Masterarbeit

Schlüssel-Management und Verschlüsslungsverfahren im Kontext eines Quantenkommunikationsnetzwerks

Eingereicht von: Arne Klee
1173482

Erstprüfung: Prof. Dr. Udo Helmbrecht
Zweitprüfung: Prof. Dr. Stefan Pickl

Betreuer: Hedwig Körfgen
Nils Otto vor dem gentschen Felde

Abgabedatum: 30. Juni 2021

Universität der Bundeswehr München
Forschungsinstitut CODE

Abstract

Im Rahmen dieser Masterarbeit soll eine Schlüsselmanagementanwendung im Rahmen eines Quantenkommunikationsnetzes entworfen und anhand eines zweckmäßigen Prototypens getestet werden. Dadurch ergibt sich die Forschungsfrage wie ein entsprechender Anforderungskatalog sowie ein sich daran orientierender Programmentwurf hergeleitet und erstellt werden können. Zusätzlich muss geprüft werden in welcher Weise ein passender Prototyp implementiert werden kann und inwiefern dieser am Ende zweckmäßig ist.
Zunächst wurden sowohl grundlegend auf die (Quanten-)Kryptographie, dem Quantenschlüsselaustausch als auch die ETSI-Protokoll 004 sowie 014 eingegangen. Ebenfalls ist kurz das Projekt MuQuaNet, in dessen Rahmen diese Masterarbeit verfasst worden ist, vorgestellt. Diese Wissensgrundlage wurde genutzt im ersten Entwurfsschritt eine Anforderunsganalyse durchzuführen, wobei die Kommunikation zwischen mehreren Knoten (1:1, 1:n, n:m) sowohl mit als auch ohne Zwischenknoten betrachtet worden ist. So ist am Ende ein umfänglicher Anforderungskatalog mit verschiedenen funktionalen und nicht-funktionalen Anforderungen entstand. Darauf aufbauend ist ein Programmentwurf ausgearbeitet worden, wobei zuerst die benötigten Komponenten definiert worden sind, aus denen die Klassen- und Sequenzdiagramme abgeleitet werden konnten. Hierbei wurden grundlegende Designprinzipien aufgestellt, wie beispielsweise das Nutzen generischer Schnittstellen, welche verschiedene Anfragen unterscheiden können. Auf Grundlage dieses Softwareentwurfes ist ein zweckmäßiger Prototyp, um eine praktische Umsetzbarkeit beweisen zu können, implementiert und dessen Funktionen vorgestellt worden. Das verwendete Protokoll ETSI 014 sowie Prototyp sind im Anschluss innerhalb der Diskussion der Ergebnisse einer kritischen Sicherheitsüberprüfung unterzogen worden. Hierbei sind einige Sicherheitslücken aufgedeckt worden, welche allerdings im Kontext dieser Masterarbeit nicht von aussschlaggebender Relevanz waren. Zusätzlich wurde der Prototyp noch auf dessen Zweckmäßigkeit beurteilt, wobei schlussendlich die Zweckmäßigkeit im gestellten Szenario bestätigt worden ist.
Am Ende konnten anhand dieser Methodik ein umfassender Anforderungskatalog mit einem darauf aufbauendem Programmentwurf erarbeitet werden. Beides bot die Grundlage für die Implementierung eines grundlegenden sowie zweckmäßigen Prototypens.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung und Problemstellung

1.1 Einleitung

In unserer heutigen Zeit nimmt die Kommunikation im digitalen Raum sowie die Digitalisierung im Allgemeinen eine immer größere und gewichtigere Rolle ein. Dies steigert nicht nur das Potenzial zur Optimierung des menschlichen Alltags sowie Abläufen innerhalb von Unternehmen und vielem mehr, sondern auch gleichzeitig das Potenzial von digitalen Angriffen. Die Anzahl sowie die Effizienz nehmen immer weiter zu und verlangen nach neuen Lösungsansätzen, um diese abzuwehren oder gleich zu verhindern. Eine neue Möglichkeit in diesem Kampf bietet die Quantenmechanik, welche die Kommunikation, insbesondere die Kryptographieverfahren, grundlegend verändern könnte. Eine besondere Rolle hierbei nimmt das Schlüsselmanagement ein, da es bei dieser neuen Technik eine der wenigen Angriffspunkte darstellt. Deshalb müssen neue Verfahren entwickelt sowie alte angepasst werden, damit die Möglichkeit einer wirklich sicheren Kommunikation wahrgenommen werden kann.
Die Quantenmechanik, begründet ab den 1920ern, dient als Grundlage für diese neue Technologie und bricht mit einigen Gesetzen der klassischen Physik. Denn für Teilchen auf atomarer Größenordnung ist eine quantenmechanische Beschreibung notwendig, welche so nicht in der herkömmlichen Physik möglich ist und somit die klassische Mechanik an dieser Stelle ihren Gültigkeitsbereich verlässt. Hierbei kommt es allerdings bei jeder Interaktion zu Auswirkungen auf das gemessene Objekt selbst, wodurch Messergebnisse verändert werden. Der alte Zustand ist somit nicht mehr reproduzierbar und wird zerstört. Hierbei ist der entscheidende Unterschied zur klassischen Mechanik die probabilistische Natur des Systems bis zur konkreten Messung. Denn bis zu dieser nimmt ein Teilchen keinen konkreten Zustand an, sondern ist viel mehr als eine Wahrscheinlichkeitsverteilung zu beschreiben. Im Laufe dieser Arbeit wird hierauf noch näher eingegangen.

