an den Benutzer zu verschlüsseln oder seine Signaturen zu überprüfen, während der private Schlüssel vertraulich behandelt und zum Entschlüsseln empfangener Nachrichten und zum Signieren von Daten verwendet wird.

2.2.1.2 Schlüsselverteilung

Sobald ein Schlüsselpaar erzeugt wurde, wird der öffentliche Schlüssel anderen Nutzern über einen Schlüsselserver oder durch direkten Austausch zur Verfügung gestellt. Schlüsselserver sind öffentliche Repositories, in denen Benutzer nach den öffentlichen Schlüsseln anderer suchen und diese herunterladen können. Sie sind ein wichtiger Bestandteil des PGP-Ökosystems, da sie die einfache Verteilung und Auffindung öffentlicher Schlüssel ermöglichen.

Wenn ein Benutzer eine verschlüsselte Nachricht oder ein signiertes Dokument sendet, kann er außerdem seinen öffentlichen Schlüssel anhängen. Der Empfänger kann dann diesen Schlüssel für die zukünftige Kommunikation mit dem Absender oder zur Überprüfung der empfangenen Daten verwenden.

2.2.1.3 Schlüsselspeicherung

PGP verwendet eine Datenstruktur, die als Keyring bezeichnet wird, um Schlüssel zu speichern und zu verwalten. Jeder Benutzer verwaltet zwei Keyrings: einen öffentlichen Keyring und einen privaten Keyring. Der öffentliche Keyring enthält die öffentlichen Schlüssel anderer PGP-Benutzer, und der private Keyring speichert die privaten Schlüssel des Benutzers.

Diese Ringe, insbesondere der private Keyring, müssen entsprechend geschützt werden. Eine gängige Sicherheitsmaßnahme ist die Verwendung einer starken Passphrase, um den privaten Keyring zu verschlüsseln. Wenn ein Angreifer Zugriff auf die Keyring-Datei erhält, müsste er die Passphrase erraten oder knacken, um die darin enthaltenen Schlüssel zu verwenden.

2.2.1.4 Schlüsselwiderruf

Das PGP-System ermöglicht die Erstellung von sogenannten „Widerrufszertifikaten“. Ein Widerrufszertifikat in PGP ist im Wesentlichen eine spezielle Art von Signatur, die verwendet werden kann, um ein PGP-Zertifikat ungültig zu machen, wenn der zugehörige private Schlüssel kompromittiert wurde, verloren gegangen ist oder nicht mehr verwendet wird.

Wenn ein PGP-Zertifikat erstellt wird, hat der Benutzer die Möglichkeit, auch ein Widerrufszertifikat zu erzeugen. Es empfiehlt sich, dieses Widerrufszertifikat zu erstellen und es an einem sicheren Ort aufzubewahren. Sollte dann jemals die Notwendigkeit entstehen, das PGP-Zertifikat zu widerrufen, kann der Benutzer dieses Widerrufszertifikat auf dem Server für öffentliche Schlüssel veröffentlichen, auf dem sein PGP-Zertifikat gehostet wird. Damit wird jedem, der die Gültigkeit des PGP-Zertifikats überprüft, signalisiert, dass es nicht mehr vertrauenswürdig ist.

Es ist wichtig zu beachten, dass ein einmal veröffentlichtes Widerrufszertifikat nicht mehr rückgängig gemacht werden kann. Das PGP-Zertifikat wird als dauerhaft ungültig betrachtet.

Ein wesentlicher Unterschied zu den Mechanismen des Zertifikatswiderrufs in PKI-Systemen wie X.509 besteht darin, dass PGP keine zentralisierte Zertifizierungsstelle (CA) verwendet oder eine Zertifikatswiderrufsliste (CRL) führt. Stattdessen basiert das Vertrauen in PGP in der Regel auf einem „Web of Trust“-Modell, bei dem die einzelnen Benutzer für die Authentizität der Zertifikate der anderen Benutzer einstehen. Die Verantwortung für die Erstellung, Sicherung und eventuelle Veröffentlichung eines Widerrufszertifikats liegt also beim einzelnen Benutzer und nicht bei einer zentralen Stelle.

Die Verwaltung von Schlüsseln in PGP ist ein sensibler Prozess, der eine sorgfältige Vorgehensweise und die Einhaltung etablierter Sicherheitsgrundsätze erfordert. Eine umfassende Erforschung dieses Themas findet sich in den Arbeiten *"PGP & GPG: Email for the Practical Paranoid"* von Michael W. Lucas [13] und *"PGP: Pretty Good Privacy"* von Simson Garfinkel [8]. Diese Bücher lieferten die Grundlagen für die Erläuterung von PGP in dieser Arbeit und sind nach wie vor eine unverzichtbare Lektüre für alle, die ein tieferes Verständnis dieser Konzepte anstreben.

2.3 Rust, TUI und sequoia-openpgp

2.3.1 TUI-Tools und ihre Bedeutung

Terminal-Benutzerschnittstellen- oder textbasierte Benutzerschnittstellen-Tools (TUI) sind Software-Dienstprogramme, die für textbasierte Benutzerschnittstellen entwickelt wurden und es den Benutzern ermöglichen, über Terminal-Emulatoren mit dem System zu interagieren, ähnlich wie bei einer Befehlszeilenschnittstelle (CLI), jedoch mit einer zusätzlichen grafischen Ebene. Mit diesen Werkzeugen lassen sich umfangreiche, textbasierte Benutzeroberflächen für eine Vielzahl von Terminalanwendungen erstellen.

Im Gegensatz zu Werkzeugen für grafische Benutzeroberflächen (GUI) sind TUI-Werkzeuge in der Regel weniger ressourcenintensiv und können für bestimmte Aufgaben, insbesondere solche, die Befehlszeilenaktivitäten beinhalten, sehr effizient sein.

2.3.2 Verwandte Technologien und Tools

2.3.2.1 Rust

Die offizielle Website von Rust besagt Folgendes:

„ Rust is blazingly fast and memory-efficient: with no runtime or garbage collector, it can power performance-critical services, run on embedded devices, and easily integrate with other languages.“ [15]

In diesem Abschnitt werden wir einige wichtige Konzepte von Rust behandeln, die bei der Entwicklung dieses Projekts nützlich waren.

1. Fehlerbehandlung[9]: Eine der mächtigen Eigenschaften von Rust, die wesentlich zu seiner Zuverlässigkeit und Sicherheit beitragen, ist der Mechanismus zur Fehlerbehandlung. Dieser Mechanismus dreht sich um einen Typ, der als „Ergebnis“ bekannt ist.

Das `Result` Enum hat zwei Varianten: `'Ok'` und `'Err'`. Die Variante `'Ok'` wird verwendet, wenn eine Operation erfolgreich war, und sie enthält das erfolgreiche Ergebnis. Die Variante `'Err'` hingegen wird verwendet, wenn eine Operation fehlschlägt, und enthält Informationen über den Fehler. Der Ergebnistyp ist eine generische Aufzählung, bei der `'T'` und `'E'` Platzhalter für beliebige Typen sind. Dieses Design ist sehr flexibel und erlaubt es, eine Vielzahl von Erfolgs- und Fehlerszenarien auszudrücken.

```
enum Result<T, E> {  
    Ok(T),  
    Err(E),  
}
```

Listing 2.1: Result Enum[9]

2. Pattern Matching[10]: Pattern Matching in Rust ermöglicht es, komplexe Datenstrukturen zu destrukturieren und zu überprüfen. In Kombination mit dem leistungsfähigen Enum-System von Rust ermöglicht Pattern Matching ausdrucksstarke Codierungsstile, die verschiedene Datenzustände effektiv und sicher behandeln können.

```
enum Coin {
    Penny,
    Nickel,
    Dime,
}

fn value_in_cents(coin: Coin) -> u8 {
    v match coin {
        w Coin::Penny => 1,
        Coin::Nickel => 5,
        Coin::Dime => 10,
    }
}
```

Listing 2.2: Pattern Matching[10]

3. Closures in Rust [11] sind anonyme Funktionen, die man in einer Variablen speichern oder als Argumente an andere Funktionen übergeben kann. Sie sind flexibel und ermöglichen On-the-Fly-Funktionalität. Die Iteratoren von Rust sind kostenneutral und werden im idiomatischen Rust ausgiebig verwendet.

```
let expensive_closure = |num| {
    println!("calculating_slowly...");
    thread::sleep(Duration::from_secs(2));
    num
};
```

Listing 2.3: Closures in Rust[11]

4. Generics sind abstrakte Stellvertreter für konkrete Typen oder andere Eigenschaften. Sie werden für Code verwendet, der für mehrere Arten von Argumenten funktionieren kann. In Rust ermöglichen Generics die Abstraktion von Typen unter Beibehaltung der Laufzeitleistung. [12]

2.3.2.2 Tui-rs crate

Tui-rs ist eine beliebte Rust-Bibliothek, die für die Erstellung von reichhaltigen Terminal Benutzerschnittstellen (TUIs) und Dashboards entwickelt wurde. Diese Kiste, die von Bibliotheken wie Blessed-contrib und Termui inspiriert ist, vereinfacht den Prozess der Erstellung anspruchsvoller textbasierter Benutzerschnittstellen. Sie bietet eine Vielzahl von Funktionen, darunter grundlegende Widgets wie Block, Text, Absatz, Liste, Messgerät, Diagramm usw. Darüber hinaus unterstützt es auch komplexere Widgets wie Tabellen. Eines der Hauptmerkmale von tui-rs ist seine Backend-Abstraktion. Es unterstützt mehrere Backends, wie z.B. crossterm und termion, sodass die Entwickler dasjenige auswählen können, das ihren Bedürfnissen am besten entspricht. [5]

Beim Aufbau einer Terminal-Benutzerschnittstelle (TUI) mit tui-rs besteht die Schnittstelle im Allgemeinen aus mehreren wichtigen Komponenten:



Abbildung 2.4: tui-rs popup beispiel [5]

1. Backend: Dies ist die Komponente, die das eigentliche Zeichnen auf dem Terminal durchführt. Die Wahl des Backends hängt von dem Kontext ab, in dem Ihre Anwendung läuft. `tui-rs` bietet Unterstützung für eine Vielzahl von Backends, wie `Crossterm` und `Termion`.
2. Terminal: Diese Komponente fungiert als Brücke zwischen dem Backend und der Anwendung und bietet eine High-Level-API für die Interaktion mit dem Terminal (z. B. Löschen des Bildschirms, Zeichnen von Inhalten, Aktivieren des Rohmodus).
3. Widgets: Widgets in `tui-rs` sind Komponenten wie Blöcke, Tabellen, Diagramme, Absätze, Listen und mehr. Sie stellen den Inhalt Ihrer TUI dar. Sie definieren, wie jedes Widget aussehen und sich verhalten soll, und die Bibliothek kümmert sich darum, sie auf dem Terminal darzustellen.
4. Layouts: Layouts helfen dabei, die Widgets auf dem Bildschirm zu organisieren. Sie definieren, wo und wie die Widgets im Verhältnis zueinander platziert werden. `tui-rs` bietet ein flexibles Layout-System, mit dem Sie komplexe UI-Designs erstellen können.
5. Zustand: Der Status stellt die Daten dar, mit denen die Anwendung arbeitet. Er umfasst den aktuellen Zustand der Benutzeroberfläche, z. B. welches Widget ausgewählt ist oder den Inhalt einer Liste, und die Daten, die die Anwendung verwaltet, z. B. Benutzereingaben oder Ergebnisse von Operationen.
6. Ereignisse: Ereignisse sind beispielsweise Benutzereingaben (Tastatureingaben, Mausebewegungen, Klicks) oder Signale vom System. Die Anwendung muss diese Ereignisse verarbeiten und den Zustand und die Benutzeroberfläche entsprechend aktualisieren.

2.3.2.3 Sequoia-openpgp crate

Die Sequoia OpenPGP-Bibliothek [7], einschließlich der `sequoia-openpgp` Crate, wird von einem engagierten Team entwickelt, von dem einige Mitglieder zuvor an der Entwicklung von

GnuPG, einer gut etablierten OpenPGP-Implementierung, mitgewirkt haben. Es bietet eine umfangreiche und gut dokumentierte API für die Arbeit mit OpenPGP. Insbesondere bietet es sowohl eine High-Level- als auch eine Low-Level-API für Flexibilität.

Die Low-Level-API von `sequoia-openpgp` gibt Entwicklern direkten Zugriff auf die OpenPGP-Datenstrukturen. Es handelt sich um eine umfangreiche und mächtige Verbindung, die die direkte Manipulation von OpenPGP-Paketen, wie literalen Datenpaketen, Public-Key-Paketen und Signaturpaketen ermöglicht.

So können Entwickler OpenPGP-Pakete nach ihren Bedürfnissen erstellen, parsen, serialisieren und inspizieren. Dieser Grad an Kontrolle ist besonders nützlich für Anwendungen, die ein gewisses Maß an Anpassung oder Interaktion mit OpenPGP erfordern, was in diesem Projekt der Fall war. Obwohl dies vom jeweiligen Anwendungsfall abhängt, gehören zu den am häufigsten verwendeten Funktionen:

1. Parsen und Serialisieren von OpenPGP-Daten: Die Low-Level-API ermöglicht das Parsen von OpenPGP-Daten aus einer Vielzahl von Formaten, einschließlich binärer und ASCII-armored Daten. Umgekehrt können Nutzer OpenPGP-Datenstrukturen in diese Formate serialisieren.
 - (a) `'PacketPile::from_reader()'`: zum Parsen von Daten aus einem Reader.
 - (b) `'PacketPile::to_vec()'`: zum Serialisieren eines `PacketPile` in einen Byte-Vektor.
2. Arbeiten mit OpenPGP-Nachrichten: Dazu gehört die Möglichkeit, OpenPGP-Nachrichten zu erstellen, zu verschlüsseln und zu entschlüsseln. Die API ermöglicht auch das Signieren und Überprüfen der Signaturen dieser Nachrichten.
3. Schlüsselverwaltung: Dazu gehören Funktionen wie die Erstellung und Bearbeitung von OpenPGP-Schlüsseln. Nutzer können unter anderem neue Schlüsselpaare erzeugen, Unterschlüssel hinzufügen oder entfernen und Verfallszeiten für Schlüssel festlegen.
`'CertBuilder::new()'`: zur Erstellung komplexer Zertifikatshierarchien.
4. Zertifikatsverwaltung: Die Low-Level-API ermöglicht die Arbeit mit OpenPGP Zertifikaten. Der Benutzer kann Zertifikate prüfen, ihre Signaturen verifizieren und mit Benutzer-IDs und Benutzerattributen arbeiten. Darüber hinaus können sie auch mit Zertifikatswiderrufen arbeiten.
 - (a) `'Cert::from_bytes(), Cert::from_reader()'`: um ein Zertifikat zu parsen.
 - (b) `'Cert::keys(), Cert::with_policy()'`: um mit Schlüsseln und Richtlinien in einem Zertifikat zu arbeiten.
 - (c) `'Cert::revoke'`: um mit Widerrufszertifikaten zu arbeiten.
5. Paket-Manipulation: Es ist möglich direkt mit den Paketen zu arbeiten, aus denen die OpenPGP-Daten bestehen. Dies gibt dem Nutzer ein hohes Maß an Kontrolle und Flexibilität, erfordert aber ein gutes Verständnis der OpenPGP-Spezifikation.