Um diese physikalischen Erkenntnisse in der Praxis der Informatik einsetzen zu können, werden Protokolle, wie das BB84, benötigt. Diese dienen dazu, dass sich zwei Parteien, beispielsweise Alice und Bob, auf einen gemeinsamen Schlüssel einigen können. Der Unterschied zu den herkömmlichen Protokollen besteht darin, dass Alice eine beliebige Abfolge von polarisierten Photonen schickt und Bob diese jedes Mal zufällig misst. Beide notieren ihre Ergebnisse, vergleichen diese im Rahmen eines Protokolls wie BB84 und einigen sich so auf einen gemeinsamen Schlüssel.
Die Sicherheit hinter solchen Verfahren basiert auf den physikalischen Naturgesetzen, wie schon weiter oben bezüglich des Zustandswechsels bei einer Messung erwähnt, und nicht wie bisher auf mathematischen Berechnungen. Hierbei entsteht ein Konflikt zwischen der physikalischen und informationstechnischen Sichtweise in Bezug auf die Sicherheit solcher Verfahren. Angriffsvektoren wie Man-in-the-Middle (MitM) werden von der Quantenmechanik nicht beachtet, obwohl sie eine Gefahr für solche Ansätze darstellen und deshalb in der klassischen Kryptographie bedacht werden (müssen). Diese Problematik stellt sich auch innerhalb dieser Masterarbeit. Deshalb wird einfachheitshalber angenommen, dass der Datenstrom nur über sichere Leitungen stattfindet und somit MitM-Attacken ignoriert. Denn dies würde den Umfang dieser Arbeit übertreffen und könnte als eigenständige Thematik für eine weitere Masterarbeit dienen.
Ein Schlüssel-Management-System muss viele verschiedene Anwendungsfälle abdecken, beispielsweise eine wechselnde Verbindung von Glasfaser- zum Kupferkabel oder die Kommunikation über eine unbestimmte Anzahl von Zwischenknoten. Der auszuarbeitende Prototyp wird hierbei nur einen vereinfachten Anwendungsfall abdecken und als Grundlage für weitere darauf aufbauende Überarbeitungen dienen. Dennoch wird ein Großteil dieser Fälle im Laufe der Anforderungsanalyse erfasst und herausgearbeitet.
Das Ziel dieser Masterarbeit besteht darin, einen grundlegenden Prototypen für ein Schlüssel-Management-System im Rahmen eines Quantenschlüsselaustausch-Verfahrens zu entwickeln. Außerdem soll eine Bewertung gegenüber klassischen, kryptographischen Verfahren im Bereich Sicherheit stattfinden. Hierzu werden unter anderem die vorher ausgearbeiteten und möglichen Anwendungsfälle genutzt sowie Stärken und Schwächen der Quantenmechanik im Kontext einer sicheren Kommunikation aufgezeigt.
Dazu wird anfangs kurz auf das Projekt MuQuaNet eingegangen, in dessen Rahmen diese Masterarbeit stattfindet. Darauffolgend werden die Themen Quantenmechanik, der Quantenschlüsselaustausch, Kryptographie sowie die ETSI-Protokolle 004 und 014 behandelt. Dies gilt als Grundlage für das Verständnis der sich daran anschließenden weiteren Teilen dieser Arbeit. Anknüpfend wird eine Anforderungsanalyse vorgestellt, in welcher anhand verschiedener Szenarien ein Anforderungskatalog für ein Schlüsselmanagement-Programm ausgearbeitet worden ist. Darauf aufbauend schließt sich ein Programmentwurf an, wobei aus dem Ergebnis des vorherigen Kapitels zunächst der Umfang des geplanten Prototy-

pens sowie die benötigten Komponenten definiert werden. Im nächsten Kapitel wird das daraus resultierende Programm präsentiert, indem zunächst der Entstehungsprozess dokumentiert und anschließend die Funktionalitäten aufgezeigt werden. Die Ergebnisse dieser Masterarbeit werden in dem vorletzten Kapitel unter anderem auf ihre Aussagekraft und in Bezug zu bis dato herkömmliche Verfahren diskutiert. Darauf folgt am Ende eine Zusammenfassung, die die wichtigsten Punkte der Arbeit noch einmal übersichtlich darstellt. Schlussendlich wird noch ein kurzer Ausblick auf zusätzliche Weiterentwicklungen für eine Verbesserung des Prototypens gegeben.

1.2 Problemstellung

Im Rahmen dieser Masterarbeit soll eine sichere Quantenschlüsselaustausch-Anwendung (QKD-Application) sowohl entworfen als auch in Form eines Prototypens implementiert werden. Am Ende soll diese Anwendung protokollunabhängig mit QKD-Geräten über Schnittstellen kommunizieren und bei nachfolgender Kommunikation Quantenschlüssel nutzen können. Es sollen verschiedene Sicherheitsaspekte, wie bespielsweise ein Informationsaustausch über eine unsichere/öffentliche Leitung, berücksichtigt werden. Außerdem sollen der aus der Anforderungsanalyse entstandene Anforderungskatalog sowie der fertige Programmentwurf allgemein(-gültige) Rahmenbedingungen für eine spätere vollumfängliche Anwendung definieren.
Somit stellt sich die Forschungsfrage wie ein entsprechender Anforderungskatalog sowie ein sich daran orientierender Programmentwurf hergeleitet und erstellt werden können. Zusätzlich muss geprüft werden in welcher Weise ein passender Prototyp implementiert werden kann und inwiefern dieser am Ende zweckmäßig ist.

Kapitel 2

Grundlagen

Das folgende Kapitel dient dazu die (technischen) Grundlagen für das Verständnis der folgenden Arbeit näherzubringen und zu definieren. Ziel dieses Kapitels ist es, dass am Ende grundlegende Protokolle, besonders jene, die für die Entwicklung einer Schlüsselmanagementanwendung essenziell sind, verstanden sowie deren Funktion nachvollzogen werden kann. Dieses Wissen umfasst Protokolle für den Quantenschlüsselaustausch sowie jene für die verwendeten Schnittstellen. Da die Kryptographie, insbesondere die Quantenkryptographie, das Fundament hierbei bildet, soll diese ebenfalls grundlegend näher erläutert werden. Auf die jeweiligen Thematiken wird nur soweit eingegangen, wie es für das Verständnis der Arbeit notwendig ist.
Außerdem wird am Anfang kurz das Projekt MuQuaNet im Wesentlichen vorgestellt, da diese Masterarbeit im Rahmen dieses Projekt erarbeitet worden ist.