Weitere Einzelheiten über die Verwendung und die Möglichkeiten der Low-Level-API finden Sie in der API-Dokumentation, die in [der offiziellen Sequoia OpenPGP-Bibliothek](#) verfügbar ist.

Kapitel 3

Konzept und Design

3.1 Anforderungsanalyse (*Systemanforderungen*)

In diesem Abschnitt werden die wichtigsten Anforderungen untersucht, die den Entwurfs- und Entwicklungsprozess geleitet haben.

Die Systemanforderungen können in zwei Hauptaspekte eingeteilt werden:

Funktionale Anforderungen und nicht-funktionale Anforderungen.

1. Funktionale Anforderungen

- (a) Schlüsselprüfung:
Die TUI sollte die Möglichkeit bieten, Schlüssel auf Details hin zu untersuchen.
- (b) Schlüsselgenerierung:
Die TUI sollte die Möglichkeit bieten, neue PGP-Schlüsselpaare für Benutzer zu erzeugen. Dazu gehören Optionen zur Angabe von Schlüsselparametern wie Schlüsseltyp, Schlüssellänge und Verschlüsselungsalgorithmus.
- (c) Öffentlichen Schlüssel exportieren:
Die TUI sollte es den Benutzern ermöglichen, generierte Schlüssel zur externen Verwendung zu exportieren.
- (d) Schlüsselverwaltung:
Die TUI sollte die Verwaltung von PGP-Schlüsseln erleichtern, einschließlich Vorgängen wie dem Hinzufügen und Entfernen von Benutzern zu einem Schlüssel, dem Teilen von Benutzern, dem Ändern von Ablaufdatum und dem Widerrufen von Schlüsseln.
- (e) Benutzeroberfläche:
Die TUI sollte eine intuitive und benutzerfreundliche Schnittstelle für die Interaktion der Benutzer mit den PGP-Schlüsselverwaltungsfunktionen bieten. Dazu gehören Funktionen wie Navigationsmenüs, Eingabeaufforderungen und informative Anzeigen von Schlüsseldetails.

2. Nicht-funktionale Anforderungen

- (a) Sicherheit:
Die TUI sollte robuste Sicherheitsmaßnahmen implementieren, um sensible PGP-Schlüsselinformationen zu schützen.
 - (b) Verlässlichkeit:
Die TUI sollte zuverlässig und belastbar sein und die Integrität und Verfügbarkeit von PGP-Schlüsselverwaltungsfunktionen, wie z. B. die Fehlerbehandlung, gewährleisten.
-

Um die Interaktionen und Funktionalitäten der PGP Manager TUI zu verstehen, wird eine Anwendungsfallanalyse durchgeführt. Das Anwendungsfalldiagramm bietet eine visuelle Darstellung der funktionalen Anforderungen des Systems und der an seinem Betrieb beteiligten Akteure.

Das Anwendungsfalldiagramm für die PGP Manager TUI zeigt die verschiedenen Aktionen, die von den Benutzern (Akteuren) des Systems durchgeführt werden können, und die entsprechenden Systemfunktionen.

1. Akteure

Benutzer:

Stellt eine Person dar, die mit der PGP Manager TUI interagiert, um verschiedene Vorgänge im Zusammenhang mit der PGP-Schlüsselverwaltung auszuführen.

2. Anwendungsfälle

(a) Schlüssel-/Signatur Details abrufen:

Ermöglicht es dem Benutzer, einen PGP-Schlüssel oder eine Signatur zu prüfen und wichtige damit verbundene Details abzurufen. Zu den bereitgestellten Details gehören die Benutzer-ID(s), der Fingerabdruck, die Gültigkeit, die Schlüssellänge, die verwendete Chiffre und die Unterschlüssel.

(b) Schlüsselpaar generieren:

Ermöglicht es dem Benutzer, ein neues PGP-Schlüsselpaar entsprechend seinen spezifischen Anforderungen zu erzeugen. Die PGP Manager TUI ermöglicht es dem Benutzer, Schlüsselparameter wie Schlüsseltyp, Schlüssellänge, Verschlüsselungsalgorithmus und Ablaufzeit zu definieren. Durch die Generierung eines neuen Schlüsselpaares kann der Benutzer seine eigene, sichere und personalisierte kryptografische Identität erstellen.

(c) Öffentlichen Schlüssel exportieren:

Ermöglicht es dem Benutzer, den öffentlichen Schlüssel aus seinem Schlüsselpaar zu extrahieren. Der exportierte öffentliche Schlüssel kann an andere Benutzer weitergegeben werden, sodass diese mit dem öffentlichen Schlüssel des Benutzers Nachrichten verschlüsseln oder Signaturen überprüfen können. Diese Funktion erleichtert die sichere Kommunikation und schafft Vertrauen zwischen PGP-Benutzern.

(d) Passphrase bearbeiten:

Ermöglicht es dem Benutzer, seine Passphrase bei Bedarf zu ändern, um die Sicherheit seines privaten Schlüssels zu gewährleisten.

(e) Ablaufzeit bearbeiten:

Ermöglicht es dem Benutzer, die Ablaufzeit seines Schlüssels zu ändern. Die PGP Manager TUI ermöglicht es dem Benutzer, eine bestimmte Gültigkeitsdauer für seinen Schlüssel festzulegen, nach der er abläuft. Durch die Anpassung der Ablaufzeit kann der Benutzer die Kontrolle über den Lebenszyklus seines Schlüssels behalten.

(f) Geteilter öffentlicher Schlüssel:

Diese Funktion ist für den Fall gedacht, dass ein PGP-Schlüssel mit mehreren Identitäten verknüpft ist. Die PGP Manager TUI ermöglicht es Benutzern, einen öffentlichen Schlüssel mit mehreren Identitäten in mehrere öffentliche Schlüssel aufzuteilen, die jeweils denselben privaten Schlüssel haben. Mit dieser Funktion entfällt die Notwendigkeit, für jede Identität ein eigenes Schlüsselpaar zu erzeugen, was die Schlüsselverwaltung vereinfacht und die Benutzerfreundlichkeit erhöht.

- (g) UserID(s) hinzufügen:
Erlaubt dem Benutzer, zusätzliche UserID(s) zu einem bestehenden Schlüssel hinzuzufügen. Durch Hinzufügen neuer User IDs kann der Benutzer mehrere Identitäten mit seinem PGP-Schlüssel verknüpfen.
- (h) Schlüssel widerrufen:
Ermöglicht es dem Benutzer, den Widerrufsprozess für einen PGP-Schlüssel einzuleiten. Der Widerruf ist in Situationen notwendig, in denen der Schlüssel kompromittiert wurde oder nicht mehr gültig ist. Der Widerruf kann in zwei verschiedenen Szenarien durchgeführt werden:
 - i. Widerruf mit privatem Schlüssel und Passphrase:
Wenn der Benutzer Zugriff auf seinen privaten Schlüssel und seine Passphrase hat, kann er den Widerrufsprozess direkt mit dem privaten Schlüssel einleiten. Auf diese Weise wird sichergestellt, dass die Sperrung sicher vom Schlüsselinhaber durchgeführt wird.
 - ii. Widerruf mit öffentlichem Schlüssel und Widerrufszertifikat:
Für den Fall, dass der Benutzer die Passphrase seines privaten Schlüssels vergessen hat oder der private Schlüssel verloren oder gestohlen wurde, gibt es eine alternative Vorgehensweise. Der Benutzer kann den Schlüssel immer noch widerrufen, indem er einen öffentlichen Schlüssel mit einem Widerrufszertifikat zusammenführt, sofern ein solches vorhanden ist. Auf diese Weise kann der Benutzer den Schlüssel für ungültig erklären und seinen Widerrufsstatus anderen Benutzern und Systemen mitteilen.

Das Anwendungsfalldiagramm veranschaulicht die Interaktionen auf hoher Ebene zwischen den Akteuren und der Schnittstelle und erfasst die wesentlichen Funktionalitäten, die für die PGP-Schlüsselverwaltung erforderlich sind. Es dient als Grundlage für die anschließende Implementierungsphase, um sicherzustellen, dass die Fähigkeiten der Schnittstelle mit den Bedürfnissen der Benutzer übereinstimmen.

Es ist eine vereinfachte Darstellung und erfasst nicht alle möglichen Interaktionen oder Ausnahme-Szenarien. Weitere Einzelheiten zu den einzelnen Anwendungsfällen und den damit verbundenen Schritten werden im Abschnitt „Implementierung“ erläutert.

3.3 Komponentenbeschreibung

In diesem Abschnitt soll ein analytischer Einblick in die Architektur der Schnittstelle gegeben werden. Sein Design wurde so konzipiert, dass es die Prinzipien der Modularität und Skalierbarkeit berücksichtigt und somit eine einfache Wartung und Möglichkeiten für zukünftige Erweiterungen bietet.

Das Komponentendiagramm dient als visuelle Darstellung der verschiedenen Komponenten des Projekts und veranschaulicht ihre inhärenten Beziehungen und Interaktionen. Es dient als eine Art Bauplan, der ein umfassendes Verständnis der Architektur des Projekts vermittelt. Durch die Abbildung der detaillierten Struktur der Schnittstelle verdeutlicht das Komponentendiagramm die funktionale Struktur und zeigt auf, wie jedes Element zum Betrieb des Ganzen beiträgt. Diese detaillierte Abbildung ermöglicht ein tieferes Verständnis der Projektkomplexität und ist damit ein wichtiges Instrument für das Verständnis und die Entwicklung komplexer Programme. Sie zeigt nicht nur die „Bausteine“ des Projekts, sondern auch, wie diese Bausteine miteinander verbunden sind und zusammenwirken. Somit ist sie ein unverzichtbares Hilfsmittel, um die gesamte Architektur des Projekts in ihrer Breite und Tiefe zu erfassen.

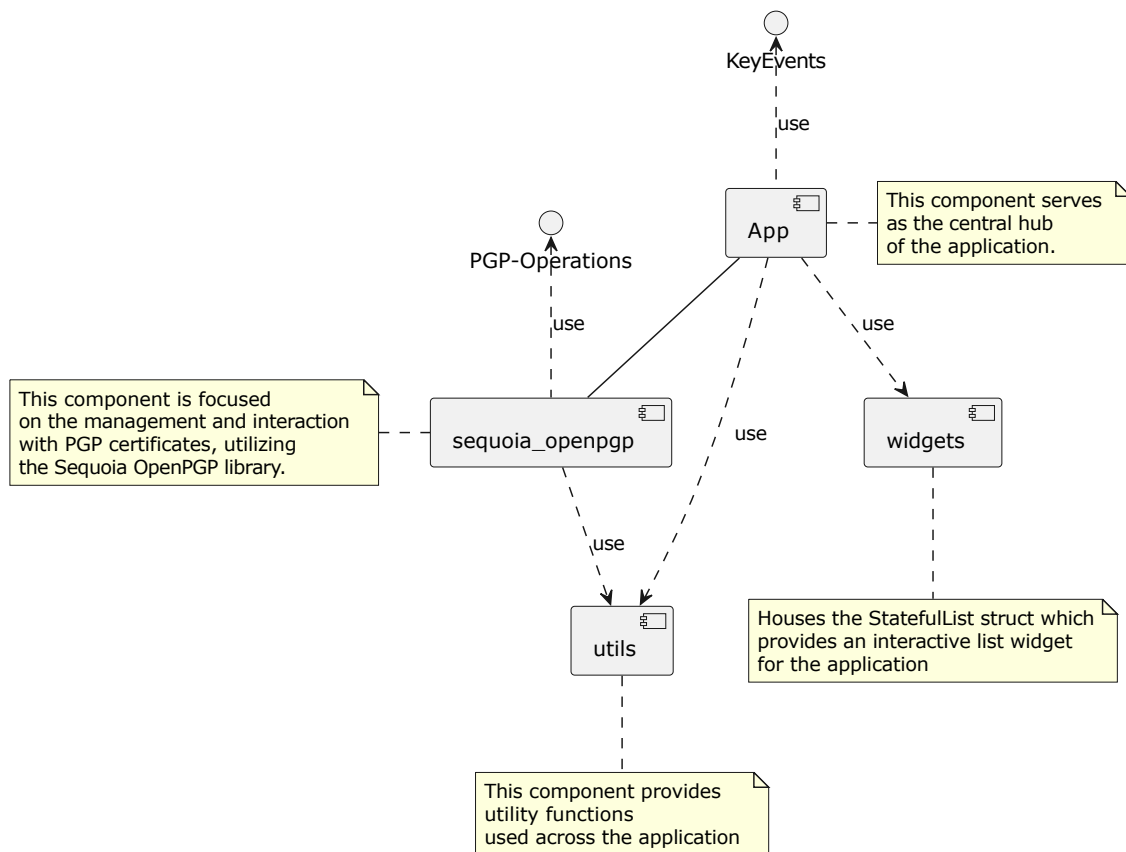


Abbildung 3.2: Komponentendiagramm

Die Architektur der PGP Manager TUI besteht aus vier Hauptmodulen:

1. Das App-Modul:

Dieses Modul dient als Brücke zwischen der Benutzeroberfläche und den Operationen, die der PGP-Schlüsselverwaltung entsprechen. Das Modul ist für die Interpretation der Benutzereingaben und die Ausführung der entsprechenden Funktionsaufrufe verantwortlich.