2.1 Das Projekt MuQuaNet

Das Projekt MuQuaNet, wobei die Universität der Bundeswehr München unter der Führung von Prof. Dr. Udo Helmbrecht federführend ist, verfolgt das Ziel, ein quantensicheres Kommunikationsnetzwerk innerhalb des Großraums München aufzubauen sowie später zu betreiben. Dieses Netz soll vor allem schrittweise weiteren Einrichtungen, Behörden und militärischen Institutionen zugänglich gemacht werden. Vor allem der militärische Anwendungsfall ist einer von zwei Punkten, der von vornherein im Fokus dieses Projektes steht. Es soll insbesondere eine sichere Möglichkeit entwickelt werden, Waffensysteme aus der Ferne zu warten, wobei an dieser Stelle das Einsatzszenario eine besondere Rolle spielt. Innerhalb der zivilen Verwendung ADRIAN ist unter anderem eine Infrastruktur zur Übermittlung hochsensibler Daten geplant. Dies könnte beispielsweise notwendig werden, wenn eine im Rahmen dieses Projekts durchgeführte Analyse für das Gefährdungspotenzi-

al eine bestimmte Personengruppe anschließend abhörischer an entsprechende Behörden übermittelt werden soll. Grundlage für die Entwicklung dieser Verfahren ist der Quantenschlüsselaustausch (QKD), welcher in diesem Kapitel noch näher erläutert wird. Der zeitliche Rahmen dieses Projektes ist vom 01.10.2020 bis zum 31.12.2024. [13]

2.2 (Quanten-)Kryptographie

Neben der Kryptoanalyse, welche sich mit dem Brechen von Verschlüsselungen beschäftigt, stellt die Kryptographie eine von zwei Teildisziplinen der Kryptologie dar. Sie gilt seit jeher als Schlüsseltechnik um eine sichere Kommunikation zu gewährleisten. Dazu werden Methoden, Werkzeuge und Algorithmen entwickelt, um Daten so zu chiffrieren, dass Unbefugte nicht in der Lage sind, diese auslesen zu können, Nachrichten zur Sicherstellung der Integrität zu signieren und vieles mehr. Dieses Prinzip wird bei der bis dato herkömmlichen Kryptographie genauso verfolgt wie bei der sich entwickelnden Quantenkryptographie.

2.2.1 Kryptographie im Allgemeinen

Die Kryptographie ist als ein Fünftupel (P, C, K, E, D) definiert und weist folgende Eigenschaften auf:

- P ist eine endliche Menge von möglichen Klartexten
- C ist eine endliche Menge von Chiffretexten bzw. Verschlüsselungen
- K stellte den Schlüsselraum mit allen möglichen Schlüssel als nicht leere, endliche Menge dar
- $E = \{E_k : k \epsilon K\}$ ist eine Menge von Verschlüsselungsfunktionen mit

  $$E_k = P \times K \rightarrow C$$

- $D = \{D_k : k \epsilon K\}$ ist eine Menge von Entschlüsselungsfunktionen mit

  $$D_k = C \times K \rightarrow P$$

- Für alle $k_e \epsilon K$ gibt es immer ein $k_d \epsilon K$, dass Folgendes gilt:

  $$\forall p \epsilon P : D_k(E_k(p, k_e), k_d) = p$$

Somit beschäftigt sich die Kryptographie hauptsächlich mit der Ver- und Entschlüsselung von Daten, indem ein Schlüssel $k \epsilon K$ mit dem zu verschlüsselnden Text $p \epsilon P$ kombiniert wird. Dazu werden Kryptosysteme entweder nach dem symmetrischen oder asymmetrischen Prinzip, welche beide noch erklärt werden, entwickelt und sollen möglichst nach Shannon perfekt sicher sein. Also soll nach folgender Definition bei einem gegebenen

Chiffretext jeder Klartext gleich wahrscheinlich sein:

> Ein Kryptosystem heißt perfekt sicher, falls das Auftreten eines chiffrierten Textes y unabhängig vom Klartext ist, d.h. für $\forall y \in C$ und $\forall x \in P$ gilt
>
> $$P(x|y) = P(x).$$

Da dies in der Praxis meistens unmöglich ist, versuchen fast alle kryptographischen Systeme das Brechen des verschlüsselten Textes so kompliziert und/oder aufwendig wie möglich zu machen. Dazu werden verschiedene mathematische Probleme, wie beispielsweise die Faktorisierung oder Primfaktorzerlegung, mit mathematischen Verfahren und Algorithmen genutzt. Der Inhalt der verschlüsselten Daten soll mindestens so lange geheim bleiben, wie daraus resultierende Informationen für einen selbst oder den Angreifer von Wert sind.
Somit lassen sich die vier obersten Ziele der Kryptographie ableiten, wobei Ersteres das ursprüngliche Ziel solcher Systeme ist:

- Vertraulichkeit: Inhalte können nur von berechtigten Stellen eingesehen werden bzw. vom Inhalt Kenntnis erlangen
- Integrität: Inhalte können nur von berechtigten Stellen verändert werden
- Authentizität: beide Seiten können sich zweifelsfrei als ausgegebene Stelle ausweisen
- Verbindlichkeit: Urheberschaft von versendeten Nachrichten, Dateien etc. sind im Nachhinein nicht mehr angreifbar

Je nach Anwendungsfall können und müssen die kryptographischen Systeme nicht alle Ziele erreichen, manchmal reicht die Umsetzung eines oder weniger Ziele aus, damit Daten, Nachrichten etc. vor fremden Zugriffen geschützt sind.
In den folgenden Abschnitten werden kurz die verschiedenen Arten von Verschlüsselungssystemen sowie aus beiden abgeleitete hybride System erklärt.
[7][11][15][25][30][34]

Symmetrische Verschlüsslung

Bei symmetrischen Verfahren wird für die Ver- sowie Entschlüsselung jeweils der gleiche digitale Schlüssel $k \in K$ verwendet, welcher sowohl beim Sender als auch Empfänger immer identisch ist. Des Weiteren sind die Ver- und Entschlüsselungsfunktionen $e_k \in E$

und $d_k \epsilon D$ bekannt und in kurzer Zeit berechenbar. Damit dieses kryptographische Verfahren sicher ist, ist die Schlüsselmenge K unbedingt geheim zu halten.
Andererseits liegt genau in der schnellen Berechenbarkeit ein Vorteil von symmetrischen Verschlüsselungsverfahren, da diese somit schneller als ihr asymmetrischer Gegenpart sind. Deshalb sind diese noch heute in Form von verschiedensten Systemen weitverbreitet. Das größte Problem stellt dabei ein sicherer Schlüsselaustausch zwischen alle Parteien dar. Unter diesen unterschiedlichen kryptographischen Systemen bietet das One-Time-Pad die größte Sicherheit und soll deshalb näher vorgestellt werden, auch da es eine interessante Option für die Quantenkryptographie bietet.
[7][11][25][30][34]

One-Time-Pad
Die Besonderheit bei diesem kryptographischen System ist, dass der Schlüssel $k \epsilon K$ genauso lang wie der zu verschlüsselnde Klartext $p \epsilon P$ ist und nur einmal verwendet werden kann. Durch die erste Begebenheit ist ein One-Time-Pad nach Shannon perfekt sicher, da jeder mögliche Klartext für einen Chiffretext $c \epsilon C$ gleich wahrscheinlich ist und somit der passende Schlüssel niemals berechnet werden kann. Der Grund, warum der genutzte Schlüssel nur einmal verwendet werden darf, ist, dass dieser mit dem Klartext bitweise XOR-verknüpft wird. Je häufiger also $k \epsilon K$ eingesetzt wird, desto mehr Informationen kann ein Angreifer sammeln, um diesen berechnen zu können.
[11][23][25][34]

Asysmmetrische Verschlüsselung

Bei der asymmetrischen Verschlüsselung, auch Public-Key-Verfahren genannt, besitzt jeder Teilnehmer T_n ein Schlüsselpaar. Dieses besteht aus einem privaten, geheimen Schlüssel in Form der Verschlüsselungsfunktion $d_{t_n} \epsilon D$ und einem daraus berechneten öffentlichen Schlüssel $t_n \epsilon K$, der als zwei Tupel (t_n, $e_{t_n} \epsilon E$) für jeden anderen Teilnehmer frei zugänglich ist. Somit ist ein vorheriger Schlüsselaustausch unter den Kommunikationspartnern im Gegensatz zur symmetrischen Verschlüsselung nicht notwendig.
Um nun dem Teilnehmer T_n eine verschlüsselte Nachricht zu senden, würde dessen öffentlicher Schlüssel zur Chiffrierung wie folgt genutzt werden: $e_{t_n}(p)$ *mit* $p \epsilon P$. Zur Entschlüsselung würde der Empfänger seinen privaten Schlüssel nutzen, um durch die Funktion mit $d_{t_n}(e_{t_n}(p)) = p$ an die ursprüngliche Nachricht zu gelangen.
Beim Public-Key-Verfahren werden in Bezug auf die Sicherheit die mathematischen Probleme der Faktorisierung und Primfaktorzerlegung benutzt. Es soll keine Sicherheit nach Shannon erreicht werden, sondern die Berechnung des privaten aus dem öffentlichen Schlüssel faktisch unmöglich oder in einer nicht adäquaten Zeit durchführbar sein. Allerdings

werden voraussichtlich Quantencomputer zukünftig in der Lage sein, diese mathematischen Probleme berechnen zu können. Schon heute gibt es einen entsprechenden Algorithmus, den Shor-Algorithmus.
[7][11][25][30][34]

Hybride Verschlüsselung

Die Hybride Verschlüsselung ist eine Mischform aus den anderen beiden Ansätzen. Ziel ist es, die Vorteile aus beiden Verfahren zu vereinen und somit deren Nachteil zu eliminieren.
Dabei ist der Ansatz, dass zunächst ein Schlüssel für symmetrische Verschlüsselung erzeugt und mit einer asymmetrischen Verschlüsselung gleichzeitig ein sicherer Kommunikationskanal aufgebaut wird. Über diesen wird als nächstes der erzeugte Schlüssel sicher an den Kommunikationspartner gesendet, wodurch beide einen identischen Schlüssel besitzen. Nun kann beispielsweise per One-Time-Pad eine klassische symmetrische Verschlüsselung durchgeführt werden.
Besonders beiden größeren Datenmengen kann damit ein immenser Geschwindigkeitsvorteil gegenüber dem asymmetrischen Verfahren erreicht und gleichzeitig das Risiko bei der symmetrischen Variante minimiert werden.
Somit ist dieses Verschlüsselungsverfahren auch für die Quantenkryptographie interessant, da durch Quanten(-zustände) echte Zufallszahlen erzeugt werden können.
[12][25]

2.2.2 Quantenkryptographie und deren Einfluss

Ein neuer, immer stärker aufkommende Aspekt und im Kontext dieser Arbeit grundlegender, ist die quantenbasierte Kryptographie, weshalb diese ebenfalls kurz erläutert wird.
Anders als bei der klassischen Kryptographie bietet die quantenbasierte Verschlüsselung keine Algorithmen, mit denen durch die Nutzung von Quanten eine Datei, Nachricht etc. verschlüsselt wird. Vielmehr werden die Kenntnisse aus der Physik, vornehmlich der Quantenmechanik genutzt, um Werkzeuge bereit zu stellen, mit denen schlussendlich nur ein Schlüsselaustausch im Rahmen einer hybriden Verschlüsselung stattfindet. Die Sicherheit hinter diesem System - dem Quantenschlüssaustausch, im englischen Quantem Key Distribution (QKD) - basiert auf den physikalischen Eigenschaften der genutzten Photonen und nicht auf mathematischen Problemen. Hierbei werden verschiedene Protokolle, wie beispielsweise das BB84, genutzt. Im folgenden Abschnitt sollen nun sowohl die QKD als auch das BB84 näher erklärt werden.
[20][25]