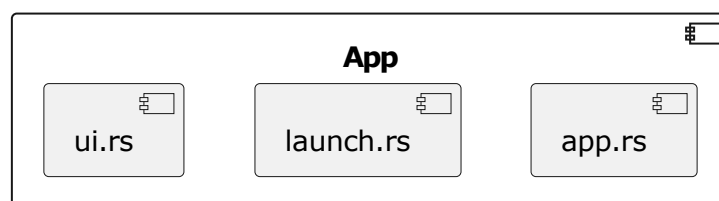


Abbildung 3.3: Das App-Modul

(a) app.rs

Die App-Struktur stellt die zentrale Datenstruktur der PGP Manager TUI dar. Sie kapselt verschiedene Felder, die den aktuellen Zustand und die Benutzereingaben innerhalb der Anwendung speichern.

(b) ui.rs

Eine Reihe von Funktionen, die zusammenarbeiten, um die interaktive und visuell ansprechende Benutzeroberfläche des PGP Manager TUI zu schaffen, die es den Be-

nutzern ermöglicht, in Verzeichnissen zu navigieren, Schlüsseldetails anzuzeigen und verschiedene Operationen mit PGP-Schlüsseln durchzuführen.

(c) `launch.rs`

Es ist die Kernkomponente, die für die Ausführung der PGP Manager TUI und die Verarbeitung von Benutzereingaben verantwortlich ist und es den Benutzern ermöglicht, verschiedene Operationen mit Schlüsseln auf interaktive Weise durchzuführen.

2. Das Sequoia-OpenPGP-Modul:

Dieses Modul nutzt die von der Sequoia-OpenPGP crate bereitgestellten Methoden, um die benötigten Funktionen zu implementieren.

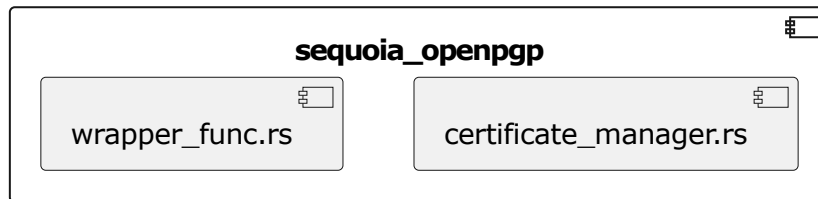


Abbildung 3.4: Das Sequoia-OpenPGP-Modul

(a) `certificate_manager.rs`

Es bietet eine Möglichkeit, verschiedene Schlüsseloperationen durchzuführen und die Ergebnisse in strukturierter Weise zu verarbeiten.

(b) `wrapper_func.rs`

Die Wrapper-Funktionen bieten eine benutzerfreundliche Schnittstelle für die Interaktion mit dem CertificateManager und die Durchführung verschiedener Aufgaben.

3. Das Utils-Modul:

Dieses Modul stellt die notwendigen Utility-Funktionen bereit, die im gesamten System benötigt werden. Es ermöglicht Operationen wie die Erstellung von Verzeichnissen und Dateien, die Extraktion von Benutzern, die Auflistung von Verzeichnisinhalten und das Parsen der ISO8601-Dauer.

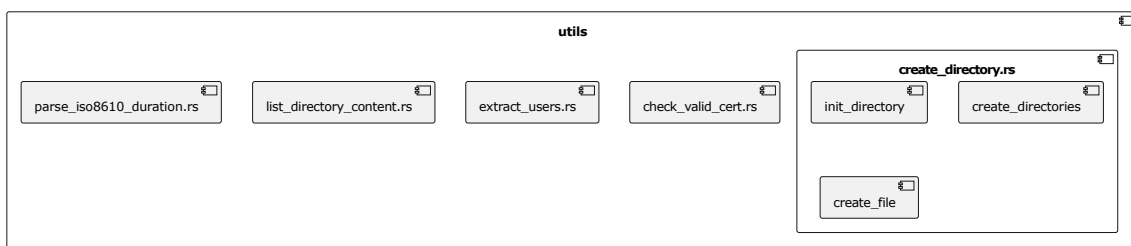


Abbildung 3.5: Das Utils-Modul

(a) `list_directory_content.rs`

Eine Hilfsfunktion, die den Inhalt eines durch den angegebenen Pfad spezifizierten Verzeichnisses auflistet.

(b) `check_valid_cert.rs`

Eine Hilfsfunktion, die eine Reihe von Prüfungen an der Datei durchführt, um den

Typ der Datei zu bestimmen. Sie hilft bei der Unterscheidung zwischen Schlüsseln und Signaturen.

(c) `extract_users.rs`

Eine Hilfsfunktion, die Benutzer aus einem Schlüssel ausliest. Sie wird für das Split-Feature nützlich sein.

(d) `create_directory.rs`

i. `create_directories`

Eine Hilfsfunktion, die die angegebenen Verzeichnisse und alle fehlenden übergeordneten Verzeichnisse erstellt.

ii. `init_directory`

Eine Hilfsfunktion, die die Hauptverzeichnisstruktur für die Schnittstelle initialisiert.

iii. `create_file`

Eine Hilfsfunktion, die für die Erstellung einer Datei zum Schreiben zuständig ist.

(e) `parse_8610_duration.rs`

Eine Hilfsfunktion, die es ermöglicht, ISO 8601-Duration-Strings zu analysieren und in ein Duration-Objekt zu konvertieren, das eine Zeitdauer darstellt.

4. Das Widgets-Modul:

Dieses Modul umfasst ein wiederverwendbares Schnittstellenelement oder „Widget“, das eine Interaktion zwischen dem Benutzer und der Anwendung ermöglicht.



Abbildung 3.6: Das Widgets-Modul

(a) `list.rs`

Die `StatefullList`-Struktur verfolgt eine Liste von Elementen und verwaltet den Status des aktuell ausgewählten Elements sowie den Auswahlstatus jedes Elements in der Liste.

Die in dieser Struktur bereitgestellten Methoden ermöglichen die Manipulation und Abfrage einer Liste mit Auswahlstatus, was in einer Terminal-Benutzerschnittstelle sehr nützlich sein kann, wo Benutzer durch eine Liste navigieren und Elemente auswählen kann.

Der modulare Aufbau fördert die Aufrechterhaltung einer klaren Trennung der Zuständigkeiten und erhöht gleichzeitig die Skalierbarkeit des Systems. Diese Flexibilität ermöglicht die Hinzufügung weiterer Module oder die Erweiterung bestehender Module, um in Zukunft weitere Features einzubauen.

Kapitel 4

Implementierung

In diesem Kapitel werden die technischen Aspekte des Projekts beleuchtet und die Umsetzung der entwickelten Lösung beschrieben. Es bietet einen Einblick in den Code, die Funktionalität und die Struktur der implementierten Komponenten.

Bevor auf die Einzelheiten der Implementierung der Systemkomponenten eingegangen wird, ist es wichtig, auf die Auswahl der Werkzeuge und Bibliotheken einzugehen, die sorgfältig geprüft wurden. Der Auswahlprozess umfasste eine Bewertung verschiedener Optionen, um sicherzustellen, dass die gewählten Tools mit den Zielen und Anforderungen des Projekts übereinstimmen.

Im Zusammenhang mit dem PGP-Schlüsselverwaltungssystem drehte sich die Wahl der Tools um zwei wichtige Aspekte: die Programmiersprache und die zu verwendenden Bibliotheken.

Bei der Auswahl von Rust als Programmiersprache für dieses Projekt wurden mehrere Faktoren berücksichtigt. Die Betonung von Rust auf Speichersicherheit, Nebenläufigkeit und Leistung machte sie zu einer geeigneten Wahl für die Implementierung eines sicheren und effizienten PGP-Schlüsselverwaltungssystems. Die umfangreichen Compilerprüfungen der Sprache, gaben Vertrauen in die Korrektheit und Zuverlässigkeit des Codes. Darüber hinaus sorgte die lebendige und aktive Community von Rust zusammen mit ihrem wachsenden Ökosystem von Bibliotheken und Tools für reichlich Unterstützung und Ressourcen für die Entwicklung.

Bei den Bibliotheken wurde die `sequoia-openpgp-crate` ausgewählt, um die notwendigen Funktionalitäten für die PGP-Schlüsselmanipulation bereitzustellen. Ausschlaggebend für diese Entscheidung waren unter anderem die Einhaltung des OpenPGP-Standards, die aktive Entwickler-Community und das umfangreiche Feature der Bibliothek. Alternativen wie die `gpgme-rs-crate`, die die Interaktion mit der GPGME-Bibliothek (GnuPG Made Easy) vereinfacht, wurden ebenfalls in Betracht gezogen, erwiesen sich aber als weniger geeignet für die spezifischen Anforderungen des Systems.

Die Wahl der `sequoia-openpgp-crate` gewährleistet die Kompatibilität und erleichtert die Interoperabilität. Die Dokumentation und die Unterstützung durch die Community trugen zu einem effizienten Entwicklungsprozess bei und halfen bei der Lösung von Problemen, die bei der Implementierung auftraten.

Schließlich wurde die `tui-rs-crate` ausgewählt, da sie ein leistungsstarkes und flexibles Toolkit für die Erstellung terminalbasierter Benutzeroberflächen in Rust bietet. Die Entscheidung, `tui-rs` zu verwenden, basierte auf seiner Simplität, Erweiterbarkeit und plattformübergreifenden Kompatibilität. Die Übernahme der Bibliothek ermöglichte die Erstellung einer intuitiven und interaktiven Terminal-Benutzeroberfläche.

4.1 Umsetzung der OpenPGP-Funktionen

In diesem Abschnitt wird nur ein Teil des Quellcodes besprochen, da der komplette Code aufgrund seiner Länge nicht vollständig dargestellt werden kann. Die besprochenen Codefragmente

dienen der Veranschaulichung wichtiger Funktionen und Abläufe. Für eine vollständige und detaillierte Untersuchung aller Aspekte und Funktionen, die in diesem Programm implementiert sind, ist es empfehlenswert, den vollständigen Quellcode zu konsultieren. Dieser enthält weitere Informationen und Zusammenhänge, die für ein vollständiges Verständnis der Funktionsweise des Codes wichtig sind.

4.1.1 Schlüssel-/Signatur Informationen abrufen:

Diese Funktionalität ist in zwei Funktionen aufgeteilt, von denen die eine Informationen über einen Schlüssel und die andere über eine eigenständige Signatur zurückgibt.

```
fn get_key_details(&self, cert_path: &String) ->
Result<String, anyhow::Error> {
    let cert = Cert::from_reader(BufReader::new(File::open(cert_path)?));
    let mut key_details = String::new();
    // check if public key or secret key
    if cert.is_tsk() {
        key_details.push_str(&format!("\nThis_is_a_Secret_key\n"));
    } else {
        key_details.push_str(&format!("\nThis_is_a_Public_key\n"));
    }
    // Rest of the code
    Ok(key_details)
}
```

Listing 4.1: get_key_details Funktion

Die Funktion `get_key_details` extrahiert und konsolidiert Informationen aus einem digitalen Zertifikat, das über den angegebenen Schlüsselpfad zugänglich ist. Nach dem Initialisieren und Lesen der Datei bestimmt die Funktion den Schlüsseltyp, ruft Benutzer-ID(s), Signatur(en), Attribut(e), den Fingerabdruck und den Widerrufsstatus ab und prüft die Gültigkeit. Außerdem werden Angaben zum Erstellungs- und Ablaufzeitpunkt ermittelt. Schließlich überprüft die Funktion die Fähigkeiten und Unterschlüssel des Zertifikats. Am Ende gibt die Funktion `key_details` zurück, eine Zeichenkette mit detaillierten Informationen über das Zertifikat. Es ist zu beachten, dass diese Funktion an mehreren Stellen fehlschlagen und einen Fehler zurückgeben kann, insbesondere beim Öffnen der Datei, beim Lesen des Zertifikats und bei der Validierung des Schlüssels. Die Funktion verwendet den Operator `?`, um Fehler weiterzuleiten, die später in der Terminal-Benutzeroberfläche behandelt werden.

```
fn get_signature_details(&self, signature_path: &String) ->
Result<String, anyhow::Error> {
    let mut revocation_details = String::new();

    let file = File::open(signature_path)
        .context("Failed_to_open_the_file");
    let mut reader = BufReader::new(file);
    let packet_pile = PacketPile::from_reader(&mut reader)
        .context("Failed_to_read_the_signature_from_the_file");

    let signature = packet_pile.descendants().find_map(|packet| {
        if let Packet::Signature(sig) = packet {
            Some(sig.clone())
        } else {

```

```

        None
    }
});
// Rest of the code
Ok(revocation_details)
}

```

Listing 4.2: get_signature_details Funktion

Die Funktion `get_signature_details` dient dazu, spezifische Informationen aus einer digitalen Signatur zu extrahieren, die sich unter einem bestimmten Pfad befindet. Der Prozess beginnt mit der Initialisierung einer leeren Zeichenkette zur Speicherung von Informationen und dem anschließenden Öffnen der Datei unter dem angegebenen Pfad. Nach dem Öffnen wird die Signatur von einem gepufferten Reader in einen `PacketPile` gelesen, eine Struktur zur Darstellung einer unstrukturierten Paketfolge. Die Funktion sucht dann nach der ersten Signatur und extrahiert relevante Informationen wie Erstellungszeit, Typ und Fingerabdruck des Ausstellers, die dann an den Detailstring angehängt werden. Treten an irgendeiner Stelle des Prozesses Fehler auf, z. B. weil keine Signatur gefunden wurde oder die Datei nicht geöffnet werden konnte, wird eine Fehlermeldung zurückgegeben. Andernfalls wird die Detailzeichenkette als erfolgreiches Ergebnis zurückgegeben.

4.1.2 Schlüsselpaar generieren:

```

pub fn generate_keypair(
    &self,
    user_id: String,
    validity: String,
    cipher: String,
    pw: String,
    rpw: String,
) -> Result<(), anyhow::Error> {
    let mut builder = CertBuilder::new();
    let uid_clone = user_id.clone();

    builder = builder.add_userid(user_id);

    match validity.as_str() {
        "n" | "N" => {
            builder = builder
                .set_creation_time(StdTime::now())
                .set_validity_period(None);
        }
        "0" | "" => {
            // setting default validity period to 2 years
            builder = builder
                .set_creation_time(StdTime::now())
                .set_validity_period(Some(
                    std::time::Duration::new(2 * 31536000, 0)
                ));
        }
        _ => {
            let duration = parse_iso8601_duration(validity.as_str()).unwrap();

```

```

        builder = builder
            .set_creation_time(StdTime::now())
            .set_validity_period(Some(duration));
    }
}
// Rest of the code
Ok(())
}

```

Listing 4.3: generate_key_pair Funktion

Die Funktion `generate_keypair` dient dazu, ein neues OpenPGP-Schlüsselpaar zu erzeugen, das aus einem privaten und einem öffentlichen Schlüssel sowie einem Widerrufszeugnis besteht. Zu den Eingabeparametern gehören Benutzer-ID, Gültigkeit, Chiffre, Passwort und Passwortwiederholung. Zunächst wird eine Instanz des `CertBuilders` erzeugt, der Methoden zur Konfiguration eines neuen OpenPGP-Zertifikats bereitstellt. Dann werden die Benutzer-ID, die Gültigkeitsdauer (abhängig von den übergebenen Parametern), die Cipher-Suite und das Passwort angegeben. Anschließend fügt die Funktion dem Zertifikat vier Unterschlüssel hinzu: einen für die Signierung, einen für die Authentifizierung, einen für die Speicherverschlüsselung und einen für die Transportverschlüsselung. Anschließend werden das Schlüsselpaar und das Widerrufszeugnis in Dateien im Verzeichnis `.pgpman/secrets` bzw. `.pgpman/revocation` im Home-Verzeichnis des Benutzers exportiert. Die Dateinamen entsprechen der BenutzerID.

4.1.3 Öffentlichen Schlüssel exportieren:

```

pub fn export_certificate(
    &self,
    cert_path: &String,
    ex_file_name: &String,
) -> Result<(), anyhow::Error> {
    let cert = Cert::from_reader(BufferedReader::new(File::open(cert_path)))?;
    let home_dir = home::home_dir().unwrap();
    let export_path = String::from(format!(
        "{}/.pgpman/certificates/{}",
        &home_dir.display(),
        ex_file_name
    ));
    if let Some(mut out_cert) = create_file(Some(&export_path.as_str()))? {
        cert.armor().serialize(&mut out_cert)?;
    }

    Ok(())
}

```

Listing 4.4: export_certificate Funktion

Die Funktion `export_certificate` versucht, ein Schlüsselpaar aus einem Pfad zu lesen und es in einem ASCII-armored Format in eine andere Datei zu schreiben, die der extrahierte öffentliche Schlüssel ist, den der Benutzer weitergeben kann. Sie gibt ein Ergebnis vom Einheitstyp `Result<(), anyhow::Error>` zurück, was bedeutet, dass sie keine nützlichen Daten zurückgibt, sondern nur Erfolg oder Misserfolg anzeigt.

4.1.4 Passphrase bearbeiten:

```

pub fn edit_password(
    &self,
    secret_key: &String,
    original: &Password,
    npw: String,
    rnpw: String,
) -> Result<(), anyhow::Error> {
    let mut key = Cert::from_reader(
        BufReader::new(File::open(&secret_key)?))?;
    // decrypt secret first
    let secret = key.primary_key().key().clone().parts_into_secret()?;
    let dec = secret.decrypt_secret(&original)?;
    key = key.insert_packets(dec)?;
    if npw == rnpw {
        let enc = key
            .primary_key()
            .key()
            .clone()
            .parts_into_secret()?
            .encrypt_secret(&rnpw.into())?;
        key = key.insert_packets(enc)?;

        if let Some(mut out) = create_file(Some(&secret_key))? {
            key.as_tsk().armored().serialize(&mut out)?;
        }
    }
    Ok(())
}

```

Listing 4.5: edit_password Funktion

Die Funktion `edit_password` ändert das Kennwort eines privaten PGP-Schlüssels, indem sie den Schlüssel mit dem ursprünglichen Kennwort entschlüsselt, ihn dann mit einem neuen Kennwort verschlüsselt und die ursprüngliche Datei aktualisiert. Sie holt sich dann den Primärschlüssel des Zertifikats, kloniert ihn, wandelt ihn in einen privaten Schlüssel um und entschlüsselt ihn mit dem ursprünglichen Passwort. Das entschlüsselte Kennwort wird dann wieder in das Zertifikat eingefügt. Wenn das neue Kennwort (`npw`) mit dem wiederholten neuen Kennwort (`rnpw`) übereinstimmt, verschlüsselt die Funktion den geheimen Schlüssel mit dem neuen Kennwort und fügt ihn wieder in das Zertifikat ein. Falls die Datei an dem durch `secret_key` angegebenen Pfad erfolgreich zum Schreiben geöffnet werden kann, wird der aktualisierte private Schlüssel in die Datei im ASCII-armored Format serialisiert.

4.1.5 Ablaufzeit bearbeiten:

```

pub fn edit_expiration_time(
    &self,
    cert_path: &String,
    original: &Password,
    validity: String,
) -> Result<(), anyhow::Error> {
    let cert = Cert::from_reader(BufReader::new(File::open(&cert_path)?))?;
    let p = &StandardPolicy::new();

```



```

let vc = cert.with_policy(p, None)?;
let sig;

let secret = vc.primary_key().key().clone().parts_into_secret()?;
let mut keypair = secret.decrypt_secret(&original)?.into_keypair()?;

match validity.as_str() {
    "0" | "" => {
        sig = vc.primary_key().set_expiration_time(&mut keypair, None)?;
    }
    _ => {
        let duration = parse_iso8601_duration(
            &validity.as_str()).unwrap();
        let t = time::SystemTime::now() + duration;
        sig = vc
            .primary_key()
            .set_expiration_time(&mut keypair, Some(t))?;
    }
}

let cert = cert.insert_packets(sig)?;
if let Some(mut out_cert) = create_file(Some(&cert_path))? {
    cert.as_tsk().armored().serialize(&mut out_cert)?;
}
Ok(())
}

```

Listing 4.6: edit_expiration_time Funktion

Die Funktion `edit_expiration_time` ist für die Anpassung der Verfallszeit eines Schlüsselpaares zuständig. Sie entschlüsselt den Primärschlüssel des Zertifikats mit dem angegebenen ursprünglichen Passwort, um ein Schlüsselpaar zu erzeugen, das für Signiervorgänge verwendet werden kann. Je nach dem an die Funktion übergebenen Gültigkeitsargument wird entweder die Verfallszeit des Zertifikats entfernt oder eine neue gesetzt. Nachdem die Ablaufzeit angepasst wurde, fügt die Funktion die aktualisierte Signatur wieder in das Zertifikat ein. Das aktualisierte Zertifikat wird dann in einem ASCII-armored Format zurück in die Datei serialisiert, aus der es gelesen wurde.

4.1.6 Geteilter öffentlicher Schlüssel:

```

pub fn split_users(
    &self,
    cert_path: &String,
    new_cert_name: &String,
    selected_users: Vec<String>,
) -> Result<(), anyhow::Error> {
    let cert = Cert::from_reader(
        BufReader::new(File::open(cert_path)?))?;
    let home_dir = home::home_dir().unwrap();
    let export_path = String::from(format!(
        "{}/.pgpman/certificates/{}",
        &home_dir.display(),
    ));
}

```

```

        new_cert_name
    ));

    let cp = cert.clone().retain_userids(|ua| {
        if let Ok(Some(address)) = ua.email() {
            selected_users.contains(&address)
        } else {
            true
        }
    });
    if let Some(mut out_cert) = create_file(Some(&export_path.as_str()))? {
        cp.serialize(&mut out_cert).unwrap();
    }

    Ok(())
}

```

Listing 4.7: split_users Funktion

Die Funktion `split_users` wird verwendet, um aus einem gegebenen öffentlichen Schlüssel ein separater öffentlicher Schlüssel für die ausgewählten Benutzer in einer bestimmten Liste zu erzeugen. Das geklonte Zertifikat wird mit der Methode `retain_userids` beschnitten, um nur die Benutzer-IDs zu behalten, die in der Liste `selected_users` enthalten sind. Dabei wird überprüft, ob die E-Mail-Adresse jeder Benutzer-ID in der Liste `selected_users` enthalten ist. Das beschnittene Zertifikat (`cp`) wird dann in eine Datei mit einem Namen (`new_cert_name`) in einem angegebenen Verzeichnis serialisiert. Wenn die Funktion ohne Probleme ausgeführt wird, gibt sie `Ok(())` zurück, was den Erfolg anzeigt. Wenn während des Prozesses Fehler auftreten, werden diese zurückgegeben.

4.1.7 BenutzerID hinzufügen:

```

pub fn add_user(
    &self,
    cert_path: &String,
    original: &Password,
    new_userid: String,
) -> Result<(), anyhow::Error> {
    let mut key = Cert::from_reader(
        BufReader::new(File::open(&cert_path)?))?;
    if !key.is_tsk() {
        Err(anyhow::anyhow!("This_is_not_a_secret_key"))?;
    }
    // decrypt secret
    let secret = key.primary_key().key().clone().parts_into_secret()?;

    let mut keypair = secret.decrypt_secret(&original)?.into_keypair()?;
    let new_userid = UserID::from(new_userid);
    let builder = signature::SignatureBuilder::new(
        SignatureType::PositiveCertification);
    let binding = new_userid.bind(&mut keypair, &key, builder)?;
    // Now merge the User ID and binding signature into the Cert.
    key = key.insert_packets(vec![Packet::from(new_userid), binding.into()]);
}

```

```

    if let Some(mut out) = create_file(Some(&cert_path))? {
        key.as_tsk().armored().serialize(&mut out)?;
    }

    Ok(())
}

```

Listing 4.8: add_user Funktion

Die Funktion `add_user` wird verwendet, um einen neuen Benutzer zu einem PGP-Zertifikat hinzuzufügen. Sie beginnt mit dem Lesen eines Zertifikats aus einer durch `cert_path` angegebenen Datei. Das Zertifikat wird in einen privaten Schlüssel mit dem ursprünglichen Kennwort entschlüsselt und ein Schlüsselpaar wird erstellt. Anschließend erstellt die Funktion eine neue `UserID`-Instanz aus der angegebenen Zeichenkette `new_userid`. Sie bindet diese neue `UserID` an das Zertifikat und erstellt dabei eine Signatur. Diese Signatur ist eine positive Zertifizierung, d. h. eine Signatur, die besagt, dass der Unterzeichner von der Gültigkeit der `UserID` überzeugt ist. Die neu erstellte Benutzer-ID und die verbindliche Signatur werden dann in das bestehende Zertifikat eingefügt. Sobald das Zertifikat mit den neuen Benutzerinformationen aktualisiert ist, wird es in einem ASCII-armored Format zurück in die Datei serialisiert.

4.1.8 Widerrufen:

```

pub fn revoke_key(
    &self,
    cert_path: &String,
    original: &Password,
    reason: &String,
) -> Result<(), anyhow::Error> {
    let key = Cert::from_reader(BufReader::new(File::open(&cert_path)?));
    let rev: Signature;

    let secret = key.primary_key().key().clone().parts_into_secret();

    let mut signer = secret.decrypt_secret(&original)?.into_keypair();

    match reason.as_str() {
        "1" => {
            rev = key.revoke(
                &mut signer,
                ReasonForRevocation::KeyRetired,
                b"Key_is_retired",
            )?
        }
        "2" => {
            rev = key.revoke(
                &mut signer,
                ReasonForRevocation::KeySuperseded,
                b"Key_is_superseded",
            )?
        }
        "3" => {

```

```

        rev = key.revoke(
            &mut signer,
            ReasonForRevocation::KeyCompromised,
            b"Key_is_compromised",
        )?
    }
    _ => {
        rev = key.revoke(
            &mut signer,
            ReasonForRevocation::Unspecified,
            b"No_reason_specified",
        )?
    }
}

let key = key.insert_packets(rev)?;
if let Some(mut out) = create_file(Some(&cert_path))? {
    key.as_tsk().armored().serialize(&mut out)?;
}

Ok(())
}

```

Listing 4.9: revoke_key Funktion

Die Funktion `revoke_key` wird verwendet, um ein PGP-Zertifikat aus einem bestimmten Grund zu widerrufen. Wie im Entwurfskapitel erwähnt, ist dies nur möglich, wenn der private Schlüssel existiert und das Passwort bekannt ist. Es liest zunächst ein PGP-Zertifikat aus dem angegebenen Dateipfad (`cert_path`) und kloniert daraus den Primärschlüssel. Dann entschlüsselt es den privaten Schlüssel mit dem angegebenen Passwort (`original`). Der Schlüssel wird auf der Grundlage des angegebenen Grundes (`reason`) widerrufen. Dieser Grund kann sein, dass der Schlüssel nicht mehr gültig ist, dass er ersetzt wurde oder dass er kompromittiert wurde. Nach dem Widerruf fügt die Funktion die Widerrufssignatur wieder in das Zertifikat ein und schreibt das aktualisierte Zertifikat zurück in die Datei.

```

pub fn revoke_certificate(
    &self,
    cert_path: &String,
    rev_cert: &String,
) -> Result<(), anyhow::Error> {
    let cert = Cert::from_reader(
        BufReader::new(File::open(&cert_path)?))?;

    let pile = PacketPile::from_reader(
        BufReader::new(File::open(&rev_cert)?))?;
    // extract packets from the revocation certificate
    let packets: Vec<Packet> = pile.into();

    // if there is no signature packet in the revocation certificate
    // return an error
    let revcert_signature = packets
        .iter()
        .find_map(|packet| match packet {

```

```

        Packet::Signature(sig) => Some(sig),
        _ => None,
    })
    .ok_or(anyhow::anyhow!(
        "No_signature_packet_in_revocation_certificate"
    ))?;

    // Get the issuers of the revocation signature.
    let revcert_issuers = revcert_signature.get_issuers();

    // If none of the issuers of the revocation signature match any
    // key in the original certificate, return an error.
    if !cert.keys().any(|key| {
        revcert_issuers
            .iter()
            .any(|issuer| *issuer == key.key_handle())
    }) {
        return Err(anyhow::anyhow!(
            "Revocation_certificate_does_not_match_original_certificate"
        ));
    }
    let cert = cert.insert_packets(packets.into_iter());

    if let Some(mut out) = create_file(Some(&cert_path))? {
        cert.armored().serialize(&mut out)?;
    }
    Ok(())
}

```

Listing 4.10: revoke_certificate Funktion

Die Funktion `revoke_certificate` widerruft einen öffentlichen PGP-Zertifikat auf der Grundlage eines bereitgestellten Widerrufs-zertifikats (`rev_cert`). Sie liest zunächst das Original-zertifikat und das Widerrufs-zertifikat aus ihren jeweiligen Dateipfaden. Dann prüft sie, ob das Widerrufs-zertifikat mit dem ursprünglichen Zertifikat übereinstimmt, indem sie die Aussteller der Widerrufs-signatur mit den Schlüsseln im ursprünglichen Zertifikat vergleicht. Wenn es keine Übereinstimmung gibt, wird ein Fehler zurückgegeben. Gibt es eine Übereinstimmung, werden die Widerrufs-pakete in das ursprüngliche Zertifikat eingefügt. Das aktualisierte Zertifikat wird dann wieder in die Datei serialisiert.