2.3 Quantenschlüsselaustausch

Die Quantem Key Distribution (QKD), auf Deutsch Quantenschlüsselaustausch, ist ein Ansatz, um eine unbedingte Sicherheit bei der Kommunikation zu erreichen. Die bisherigen symmetrischen sowie asymmetrischen Verschlüsselungsverfahren, welche im vorherigen Abschnitt vorgestellt worden sind, bieten nur eine auf der Mathematik und fehlender Rechenleistung basierende Sicherheit. In beiden Fällen kann der komplette, verschlüsselte Datenverkehr von einer dritten Partei mitgeschnitten und abgespeichert werden. Sobald ein mathematisches Problem berechenbar wird oder eine entsprechende Rechenleistung, beispielsweise in Form von Quantencomputern, zur Verfügung steht, sind diese Verfahren nicht mehr sicher. Diese (zukünftige) Sicherheitslücke nutzt QKD, da die erstellten Schlüssel nicht maschinell durch einen Pseudo Random Number Generator (PRNG) gewonnen werden, sondern auf einer echten physikalischen Sicherheit beruhen.
Die Idee ist, dass das Schlüsselmaterial durch die Polarisierung von Photonen gewonnen wird. Es gibt zwar noch weitere Ansätze wie beispielsweise Elektronen oder Ionen zu nutzen, aber diese erweisen sich als praktisch nicht umsetzbar. Ein weiterer Sicherheitsaspekt ist das No-Cloning-Theorem, welches besagt, dass es unmöglich ist eine perfekte Kopie der Quantenzustände zu erstellen. Denn nach den Gesetzen der Quantenphysik wird bei jeder Messung eines Zustandes dieser verändert. Es ist zusätzlich unmöglich den ursprünglichen Zustand wiederherzustellen, da jedes Photon vor einer Messung jeden möglich Zustand gleichzeitig annimmt. Da jede noch so minimale Extrahierung von Quantenzuständen eine Form der Messung ist, würde das Eingreifen einer dritten Partei in die Kommunikation anhand von beispielsweise statistischen Tests auffallen. Durch diese auf den physikalischen Gesetzen basierte Sicherheit stellt die QKD eine Lösung für Langzeitsicherheit, eventuell eine dauerhafte, dar.
Schon heute wird intensiv an der Weiterentwicklung dieser Technologie sowie dem Aufbau von Quantum Communication Infrastructure (QCI), auch im Rahmen des Projekts MuQuaNet, geforscht. Das übergeordnete Ziel hierbei ist eine totale Kommunikationssicherheit innerhalb sowie zwischen kritischen Infrastrukturen. Dies reicht von Finanztransaktionen bis hin zu militärischen Anwendungsszenarien, wie beispielsweise die Fernwartung eines Waffensystems im Einsatz.
Da die Quantenschlüssel sicher sind, ist das Schlüsselmanagement von besonderer Bedeutung, sodass nicht nur eine sichere Verwaltung dieser, sondern besonders eine sichere Übertragung, garantiert ist. Allerdings muss zunächst ein entsprechender Schlüssel anhand von QKD-Protokollen erzeugt werden. Für dieses Problem gibt es mehrere protokollarische Lösungen, welche wären:

- BB84
- E91

- BBM92
- B92
- Six-State (SSP)
- Differential-Phase-Shift (DPS)
- SARG04
- Coherent One-Way (COW)
- S13

Im Rahmen dieser Masterarbeit wird im folgenden Abschnitt nur das Protokoll BB84 näher erläutert, da die Laborgeräte im Rahmen des Projektes MuQuaNet alle Verfahren wie beispielsweise COW nutzen, die vom BB84-Protokoll abgeleitet sind. Eine kurze Erklärung zu den restlichen QKD-Protokollen befindet sich im Anhang unter [26].
[6][12][17][26][29][30]

2.3.1 Protokoll BB84

Das Protokoll BB84, welches 1984 von Charles H. Bennett und Gilles Brassard veröffentlicht worden ist, ist das erste Protokoll in der Quantenkryptographie.
Dabei nutzt BB84 die vier Quantenzustände $|\updownarrow\rangle$ (90° polarisiert), $|\leftrightarrow\rangle$(0° polarisiert), $|\nwarrow\rangle$(-45° polarisiert) und $|\nearrow\rangle$(45° polarisiert), wovon jeweils zwei zusammen eine der beiden Basen $\bigoplus$, $\bigotimes$ bilden. Durch Zuhilfenahme der Polarisierung von Lichtphotonen wird ein Schlüssel sicher vom Sender zum Empfänger übertragen, wobei beide Seiten diese rein zufällig filtern. Nach einem Vergleich der Ergebnisse hat jeder einen von Fehlern bereinigten Schlüssel, welcher mit dem seines Gegenüber identisch ist.
Im folgenden Abschnitt wird der Sender Alice, der Empfänger Bob und ein möglicher Angreifer Eve genannt. Außerdem wird der Protokollablauf als ideal betrachtet und in der Praxis unvermeidlich auftretendes Rauschen aufgrund von technischen Unausgereiftheiten oder Übertragungsfehlern nicht berücksichtigt.
Im ersten Schritt einigen sich beide Seiten, welchen Wert, 0 oder 1, die jeweilige Polarisierungsrichtung repräsentieren soll, beispielsweise $|\updownarrow\rangle = 0$ und $|\leftrightarrow\rangle = 1$. Der Sender entnimmt einer entsprechenden Quelle eine beliebig lange Photonensequenz. Jedes einzelne Element wird nun komplett zufällig in einen der vier Quantenzustände ($|\updownarrow\rangle$, $|\leftrightarrow\rangle$, $|\nwarrow\rangle$ oder $|\nearrow\rangle$) versetzt. Diese Sequenz sendet Alice über einen Quantenkanal an Bob.
Als nächstes wählt Bob für jedes einzelene Photon der Sequenz rein zufällg eine der beiden Basen, $\bigoplus$ oder $\bigotimes$, aus, um jedes Element nach Zustand messen zu können. Dabei wird der Empfänger in 50% aller Fälle eine falsche Basis wählen, da dieser die ursprünglichen Polarisierungsrichtungen nicht kennt. Am Ende dieses Schrittes haben beide Seiten jeweils eine Liste mit polarisierten Photonen, wobei diese noch nicht komplett übereinstimmen. Eine Auflistung aller möglichen Messkombinationen ist der Tabelle auf der Abbildung 2.1

zu entnehmen.
Als letztes vergleichen Alice und Bob, welche Basis sie jeweils pro Photon gewählt haben, beispielsweise indem Alice eine solche Liste übersendet. Der Empfänger streicht dann einfach die Elemente, bei denen er eine falsche Basis gewählt hat und teilt dem Sender mit, welche dies sind, beispielsweise: Streiche die Einträge Nr. 2, 7, 13, 14, 21 (nicht in Tabelle enthalten, dient nur zur Veranschaulichung). Der Vergleich kann dabei über eine öffentliche, nicht sichere Leitung geschehen, da keine näheren Informationen zum Schlüssel selbst verschickt werden, wobei ein MitM-Angriff durch eine authentifizierte Leitung verhinderbar ist. Bei diesem Prozess kann der später entstandene Schlüssel weder vorausberechnet oder in sonst einer Art und Weise vorhergesagt werden, da die wichtigen Entscheidungen in diesem Ablauf komplett zufällig getroffen werden.