4.1.9 Die Implementierung des CertificateManager:

```

pub struct CertificateManager;
pub enum CertificateOperation {
    GetKeyDetails,
    GetSignatureDetails,
}
pub enum CertificateOperationOutput {
    Details(Result<String, anyhow::Error>),
    Result(Result<(), anyhow::Error>),
}
impl CertificateManager {

```

```

pub fn execute(
    &self,
    operation: CertificateOperation,
    cert_path: Option<&String>,
    file_name: Option<&String>,
) -> Result<CertificateOperationOutput, anyhow::Error> {
    match operation {
        CertificateOperation::GetKeyDetails => {
            if let Some(cert_path) = cert_path {
                Ok(CertificateOperationOutput::Details(
                    self.get_key_details(cert_path),
                ))
            } else {
                Err(anyhow!(
                    "A_file_path_is_required_for_GetKeyDetails_operation"
                ))
            }
        }
    }
}
// rest of the code
}
}
}

```

Listing 4.11: CertificateManager Implementierung

Dieser Code stellt eine CertificateManager-Struktur und eine zugehörige Aufzählung (enum) CertificateOperation dar, die die verschiedenen Vorgänge beschreibt, die mit diesem Manager durchgeführt werden können. Die Aufzählung CertificateOperationOutput erfasst die mögliche Ausgabe dieser Operationen, d.h. entweder einige Informationen, die als Zeichenkette zurückgegeben werden, oder einfach den Erfolg oder Misserfolg der Operation.

Die CertificateManager-Struktur enthält keine Felder, sodass Instanzen dieser Struktur keine Daten enthalten. Ihr Zweck ist es, die zugehörigen Funktionen in einem Namensraum unterzubringen. Diese Funktionen sind die zuvor erwähnten Funktionen und werden als Methoden bezeichnet, da sie Teil der Implementierung der Struktur sind. Die Ausführungsmethode (execute) ist die primäre Methode von CertificateManager. Sie nimmt eine Instanz von CertificateOperation und einige optionale Parameter entgegen und führt dann die entsprechende Operation aus. Diese Funktion macht ausgiebigen Gebrauch von Rusts match-Schlüsselwort, um jede Variante von CertificateOperation einzeln zu behandeln und sicherzustellen, dass die notwendigen Vorbedingungen für jede Operation erfüllt sind, bevor die Operation ausgeführt wird. Der self-Parameter, der in diesen Methoden zu sehen ist, ist ein spezieller Parameter in Rust, der die Instanz der Struktur repräsentiert, mit der die Methode aufgerufen wird. Es ist mit "this" in einigen anderen objektorientierten Programmiersprachen vergleichbar. In diesen Methoden wird &self verwendet, was eine Referenz auf self ist und es den Methoden erlaubt, die Daten in der Struktur zu lesen, ohne den Besitz (ownership) zu übernehmen. Da CertificateManager keine Daten enthält, wird der self-Parameter in erster Linie für das Namespacing verwendet und hat keine praktischen Auswirkungen.

4.2 Umsetzung der Terminal-Benutzeroberfläche

Nach erfolgreicher Implementierung der wesentlichen Funktionalitäten wird der Schwerpunkt auf die Entwicklung der Benutzeroberfläche des Terminals konzentriert. Diese Untersuchung beinhaltet ein tiefes Verständnis der Prozesse, die beim Rendern der Schnittstelle involviert sind,

eine gründliche Prüfung der grundlegenden Komponenten der Benutzerschnittstelle.

4.2.1 Den Zustand einer Liste verwalten

```
pub struct StatefulList<T> {
    pub items: Vec<T>,
    pub state: ListState,
    pub selected_items: Vec<bool>,
}
impl<T> StatefulList<T> {
    pub fn new(items: Vec<T>, state: ListState, selected_items: Vec<bool>) ->
        StatefulList<T> {
        Self {
            items,
            state,
            selected_items,
        }
    }
    // Rest of the code
}
```

Listing 4.12: StatefulList Implementierung

Die Struktur `StatefulList<T>` bietet eine effiziente Möglichkeit, eine dynamische Liste von Elementen mit integrierten Auswahlmöglichkeiten zu verwalten. Diese Struktur umfasst einen Vektor von Elementen (`Vec<T>`), eine Instanz von `ListState`, die das aktuell ausgewählte Element verfolgt, und ein `Vec<bool>`, das den Auswahlzustand jedes Listenelements angibt. Zu den wesentlichen Methoden gehören der Konstruktor `new`, die statische Methode `with_items` zum Aufbau einer `StatefulList<T>` mit einer anfänglich unselektierten Liste von Elementen und die Methode `selected` zum Abrufen des aktuell ausgewählten Elements. Die Methoden `next` und `previous` ermöglichen die Navigation durch die Liste, indem sie das nächste bzw. vorherige Element auswählen. Schließlich steht die Methode `unselect` zur Verfügung, um die Auswahl des aktiven Elements aufzuheben. `StatefulList<T>` ist aufgrund seiner Flexibilität und Anpassungsfähigkeit an jeden Typ `T` nützlich und erleichtert die Handhabung von Listen mit verschiedenen auswählbaren Elementen in unterschiedlichen Kontexten. Diese Fähigkeit ist vorteilhaft für die Verwaltung von Benutzeroberflächenelementen, bei denen Benutzer eine Reihe von Optionen auswählen oder durch sie navigieren können, oder in jedem Szenario, das die Aufrechterhaltung eines aktiven Zustands innerhalb einer Sammlung von Elementen erfordert.

4.2.2 Implementierung der App-Struktur

```
pub struct App {
    pub items: StatefulList<String>,
    pub key_details: Option<String>,
}
```

Listing 4.13: App Struktur

Die „App“ Struktur kapselt den Zustand und das Verhalten der Terminal-Benutzeroberfläche. Diese Struktur enthält Felder für die Verwaltung der Liste der anzuzeigenden Elemente (Schlüsseldateien oder Verzeichnisse), die Details des aktuell ausgewählten Schlüssels, Informationen über Popups und Scrollen, das aktuell erkundete Verzeichnis, Exportpfade, Benutzeridentifikation, Gültigkeits- und Verschlüsselungsdetails, Passwortinformationen, Benutzerauswahlen, Hilfedetails und Widerrufsdetails. Die App-Struktur enthält einen Implementierungsblock mit einer

neuen Methode, die ein üblicher Weg ist, um eine Instanz einer Struktur in Rust zu erstellen. Die neue Methode nimmt einen PathBuf, der das aktuelle Verzeichnis repräsentiert, und einen Vec<String>, der die Liste der anzuzeigenden Elemente enthält. Sie initialisiert eine neue App-Instanz mit diesen Werten und verschiedenen anderen Standardeinstellungen.

4.2.3 Komponenten der Benutzerschnittstelle

```
pub fn draw_input_prompt<B: Backend>(
    terminal: &mut Terminal<B>,
    prompt: &[Spans],
    display_input: bool,
) -> Result<String, anyhow::Error> {
    let mut input = String::new();
    loop {
        terminal.draw(|f| {
            let chunks = Layout::default()
                .direction(Direction::Vertical)
                .margin(2)
                .constraints([Constraint::Percentage(0),
                    Constraint::Length(3)].as_ref())
                .split(f.size());
            let input_text = if display_input {
                input.clone()
            } else {
                "*".repeat(input.len())
            };
            let input_widget = Paragraph::new(prompt_text)
                .block(
                    Block::default()
                        .borders(Borders::ALL)
                        .title("User_Input")
                        .style(
                            Style::default()
                                .fg(Color::Yellow)
                                .add_modifier(Modifier::BOLD),
                        ),
                )
                .wrap(Wrap { trim: true });
            f.render_widget(Clear, f.size());
            f.render_widget(input_widget, chunks[1]);
            // Rest of the code
        })
        Ok(input)
    }
}
```

Listing 4.14: Benutzer Eingabe

Die Funktion `draw_input_prompt` wird verwendet, um einen Benutzereingabebildschirm zu zeichnen, der einen Texteingabebereich und einen Eingabeprompt enthält. Sie nimmt eine Instanz von `Terminal` und eine Zeichenfolge von `Spans` entgegen, die den Eingabeprompt darstellt. Sie gibt eine Zeichenfolge zurück, die die Benutzereingabe enthält. Die Funktion verwendet eine Schleife, um die Benutzereingabe zu verarbeiten, bis der Benutzer die Eingabetaste drückt. Die

Funktion zeichnet zunächst einen leeren Bildschirm, um den vorherigen Inhalt zu löschen, und teilt dann den Bildschirm in zwei Teile: einen für den Eingabeprompt und einen für den Eingabebereich. Der Eingabebereich wird mit einem Sternchenmaskierungseffekt gerendert, wenn der Benutzer eine Eingabe tätigt, um die Eingabe zu verbergen. Wenn der Benutzer die Eingabetaste drückt, wird die Eingabe zurückgegeben.

```
pub fn show_input_popup<B: Backend>(
    terminal: &mut Terminal<B>,
    message: &str,
) -> Result<(), anyhow::Error> {
    let popup = Block::default()
        .borders(Borders::ALL)
        .border_style(Style::default().fg(Color::Yellow));
    let text = Text::from(Spans::from(Span::styled(
        message,
        Style::default().fg(Color::White),
    )));
    terminal.draw(|f| {
        let rect = centered_rect(30, 20, f.size());
        let inner_rect = Rect::new(
            rect.x + 1, rect.y + 1, rect.width - 2, rect.height - 2);
        f.render_widget(Clear, rect);
        f.render_widget(popup, rect);
        let paragraph = Paragraph::new(text).wrap(Wrap { trim: true });
        f.render_widget(paragraph, inner_rect);
    })?;
    // Wait for any key press to close the popup.
    loop {
        if let Event::Key(_) = event::read()? {
            break;
        }
    }
    Ok(())
}
```

Listing 4.15: Erfolgs-/Fehlermeldungen popup

Die Funktion `show_input_popup` präsentiert dem Benutzer eine Nachricht in Form eines Pop-up-Fensters auf der Terminal-Benutzeroberfläche. Als Argumente erhält die Funktion ein Terminal-objekt und einen Nachrichtenstring. Der Message-String ist der Text, der in dem Pop-up-Fenster angezeigt werden soll. Die Funktion stellt eine sehr nützliche Möglichkeit dar, Fehlermeldungen oder Benachrichtigungen für Benutzer innerhalb der Terminal-Benutzeroberfläche anzuzeigen. Wenn ein Fehler auftritt, wird die Funktion aufgerufen, um eine freundliche, lesbare Fehlermeldung in einem Popup anzuzeigen, anstatt die Ausführung des Programms zu unterbrechen oder eine unbearbeitete Fehlermeldung auf der Konsole auszugeben.

4.3 Verbindung der Komponenten

4.3.1 Wrapper Funktionen

Durch die Verwendung der Funktionen `draw_input_prompt` führt der Code schnittstellenfreundliche Kapselungen für die `CertificateManager`-Methoden ein. Die Wrapper-Funktionen sind so

konzipiert, dass sie die Interaktionen der Benutzeroberfläche integrieren, das allgemeine Benutzererlebnis verbessern und fehlerresistente Vorgänge innerhalb der Anwendung fördern.

```
pub fn split_users_tui<B: Backend>(
    cert_manager: &CertificateManager,
    terminal: &mut Terminal<B>,
    cert_path: &String,
    file_name: &mut String,
    users: &mut StatefulList<String>,
    selected_items: &mut Vec<bool>,
) -> Result<(), anyhow::Error> {
    if users.items.is_empty() {
        return Err(anyhow::anyhow!("No_users_found_in_the_certificate!"));
    }
    selected_items.resize(users.items.len(), false);
    let should_continue = show_user_selection_popup(
        terminal, users, selected_items)?;
    if let Some(true) = should_continue {
        *file_name = draw_input_prompt(
            terminal,
            &[Spans::from(
                "Enter_exported_certificate_name
                (e.g._name.certificate.pgp):",
            )],
            true,
        )?;
        let selected_users: Vec<String> = users
            .items
            .iter()
            .enumerate()
            .filter_map(|(index, user)| {
                if selected_items[index] {
                    Some(user.clone())
                } else {
                    None
                }
            })
            .collect();
        cert_manager.split_key(
            cert_path, &file_name, selected_users.clone())?;
    }
    Ok(())
}
```

Listing 4.16: split_users_tui Wrapper Funktion

Die Funktion `split_users_cli` dient als Schnittstelle zwischen dem Benutzer und der Methode `split_key` des `CertificateManager`. Mit dieser Funktion können Benutzer ein Zertifikat aufteilen, wobei bestimmte Benutzer ausgewählt werden können, die in den Aufteilungsvorgang einbezogen werden sollen. Zunächst prüft die Funktion, ob im Zertifikat Benutzer vorhanden sind. Ist dies nicht der Fall, erzeugt sie einen Fehler und gibt diesen zurück. Anschließend passt die Funktion die Liste `selected_items` an die Anzahl der Benutzer an.

Mit der Funktion `show_user_selection_popup` wird ein Pop-up-Fenster angezeigt, in dem der

Benutzer die Benutzer auswählen kann, die in den Split-Vorgang einbezogen werden sollen. Mit dem Flag `should_continue` wird geprüft, ob der Benutzer mit dem Vorgang fortfahren möchte. Entscheidet sich der Benutzer, den Vorgang fortzusetzen, wird er über die Funktion `draw_input_prompt` aufgefordert, den Namen für die neue Zertifikatsdatei einzugeben. Anschließend sammelt `selected_users` die vom Benutzer ausgewählten Benutzer. Schließlich wird die Methode `split_key` des `CertificateManager` aufgerufen, wobei der ursprüngliche Zertifikatspfad, der Name des neuen Zertifikats und die ausgewählten Benutzer übergeben werden. Die Funktion kapselt somit die Benutzerinteraktionen und die Fehlerbehandlung rund um die `split_key` Methode.