Alice	Bob's Basis	Bob's Messung	Bob's Ergebnis	Kompatibilität	Schlüssel
$\|\updownarrow\rangle$	$\oplus$	$\|\updownarrow\rangle$	1	✓	1
$\|\updownarrow\rangle$	$\otimes$	$\|\nwarrow\rangle,\|\nearrow\rangle$	0,1	✗	-
$\|\leftrightarrow\rangle$	$\oplus$	$\|\leftrightarrow\rangle$	0	✓	0
$\|\leftrightarrow\rangle$	$\otimes$	$\|\nwarrow\rangle,\|\nearrow\rangle$	0,1	✗	-
$\|\nwarrow\rangle$	$\otimes$	$\|\nwarrow\rangle$	0	✓	0
$\|\nwarrow\rangle$	$\oplus$	$\|\leftrightarrow\rangle,\|\updownarrow\rangle$	0,1	✗	-
$\|\nearrow\rangle$	$\otimes$	$\|\nearrow\rangle$	1	✓	1
$\|\nearrow\rangle$	$\oplus$	$\|\leftrightarrow\rangle,\|\updownarrow\rangle$	0,1	✗	-

Abbildung 2.1: Tabelle aller Kombinationsmöglichkeiten beim Protokoll BB84
Quelle: [25]

Nun besteht allerdings noch die Möglichkeit, dass ein Angreifer Eve den Datenverkehr mitschneiden und/oder auslesen möchte. Eine Angriffsoption wäre die sogenannte Intercept and Resend-Attacke, bei der die an Bob gerichtete Nachricht abgefangen wird. Eve wählt zufällig für jedes Photon einer der beiden Basen, führt die Messungen durch und sendet die daraus resultierenden Ergebnisse an Bob weiter, als wäre es eine Nachricht direkt von Alice. In etwa 50% der Fälle wird der Angreifer mit der Wahl der Basis richtig liegen und keine der beiden anderen Parteien bemerkt diesen Kommunikationseingriff. Bei den Fällen, bei denen Eve falsch liegt, hat Bob bei seinen Messungen nur eine 50%ige Wahrscheinlichkeit, dass mit seinen Messungen ein richtiges Ergebnis erreicht wird. Beispiel: Alice sendet ein Photon mit der Polarisierung $|\nwarrow\rangle$, Eve misst mit der Basis $\otimes$, sendet an Bob $|\updownarrow\rangle$ und dieser wählt die Basis $\oplus$. Wenn ein Angreifer also in den Datenaustausch eingreift, dann sind rund 25% aller durchgeführten Messungen falsch. Somit lässt sich mit einfachen statistischen Hilfsmitteln überprüfen, ob Informationen abgegrif-

fen worden sind. Alice und Bob wählen öffentlich eine zufällige Sequenz des Schlüssels, welche anschließend verworfen wird, und nutzen diese als Stichprobe, um die Fehlerrate im Quantenkanal abschätzen zu können. Liegt die erreichte Fehlerrate in einem entsprechend hohen Bereich und es wird ein sogenanntes Fehlerkorrektur-Protokoll angewendet, um die fehlerhaften Elemente zu entfernen. Abschließend ist somit zu sagen, dass die Sicherheit vom Protokoll BB84 nicht in der Verhinderung von Angriffen liegt, sondern viel mehr im Erkennen von möglichen Lauschangriffen, damit diese abgewehrt werden können.
[11][25][29]

2.4 ETSI-Protokolle 004 und 014

Im Rahmen dieser Arbeit sind die ETSI-Protokolle 004 sowie 014 von besonderem Interesse. Denn mit diesen Standards werden Schnittstellen zwischen einem Quantenschlüsselmanagement und Anwendungen, welche diese Schlüssel nutzen, definiert und spezifiziert. Denn:

„Neue Standards sind erforderlich, um die Quantenkommunikation in Netzwerke zu integrieren und ihre Kommerzialisierung zu stimulieren “[9]

Im folgenden Abschnitt soll ein Überblick über die technischen Grundlagen für eine Schlüsselnutzung die Nutzung der Protokolle ETSI 004 und 014 geschaffen werden. Dafür werden diese beiden Protokolle kurz näher erläutert, indem ihre Funktionsweise und auch grundlegende Unterschiede zwischen beiden herausgearbeitet werden. Für ein besseres Verständnis werden in diesem Zusammenhang darauffolgend sowohl eine REST-API, das JSON-Format als auch HTTPS mit dem zugrundeliegendem TLS-Protokoll erklärt, welche insbesondere für ETSI 014 von entscheidender Bedeutung sind.
Voranstehend befindet sich ein kurzes Glossar, um verwendete Begriffe und deren Abkürzungen auf Grundlage der Quellen [2] sowie [32] kurz und bündig zu erläutern.

- Quality of Service
  Unter Quality of Service (QoS) werden alle Dienste zusammengefasst, welche einer Anwendung zur Verfügung gestellt werden und gibt dabei an wie sehr die qualitativen Anforderungen erfüllt worden sind, beispielsweise ob und wie Daten eines Livestreams mit einer vorher festgelegten Qualität beim Empfänger ankommen.
- Secure Application Entity
  Stellt ein Objekt dar, welches mindestens einen Quantenschlüssel von einem KME für mindestens eine Anwendung bezieht und mit anderen SAE's zusammenarbeiten kann

- Key Management Entity
  Eine KME ist ein Objekt, welches die Schlüsselmenge innerhalb eines Netzes alleine oder mit anderen Managementobjekten verwaltet
- QKD Entity
  Ein Objekt, das unter Berücksichtigung von dem verwendeten QKD-Protokoll(en) eine Schlüsselverteilfunktion zur Verfügung stellt, wobei mindestens eine weitere QKDE mit einbezogen wird
- Trusted Node
  Ein als sicher eingestufter Knoten innerhalb eines Netzes mit vertrauenswürdigen Geräten. Enthält mindestens eine KME und eine QKDE.

2.4.1 ETSI 004

ETSI 004 ermöglicht die Implementierung einer grundlegenden, objektbasierten Schnittstelle für eine Schlüsselübergabe, welche so aufgebaut sein soll, „dass sie so klein wie möglich ist und gleichzeitig sehr flexibel ist“ [2]. Bei der Implementation ist es möglich, entweder eine oder mehrere Schlüsselmanagement-Ebenen einzubauen und somit dem Anwendungsfall entsprechend anpassbar sein zu können, wobei die bei ETSI 004 definierte Schnittstelle für jede dieser Ebenen gültig sein kann. Nicht in diesem Standard mit inbegriffen sind Vorgaben der Funktionsweise des Schlüsselmanagementsystems, dies obliegt dem Anwender. ETSI 004 ist in der Lage einen kontinuierlichen Schlüsselstrom aus mindestens zwei Remote-Anwendungen zu generieren und diesen den entsprechenden Anwendungen bereitzustellen. Dazu werden Sitzungen erstellt, denen jeweils ein Quality of Service zugewiesen werden kann. Die damit einhergehenden Anforderungen der QoS werden vom Netzwerk beispielsweise durch eine Priorisierung organisiert. Durch die Einführung der Quality of Services und dessen Funktionalitäten soll es zu einer besseren Auslastung von QKD-Diensten kommen und es werden gleichzeitig Metadaten eingeführt. Diese haben den Nutzen, Netzwerkinformationen, wie Statistiken, sammeln und auslesen zu können. Schnittstellen sind bei ETSI 004 implementierungsunabhängig, was eine größere Freiheit für die Programmierer bedeutet, da es egal ist, welche Programmiersprache, Bibliotheken etc. verwendet werden. Dies verfolgt dem anfänglich genannten Ansatz der Flexibilität, weil es dadurch jederzeit möglich ist, sich den gegebenen Umständen und Anforderungen angepasst werden kann. [2][32]
Auf der Abbildung 2.2 ist ein einfaches Anwendungsbeispiel für ETSI 004 zu sehen, welches ausreicht, um dessen grundlegende Funktionsweise näher zu erläutern. Der Aufbau ist wie folgt:

- Einzelner QKD-Link mit zwei Endpunkten (A und B)
- Jeweils eine Anwendung (gelb markiert)

- Jeweils ein QKD-Modul (blau umrandet) mit:
  - QKD-Schlüsselmanagement (grün markiert)
  - QKD-Protokoll für Schlüsselerzeugung (rot markiert)

Für eine sichere Kommunikation zwischen den Anwendungen von A und B werden Quantenschlüssel für die Verschlüsselung genutzt. Diese werden über eine nach dem ETSI-004-Standard definierte Schnittstelle vom jeweiligen QKD-Modul angefordert und empfangen. Beide Module von A und B sind per QKD-Link miteinander verbunden, sodass unter anderem eine Schlüsselsynchronisation stattfinden kann, um sicherzustellen, dass die Schlüsselmenge A gleich der Schlüsselmenge B ist. Dadurch kann die Peer-Anwendung jederzeit an einen identischen Satz von Schlüssel an beiden Orten (A und B) gelangen. [2]

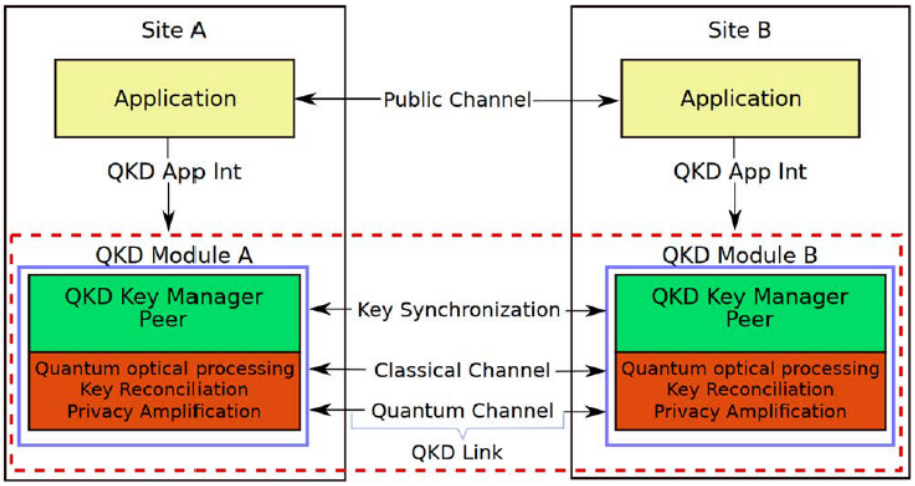


Abbildung 2.2: Darstellung einer einfachen QKD-Anwendungsschnittstelle via ETSI 004
Quelle: [2]

2.4.2 ETSI 014

ETSI 014 ist eine Schlüsselanforderungsschnittstelle, welche nach dem Anfrage-Antwort-Prinzip funktioniert. Die Anfrage findet immer anhand einer REST-Request statt und da dies somit eine auf REST basierende Schnittstelle ist, geschieht dies unter Verwendung von HTTPS-Protokollen und in einem JSON-Format. Der Ablauf dieses Vorgangs funktioniert, stark vereinfacht, wie folgt: Die Secure Application Entity, beispielsweise ein Verschlüsselungsmodul, fordert bei der Key Management Entity, welche sich am selben,

sicheren Ort bzw. im selben, sicheren Netz befindet, einen Schlüssel per Anfrage an. Die KME bearbeitet diese und antwortet mit der Überlieferung eines Schlüssels, welche mit einer QKD-Technologie vorher sicher erzeugt worden ist. [2][32]
Im Normalfall gibt nicht nur diesen einen Standort sondern, wie auf Abbildung 2.3 zu sehen, mehrere Standorte innerhalb eines QKD-Netztes, die miteinander kommunizieren und gemeinsam genutzte Schlüssel an unterschiedliche Orte liefern. Es kann entweder eine einfache Verbindung mit einem QKD-Link bestehen oder mehrere solcher QKD-Links vorherrschen, wofür die Abbildung 2.3 ein Beispiel ist. Der genaue Aufbau sowie das Management eines solchen Netzes liegen außerhalb des Rahmens des ETSI-Protokolls 014. Es wird allerdings vorgegeben, dass an einem Standort sich mindestens eine Secure Application Entity befindet, welche mit der Key Management Entity, die sich innerhalb eines Trusted Nodes befindet, über eine mit HTTPS-Protokollen spezifizierte Schnittstelle kommuniziert. Dabei muss mindestens die TLS-Version 1.2 verwendet werden. Da sich per HTTPS verständigt wird, müssen Nachrichten in einem JSON-Format versendet werden. Innerhalb der TNs verfügte die KME über mindestens eine QKD Entity, welche mit anderen QKDEs an anderen Standorten per QKD-Link verbunden sind und somit auch die KMEs untereinander austauschen können. Für Kommunikation innerhalb des QKD-Netztes muss jede SAE sowie KME eine eindeutige ID besitzen, damit sie sich authentifizieren kann. Genauso wie jeder Schlüssel eine eindeutige ID besitzt, um ihn unter anderem von eine KME anfordern zu können. Außerdem nimmt das ETSI-Protokoll 014 folgende Gegebenheit an: Alle Trusted Nodes, Secure Application und Key Management Entitys sind sicher sowie dass die Schnittstelle zwischen SAE und KME innerhalb eines sicheren Standortes eingebettet ist. [32]