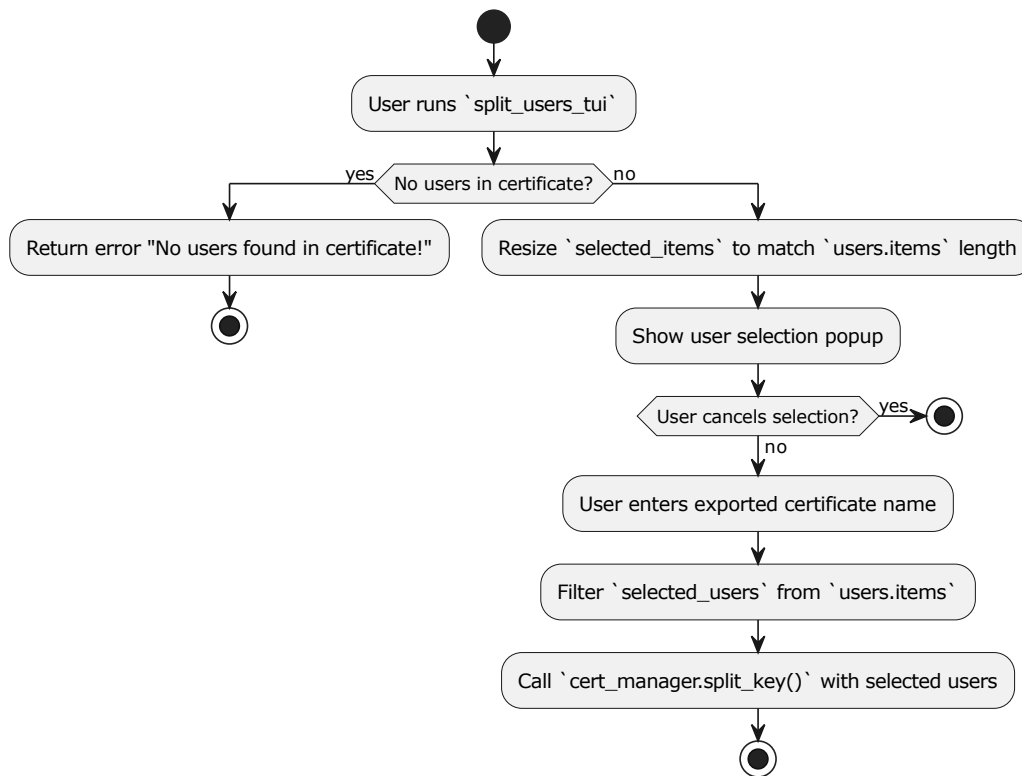


Abbildung 4.1: Flußdiagramm für die Wrapper-Funktion `split_users_tui`

4.3.2 Rendern der Schnittstelle

```

pub fn run_app<B: Backend>(
    terminal: &mut Terminal<B>,
    mut app: App,
) -> Result<(), Box<dyn std::error::Error>> {
    let manager = CertificateManager;
    let (tx, rx) = mpsc::channel();

    // Spawn a separate thread to receive events from Crossterm
    thread::spawn(move || {
        let mut last_event_time = Instant::now();
        loop {
            if event::poll(Duration::from_millis(100)).unwrap() {
                let event = event::read().unwrap();
                tx.send(event).unwrap();
            }
        }
    });
  
```

```

        last_event_time = Instant::now();
    } else {
        if Instant::now().duration_since(
            last_event_time) > Duration::from_secs(1) {
            tx.send(crossterm::event::Event::Resize(0, 0)).unwrap();
            last_event_time = Instant::now();
        }
    }
}
});
}

```

Listing 4.17: Implementierung der Hauptfunktion des Anwendungsläufers

Der bereitgestellte Code führt die Funktion `run_app` ein, mit der die Anwendung in Betrieb genommen wird. Diese Funktion verwendet generische Programmierung, um jedes Backend `B` zu akzeptieren, das die Eigenschaft `Backend` implementiert. Die Funktion erstellt zunächst eine Instanz von `CertificateManager`. Dann richtet sie einen Multiple-Producer-Single-Consumer-Kanal (*mpsc*) ein, um die asynchrone Kommunikation zwischen Threads zu erleichtern. Der Kanal wird zum Senden und Empfangen von `Event`-Instanzen verwendet, die Benutzereingaben darstellen. Anschließend wird ein neuer Thread gestartet. Innerhalb dieses Threads prüft eine Endlosschleife alle 100 Millisekunden auf neue Ereignisse. Wird ein Ereignis festgestellt, wird es über den Kanal an den Hauptthread gesendet. Wenn innerhalb einer Sekunde keine Ereignisse aufgetreten sind, erzeugt der Code programmatisch ein Größenänderungsereignis und sendet dieses Ereignis über den Kanal. Durch diesen Mechanismus wird sichergestellt, dass das Terminal seinen Zustand regelmäßig aktualisiert, auch wenn keine Benutzereingaben erfolgen. Die Verwendung dieses separaten Threads zur Verwaltung von Ereignissen ermöglicht eine effiziente und nicht blockierende Verarbeitung von Benutzereingaben und Änderungen des Terminalzustands, was zu einer reaktionsschnellen Benutzeroberfläche führt.

4.3.3 Tasten-Ereignisse

```

loop {
    terminal.draw(|f| ui(f, &mut app))?;
    match rx.recv()? {
        Event::Key(key) => {
            // key events
        }
    }
}

```

Listing 4.18: Ereignisschleife in der Terminal-Benutzeroberfläche

Der bereitgestellte Code stellt die Hauptereignisschleife der Anwendung dar, die kontinuierlich die Anzeige des Terminals aktualisiert und Benutzereingaben verarbeitet, bis die Anwendung beendet wird. Die Anwendung wird bei jeder Iteration der Schleife mit `terminal.draw(|f| ui(f, &mut app))?` neu gezeichnet. Als Nächstes wartet sie auf ein Ereignis vom rx-receiver, der auf den separaten Thread wartet, der die Eingabeereignisse verarbeitet. Wenn ein Tastenereignis empfangen wird, wird es auf der Grundlage des aktiven Zustands der Anwendung und der spezifischen Taste, die gedrückt wurde, behandelt.

```

KeyCode::Char('p') => {
    if let Some(cert_path) = app.items.selected() {
        let cert_path = Path::new(&cert_path);
    }
}

```

```

    match check_certificate(cert_path) {
      CertificateType::Cert(_) => {
        match edit_password_tui(
          &manager,
          terminal,
          &cert_path.to_string_lossy().to_string(),
          &mut app.original_pw,
          &mut app.pw,
          &mut app.rpw,
        ) {
          Ok(()) => {
            show_input_popup(
              terminal,
              "Password_changed_successfully.",
            );
          }
          Err(e) => {
            show_input_popup(
              terminal,
              &format!("Error:_{e}", e),
            );
          }
        }
      }
      _ => {
        show_input_popup(terminal, "Not_a_valid_certificate!");
      }
    }
  }
}

```

Listing 4.19: Das Ereignis Tastendruck: Passwortänderung

Der obige Codeausschnitt beschreibt das Verhalten der Anwendung, wenn eine Taste gedrückt wird. In diesem Zusammenhang soll (*p*) den Prozess der Änderung des Passworts eines Zertifikats einleiten. Wenn die Taste (*p*) gedrückt wird, prüft die Anwendung zunächst, ob gerade ein Element ausgewählt ist. Ist dies der Fall, überprüft der Code die Gültigkeit des Elements mit der Hilfsfunktion `check_validity`, die den Typ des ausgewählten Elements (Zertifikat, Signatur oder ungültig) prüft. Handelt es sich bei dem ausgewählten Element um ein Zertifikat und ist es gültig, fährt die Anwendung mit der Wrapper-Funktion `edit_password_tui()` fort, die das Passwort für das Zertifikat ändert. Diese Funktion nimmt den Pfad des Zertifikats, das ursprüngliche Passwort und das neue Passwort auf. Wenn die Passwortänderung erfolgreich war, wird eine Popup-Meldung mit dem Text "Passwort erfolgreich geändert" angezeigt. Wenn während des Vorgangs ein Fehler auftritt, wird stattdessen eine Fehlermeldung angezeigt. Handelt es sich bei der ausgewählten Datei nicht um ein gültiges Zertifikat, z. B. einen Ordner oder eine Signatur, wird eine Popup-Meldung mit dem Hinweis "Kein gültiges Zertifikat!".

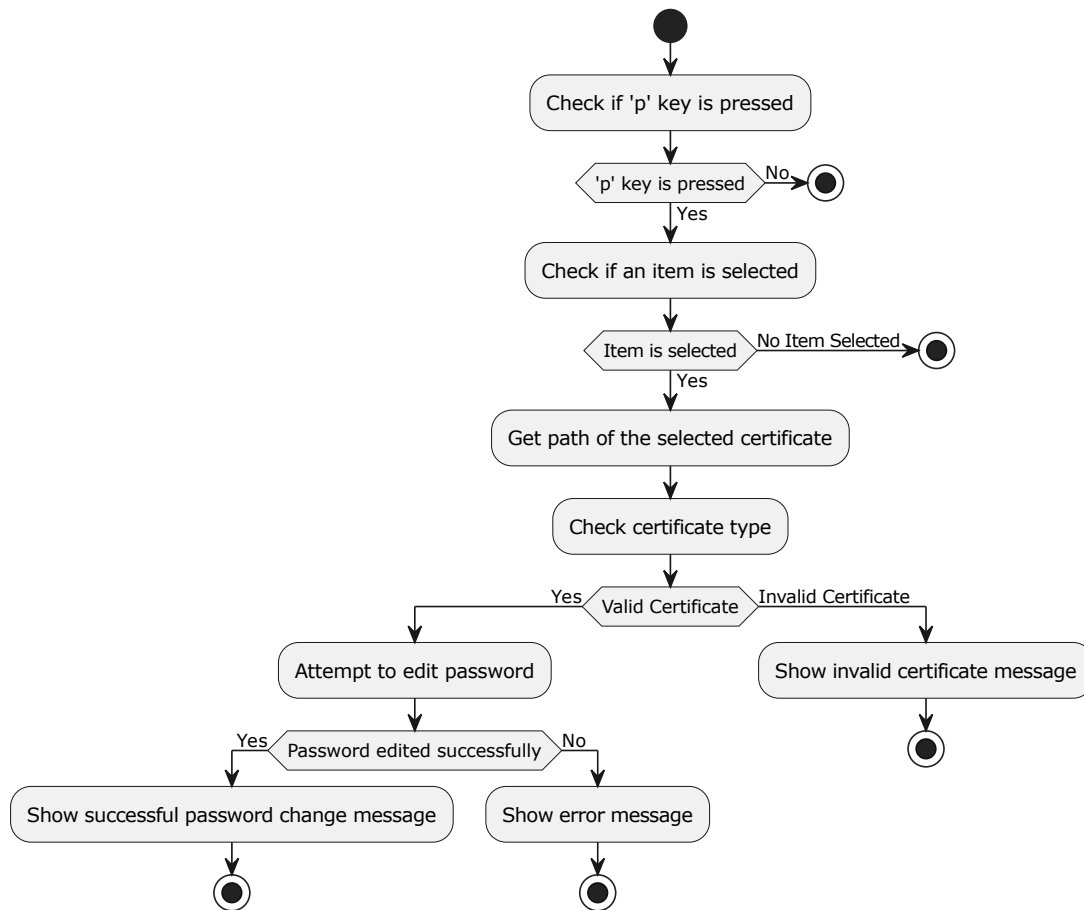


Abbildung 4.2: Flußdiagramm für das Ereignis Tastendruck: Passwortänderung

Zusammenfassend ist in diesem Kapitel über die Implementierung unbedingt darauf hinzuweisen, dass aufgrund des Umfangs der Codebasis nicht alle Funktionen in den vorangegangenen Abschnitten ausführlich beschrieben wurden. Einige Hilfsfunktionen, einschließlich derjenigen, die für die Verzeichnisinitialisierung oder das Parsen der ISO 8601-Dauer zuständig sind, wurden nicht im Detail erläutert. Nichtsdestotrotz wurde in diesem Kapitel versucht, die wichtigsten Aspekte der Implementierung zu beleuchten, wobei der Schwerpunkt auf dem CertificateManager und der Terminal User Interface (TUI) lag. Diese zentralen Elemente bilden die Grundlage der Systemarchitektur, ermöglichen die Interaktion mit dem Benutzer und steuern die Kernfunktionen. Ihr Verständnis ist unerlässlich, um den Gesamtaufbau der Anwendung und das Zusammenspiel zwischen den verschiedenen Komponenten zu verstehen. Für eine umfassende Untersuchung wird den Lesern empfohlen, den vollständigen Quellcode zu diesem Bericht zu lesen. Er bietet einen detaillierten Einblick in alle Funktionalitäten, auch in die, die in diesem Kapitel nicht ausführlich behandelt werden.

Kapitel 5

Ergebnisse

Die Terminal-Benutzerschnittstelle (TUI) und die Kernfunktionalitäten der implementierten Anwendung haben sich als äußerst effizient erwiesen. In diesem Kapitel werden die Ergebnisse dieser Implementierungen vorgestellt.

Die Implementierung von PGP-Funktionen wie die Erzeugung von Schlüsselpaaren, die Änderung von Passwörtern und die Aufteilung von Zertifikaten hat sich als effektiv erwiesen. Jede dieser Funktionen funktioniert wie vorgesehen und trägt erheblich zum Gesamtnutzen der Anwendung bei. Es ist jedoch wichtig zu betonen, dass die Validierung dieser Funktionen in erster Linie nur durch den Autor des Projekts durchgeführt wurde. Infolgedessen kann es zu unerkannten Abweichungen oder Fehlern im Programm kommen. Ein breit angelegter Test durch eine Reihe von Benutzern könnte weitere potenzielle Probleme zutage fördern. Der Quellcode der Anwendung ist zwar umfangreich, aber gut strukturiert und organisiert, was eine effiziente Implementierung der Funktionen und die Beibehaltung der Codebasis des Programms erleichtert.

5.1 Das Hauptfenster



Abbildung 5.1: Hauptfenster der Schnittstelle

Die Abbildung 5.1 zeigt das Hauptfenster der Anwendung. Es ist in drei Abschnitte unterteilt:

1. Die Hilfeleiste befindet sich am oberen Rand des Hauptfensters und dient als wichtige Ressource für die Benutzer, die durch das Programm navigieren. Sie zeigt nützliche Anweisungen und Tastenkombinationen an und stellt sicher, dass die Benutzer leicht auf die verschiedenen Funktionen der Anwendung zugreifen und sie bedienen können. Das Vorhandensein der Hilfeleiste sorgt für eine nahtlose Benutzererfahrung, da sie als ständiger Bezugspunkt für die Benutzer dient, was mögliche Verwirrung reduziert und den Navigationsprozess überschaubar macht.
2. Der Navigationsbaum befindet sich in der Mitte des Hauptfensters und bietet eine organisierte, hierarchische Ansicht der Verzeichnisse und Dateien, die es dem Benutzer ermöglicht, den Inhalt des Systems auf strukturierte und intuitive Weise zu erkunden. Durch Interaktion mit diesem Navigationsbaum können die Benutzer Verzeichnisse oder Dateien auswählen, sie öffnen oder andere relevante Operationen durchführen.
3. Die Navigationsleiste befindet sich im unteren Teil des Hauptfensters. Sie zeigt den Pfad des aktuell ausgewählten Ordners oder der Datei an und gibt dem Benutzer eine klare Vorstellung von seiner Position im System. Sie ist ein entscheidendes Element für die Aufrechterhaltung des Kontexts während der Navigation und stellt sicher, dass die Benutzer immer einen klaren Bezug zu ihrer aktuellen Position innerhalb der Verzeichnisstruktur des Systems haben.

Die Datei mit der Bezeichnung `ui.rs`, die sich im Verzeichnis `/src/app` befindet, enthält den Code, der für die Entwicklung und Funktionsweise des Hauptanwendungsfensters verantwortlich ist.

5.2 Die Benutzereingabe

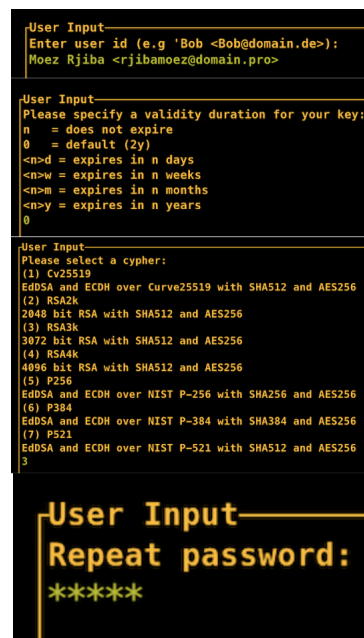
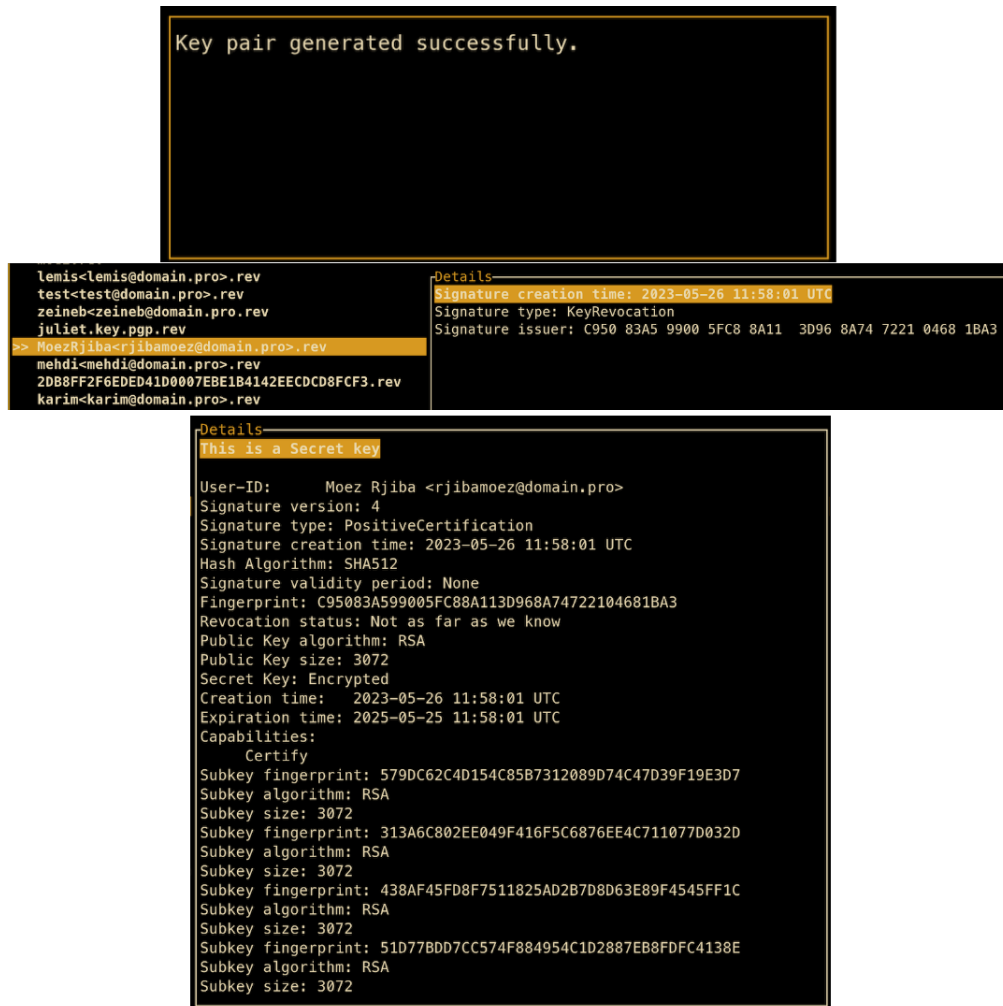


Abbildung 5.2: Parameter für die Erzeugung von Schlüsselpaaren



```

Key pair generated successfully.

lemis<lemis@domain.pro>.rev
test<test@domain.pro>.rev
zeineb<zeineb@domain.pro>.rev
juliet.key.pgp.rev
>> MoezRjiba<rjibamoez@domain.pro>.rev
mehdi<mehdi@domain.pro>.rev
2DB8FF2F6ED41D0007EBE1B4142EECD8FCF3.rev
karim<karim@domain.pro>.rev

Details:
Signature creation time: 2023-05-26 11:58:01 UTC
Signature type: KeyRevocation
Signature issuer: C950 83A5 9900 5FC8 8A11 3D96 8A74 7221 0468 1BA3

Details:
This is a Secret key
User-ID: Moez Rjiba <rjibamoez@domain.pro>
Signature version: 4
Signature type: PositiveCertification
Signature creation time: 2023-05-26 11:58:01 UTC
Hash Algorithm: SHA512
Signature validity period: None
Fingerprint: C95083A599005FC88A113D968A74722104681BA3
Revocation status: Not as far as we know
Public Key algorithm: RSA
Public Key size: 3072
Secret Key: Encrypted
Creation time: 2023-05-26 11:58:01 UTC
Expiration time: 2025-05-25 11:58:01 UTC
Capabilities:
  Certify
Subkey fingerprint: 579DC62C4D154C85B7312089D74C47D39F19E3D7
Subkey algorithm: RSA
Subkey size: 3072
Subkey fingerprint: 313A6C802EE049F416F5C6876EE4C711077D032D
Subkey algorithm: RSA
Subkey size: 3072
Subkey fingerprint: 438AF45FD8F7511825AD2B7D8D63E89F4545FF1C
Subkey algorithm: RSA
Subkey size: 3072
Subkey fingerprint: 51D77BDD7CC574F884954C1D2887EB8FDFC4138E
Subkey algorithm: RSA
Subkey size: 3072

```

Abbildung 5.3: Ergebnis der Erzeugung von Schlüsselpaaren

Abbildung 5.2 zeigt, wie die Funktion `draw_input_prompt` (4.14) mit der Methode `generate_keypair` (4.3) zusammenarbeitet. Diese Abbildung zeigt auf elegante Weise, wie die Benutzereingabe mit den verschiedenen Feldern der Methode interagiert. Sie verdeutlicht, wie die Schnittstelle den Benutzer einbindet und die Eingabe auf eine klare, strukturierte Weise erleichtert.

Abbildung 5.3 unterstreicht das Ergebnis der oben genannten Operation und hebt die zentrale Rolle der Funktion `show_input_popup` (4.15) hervor. Diese Funktion ist wesentlich für die Darstellung des Ergebnisses der Operation in Form einer Kurzmeldung, die besagt: „Schlüsselpaar erfolgreich erzeugt“.

Die Abbildung unterstreicht außerdem die entscheidende Rolle der Methoden, `CertificateDetails` (4.1) und `SignatureDetails` (4.2). Diese Methoden übermitteln dem Benutzer wichtige Informationen über das erzeugte Schlüsselpaar und das Sperrzertifikat und unterstreichen damit die Transparenz und den benutzerorientierten Ansatz des Programms. Insgesamt zeigen diese Abbildungen auf elegante Weise den Zusammenhalt und das benutzerzentrierte Design.

5.3 Benutzer aufteilen

Abbildung 5.4 bietet eine eindrucksvolle Veranschaulichung der `split_users` Methode (4.7) in der Praxis und zeigt das resultierende Zertifikat in einer visuell ansprechenden Weise. Die Abbildung lenkt die Aufmerksamkeit vor allem auf den interaktiven Aspekt der Anwendung. Es

erscheint ein Popup, das eine Liste der vorhandenen Benutzer anzeigt und so den Benutzer auffordert, eine Auswahl zu treffen. Diese Interaktivität wird durch die Einbindung der `StatefulList` verstärkt, die die Auswahl von Elementen aus einer Liste erleichtert. Diese Abbildung unterstreicht das benutzerzentrierte Design der Anwendung, die robuste Funktionalität ihrer Methoden und die Eleganz, mit der sie die Benutzerinteraktion mit der Benutzeroberfläche verbindet. Sie bringt die dynamische Natur der Anwendung und ihr Engagement für eine intuitive und reaktionsschnelle Benutzererfahrung sehr schön zum Ausdruck.

```

-Details-
This is a Public Key
User-ID:      John Sina <sinajohn@domain.com>
Signature version: 4
Signature type: PositiveCertification
Signature creation time: 2023-05-02 11:53:00 UTC
Hash Algorithm: SHA512
Signature validity period: None
User-ID:      Moez Rjiba <rjibamoez@gmail.com>
Signature version: 4
Signature type: PositiveCertification
Signature creation time: 2023-05-02 11:52:13 UTC
Hash Algorithm: SHA512
Signature validity period: None
User-ID:      salma <salma@domain.pro>
Signature version: 4
Signature type: PositiveCertification
Signature creation time: 2023-04-21 10:18:14 UTC
Hash Algorithm: SHA512
Signature validity period: None
Fingerprint: E6EEEA2AF00E184F16EA3A7B6279556487D3FF85
Revocation status: Not as far as we know
Public Key algorithm: EdDSA
Public Key size: 256
Creation time: 2023-04-21 10:18:14 UTC
Expiration time: 2025-04-20 10:18:14 UTC
Capabilities:

User Selection
[x] sinajohn@domain.com
[x] rjibamoez@gmail.com
[x] salma@domain.pro

salma.pgp
foo.bar.gpg
salma.certificate.pgp
elyes.certificate.pgp
new_salma.certificate.pgp
moez.pgp
mo.pgp
tet.pgp
new.certificate.pgp
>> mynew.certificate.pgp
john.certificate.pgp
salma.pgp

-Details-
This is a Public key
User-ID:      John Sina <sinajohn@domain.com>
Signature version: 4
Signature type: PositiveCertification
Signature creation time: 2023-05-02 11:53:00 UTC
Hash Algorithm: SHA512
Signature validity period: None
User-ID:      Moez Rjiba <rjibamoez@gmail.com>
Signature version: 4
Signature type: PositiveCertification
Signature creation time: 2023-05-02 11:52:13 UTC
Hash Algorithm: SHA512
Signature validity period: None
Fingerprint: E6EEEA2AF00E184F16EA3A7B6279556487D3FF85
Revocation status: Not as far as we know
Public Key algorithm: EdDSA
Public Key size: 256
Creation time: 2023-04-21 10:18:14 UTC
Expiration time: 2025-04-20 10:18:14 UTC
Capabilities:
Certify

```

Abbildung 5.4: Ergebnis der Aufteilung des öffentlichen Schlüssels

5.4 Schlüssel widerrufen

Abbildung 5.5 zeigt anschaulich die Funktionsweise der Methode `revoke_key` (4.9) und das daraus resultierende Zertifikat. Sie verdeutlicht die Feinheiten der Interaktion innerhalb der Anwendung, bei der der Benutzer ein Eingabefenster zur Auswahl des Grundes für den Widerruf vorfindet. Die Abbildung unterstreicht wirkungsvoll die Kombination aus der Entscheidungsfindung des Nutzers und der effizienten Reaktion des Programms.

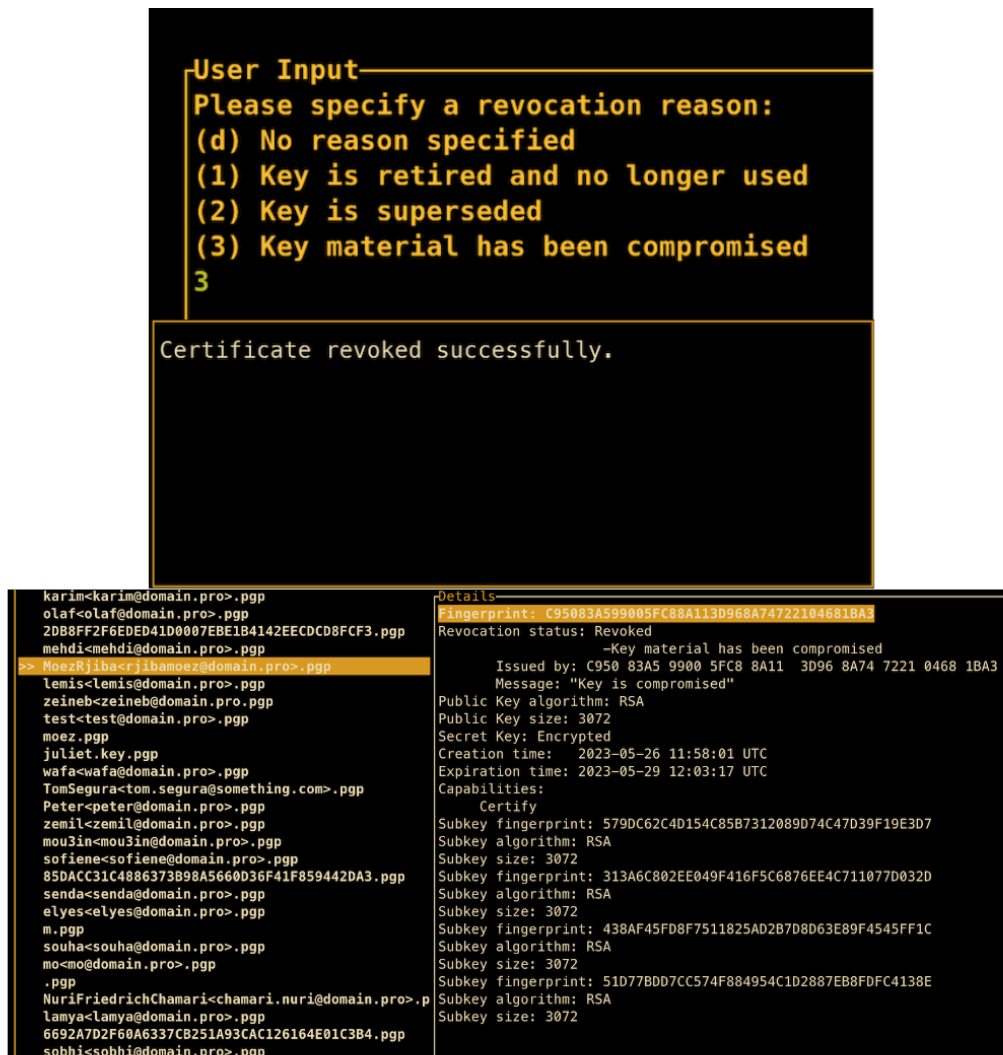


Abbildung 5.5: Ergebnis des Zertifikatswiderrufs

5.5 Hilfe-Fenster

Abbildung 5.6 zeigt das Hilfe-Fenster, eine wichtige Eigenschaft, um die Benutzerfreundlichkeit zu verbessern. Dieses Fenster gibt einen kurzen Überblick über die verfügbaren Tastenkombinationen, begleitet von einer ausführlichen Erläuterung der Nutzbarkeit der einzelnen Funktionen. Es zeigt das Engagement der Anwendung für ein benutzerfreundliches Design, indem es intuitive und leicht verständliche Anweisungen bietet. Sie soll den Benutzer nicht nur informieren, sondern ihn auch in die Lage versetzen, die Anwendung effektiv zu nutzen, was den Schnittpunkt von Funktionalität und Benutzererfahrung im Design der Anwendung weiter unterstreicht.

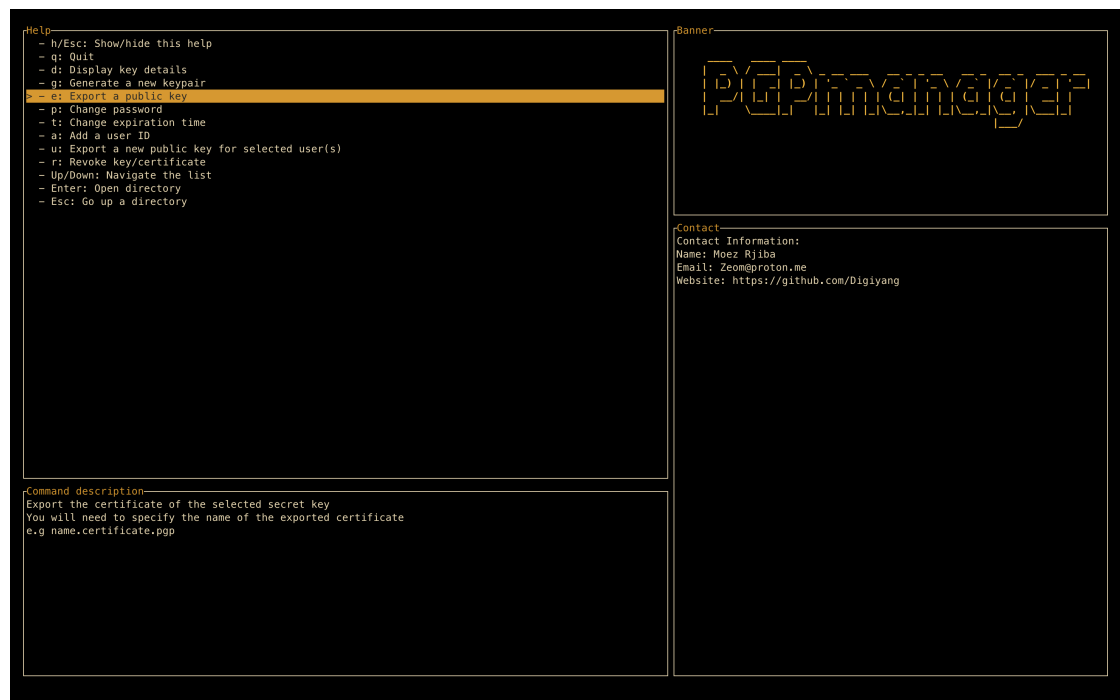


Abbildung 5.6: Hilfe-fenster der Schnittstelle

Kapitel 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung und Fazit

Das durchgeführte Projekt hat den komplexen, aber faszinierenden Prozess der Gestaltung und Implementierung einer Terminal-Benutzerschnittstelle mit Schwerpunkt auf Funktionalität und Benutzerfreundlichkeit aufgezeigt. Auf dieser Reise wurde das Zusammenspiel zwischen kritischen Aspekten wie Zertifikatsmanagement, Benutzerinteraktion und Gestaltung der Benutzeroberfläche gründlich erforscht.

Trotz der Einschränkungen durch begrenzte Tests und potenzielle unentdeckte Fehler aufgrund von Zeitbeschränkungen hat sich das Projekt als eine interessante Übung in der Softwareentwicklung erwiesen.

Auch wenn die derzeitige Implementierung vielversprechende Ergebnisse gezeigt hat, gibt es noch Raum für weitere Entwicklungen. Die Anwendung könnte von umfangreicheren Tests, Optimierungen und möglichen Funktionserweiterungen profitieren. Zukünftige Arbeiten könnten auch die Gewährleistung höherer Sicherheitsstandards und das Hinzufügen benutzerfreundlicher Funktionen beinhalten.

Insgesamt stellt das Projekt eine solide Grundlage für ein potenziell umfassendes Zertifikatsmanagementsystem dar. Sein Design und seine Implementierung sind ein Beispiel für das Verhältnis zwischen Funktion und Form bei Softwareanwendungen. Da sich die Softwareentwicklung ständig weiterentwickelt, sind Projekte wie dieses von entscheidender Bedeutung für die Erforschung neuer Paradigmen und die Entwicklung besserer, benutzerorientierter Anwendungen. Die aus diesem Projekt gewonnenen Erkenntnisse ebnen den Weg für weiteres Wachstum und Innovation.

6.2 Zukünftige Entwicklungen

Mit Blick auf die Zukunft gibt es zahlreiche Möglichkeiten für die weitere Entwicklung und Verbesserung dieses Projekts. Zum Beispiel könnte die Integration zusätzlicher, noch nicht implementierter Pretty Good Privacy (PGP)-Funktionen die Funktionalität und Sicherheit der Software erhöhen. Dazu gehören fortgeschrittene Verschlüsselungs- und Signiermethoden, stärkere Schlüsselerhandlungsprotokolle oder noch umfassendere Schlüsselverwaltungssysteme. Ebenso könnte die Einführung des Konzepts der Keyrings für die organisierte Speicherung und den einfachen Abruf öffentlicher und privater Schlüssel die Benutzerfreundlichkeit und die Effizienz der Schlüsselverwaltungsprozesse verbessern. Es gibt viele Richtungen, in die sich dieses Projekt entwickeln kann, und alle versprechen spannende Möglichkeiten für die Erweiterung seiner Fähigkeiten und die Verfeinerung seiner Verbindung mit dem Benutzer. Da sich der Bereich der Verschlüsselung und Sicherheit weiter entwickelt, wird auch das Potenzial für zusätzliche Funktionen und Verbesserungen in dieser Anwendung zunehmen.

Kapitel 7

Anhang

7.1 Installationsanleitung

Das Projekt ist auch auf dem Gitlab der Hochschule unter dieser Adresse verfügbar:

<https://gitlab.beuth-hochschule.de/s64596/pgp-manager>.

Um darauf zugreifen zu können, müssen Sie Mitglied der Hochschule sein.

7.1.1 Voraussetzungen

Bevor Sie die Terminal-Benutzerschnittstelle (TUI) ausführen können, müssen Sie zunächst sicherstellen, dass Ihr System die erforderlichen Voraussetzungen erfüllt.

1. Rust Programmiersprache:

Die TUI ist in Rust geschrieben, daher ist Rustup, das Installations- und Versionsverwaltungsprogramm für Rust, erforderlich. Wenn Rust nicht auf Ihrem System installiert ist, besuchen Sie bitte [die offizielle Rust-Website](#) für Installationsanweisungen.

2. Sequoia OpenPGP Crate:

Die TUI stützt sich auf die Sequoia OpenPGP-Crates für die Zertifikatsverwaltung. Sie müssen bestimmte Abhängigkeiten in Ihrem System installiert haben. Besuchen Sie [das Sequoia-Repository](#) für detaillierte Anweisungen zu den erforderlichen Bibliotheken für verschiedene Plattformen.

7.1.2 Installation

Nachdem Sie die oben genannten Voraussetzungen erfüllt haben, können Sie die TUI mit den folgenden Schritten ausführen:

1. Navigieren Sie in das Verzeichnis „gui“:

```
$ cd gui
```

2. Bauen Sie das Projekt mit Cargo auf:

```
$ cargo build --release
```

Die ausführbare Datei befindet sich im Verzeichnis `/target/release` innerhalb des Projektverzeichnisses.

7.1.3 Ausführung

Um das Programm auszuführen, gehen Sie folgendermaßen vor:

1. Navigieren Sie zum release verzeichnis:

```
$ cd target/release
```

2. Führen Sie die ausführbare TUI-Datei aus::

```
$ ./gui
```

7.1.4 Benutzerhandbuch

Nach dem Ausführen des Programms interagieren Sie mit der Anwendung über das Terminal. Die TUI ist intuitiv und benutzerfreundlich gestaltet. Tastenbelegungen und die entsprechenden Funktionen werden im Hilfefenster der Anwendung angezeigt. Verwenden Sie die Pfeiltasten, um durch die Liste der Befehle zu navigieren. Drücken Sie die entsprechende Taste, um eine Operation auszulösen.

Sollten Sie bei der Verwendung der Anwendung auf Schwierigkeiten oder Probleme stoßen, schauen Sie bitte im Quellcode des Projekts und in der mit dem Bericht gelieferten Dokumentation nach, um ein umfassendes Verständnis der Funktionen der Anwendung zu erhalten.

Literatur

- [1] Johannes A. Buchmann, Evangelos Karatsiolis und Alexander Wiesmaier. *Introduction to Public Key Infrastructures*. ISBN 978-3-642-40656-0. Springer-Verlag, 2013. Kap. 1, S. 13.
 - [2] Johannes A. Buchmann, Evangelos Karatsiolis und Alexander Wiesmaier. *Introduction to Public Key Infrastructures*. ISBN 978-3-642-40656-0. Springer-Verlag, 2013. Kap. 2, S. 21.
 - [3] IBM Technology corporation. *Public key cryptography*. Last accessed 04 June 2023. 2023. URL: <https://www.ibm.com/docs/en/ztpf/2023?topic=concepts-public-key-cryptography>.
 - [4] IBM Technology corporation. *x.509 certificate revocation*. Last accessed 04 June 2023. 2023. URL: <https://www.ibm.com/docs/en/zos/2.5.0?topic=management-x509-certificate-revocation>.
 - [5] Florian Dehau. *tui-rs crate*. Last accessed 04 June 2023. 2022. URL: <https://crates.io/crates/tui>.
 - [6] Christian Forler. *IT-Sicherheit Vorlesung*. Vorlesungsskript, Kapitel 5: Asymmetrische Kryptographie, Seite 250. 2020.
 - [7] pep foundation. *sequoia-openpgp crate*. Last accessed 04 June 2023. URL: <https://sequoia-pgp.org/>.
 - [8] Simson Garfinkel. *PGP: Pretty Good Privacy*. ISBN 1-56592-098-8. O'Reilly & Associates, Inc., 1995. Kap. 12, S. 235–237.
 - [9] Steve Klabnik und Carol Nichols. *The Rust Programming Language*. ISBN 978-1-59327-828-1. No Starch Press, 2018. Kap. 9, S. 155–164.
 - [10] Steve Klabnik und Carol Nichols. *The Rust Programming Language*. ISBN 978-1-59327-828-1. No Starch Press, 2018. Kap. 6, S. 98–110.
 - [11] Steve Klabnik und Carol Nichols. *The Rust Programming Language*. ISBN 978-1-59327-828-1. No Starch Press, 2018. Kap. 13, S. 263–283.
 - [12] Steve Klabnik und Carol Nichols. *The Rust Programming Language*. ISBN 978-1-59327-828-1. No Starch Press, 2018. Kap. 10, S. 174–182.
 - [13] Michael W Lucas. *PGP & GPG: Email for the Practical Paranoid*. ISBN 1-59327-071-2. No Starch Press, 2006.
 - [14] Bundesamt für Sicherheit in der Informationstechnik. *Public Key Infrastrukturen (PKIen)*. Last accessed 04 June 2023. URL: <https://www.bsi.bund.de/dok/6615610>.
 - [15] Rust team. *Rust*. Last accessed 04 June 2023. URL: <https://www.rust-lang.org/>.
 - [16] International Telecommunication Union. *Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks*. ITU-T Recommendation X.509. 2019. URL: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-201910-I!!PDF-E&type=items.
-