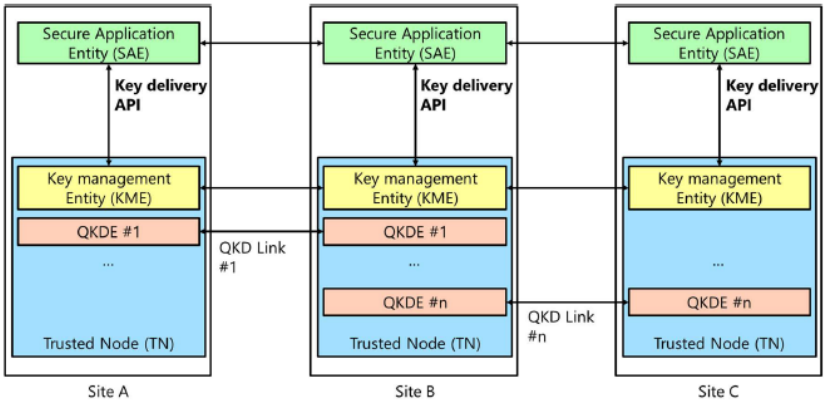


Abbildung 2.3: Darstellung eines einfachen Beispiels für ein QKD-Netz via ETSI 014
Quelle: [32]

Auf der Abbildung 2.4 wird nun die Funktionsweise einer Schlüsselübergabeschnittstelle(Key Delivery API) dargestellt, wobei die vorher beschrieben QKD-Netzstruktur als gegeben gilt. Es werden zwei Standorte, A und B, mit ihrer jeweiligen SEA, KME sowie API. Die Verbindung zwischen KME A und KME B sowie SAE A und SAE B wird nicht näher definiert, kann somit also eine Direktverbindung oder über ein Netz sein. Über diese Verbindung werden die Quantenschlüssel unter KME A und KME B getauscht, sodass an beiden Standorten dieselbe Schlüsselmenge besteht. [32]

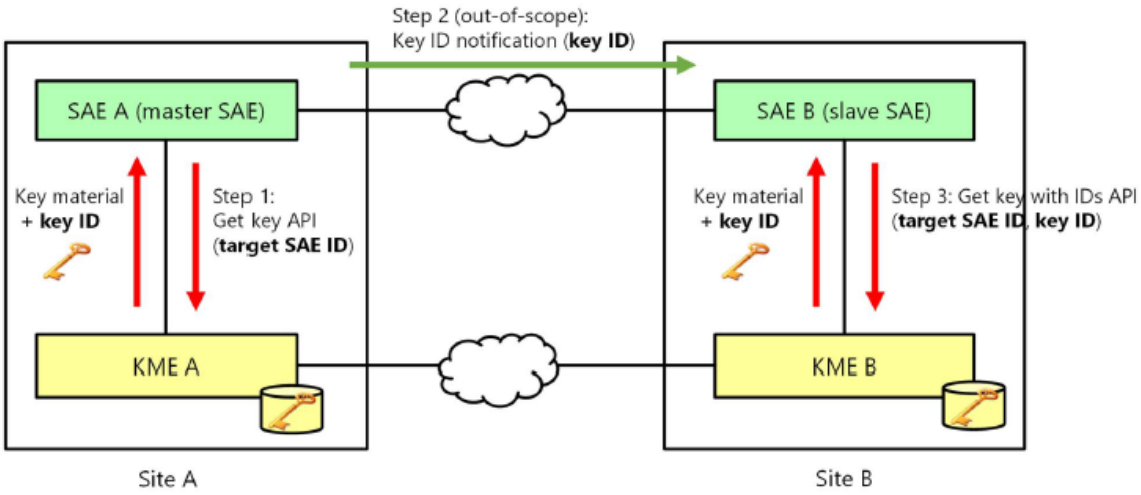


Abbildung 2.4: Darstellung einer möglichen Nutzung der Schlüsselübergabe-API via ETSI 014
Quelle: [32]

Die Kommunikation zwischen zwei Punkten A und B ist nach einem Master-Slave-Prinzip geregelt, wobei der anfragende Part immer der Master und das Gegenüber immer der Slave ist. Der Gesamtablauf lässt sich in drei aufeinanderfolgende Schritte unterteilen. Diese wären:

- **Schritt 1:**
  Am Standort A fordet die Secure Application Entity über die Schnittstelle per HTTPS-Request einen Schlüssel sowie die zugehörige ID an. Die KME A bearbeitet die Anfrage und liefert das Schlüsselmaterial.
- **Schritt 2:**
  Damit die SAE B denselben Schlüssel verwenden kann, wird die entsprechende ID von der SAE A an SAE B weitergeleitet. Der Vorgang geschieht in einer nicht näher vom ETSI-Protokoll definierten Form.

- **Schritt 3:**
  Nun fordert die Secure Application Entity B ihrerseits über eine Schnittstellemethode den zur überlieferten ID gehörigen Schlüssel bei seiner KME an. Nachdem diese geliefert worden sind, verfügen nun beide Seiten über identische Schlüssel und können mit der eigentlichen Kommunikation starten.

[32]

Unterschied zu 004

Beide im vorherigen Abschnitt vorgestellten Standards weisen einige Unterschiede zueinander auf, wovon im folgenden Abschnitt einige aufgezeigt werden sollen.
Der grundlegendste Unterschied zwischen beiden Protokollen ist, dass ETSI 014 auf einer REST-basierten Schnittstelle aufbaut und ETSI 004 auf einer Remote Function Call-basierten. Des Weiteren liefert 014 einzelne Schlüsselblöcke nach einer Anforderung per REST-Request, während 004 Sitzungen aufbaut und Schlüsselströme überträgt, aus denen per Key Stream ID der gewünschte Schlüssel extrahiert werden kann. Da das ETSI-Protokoll 014 mit HTTPS-Protokollen funktioniert, ist es bei der Implementierung notwendig entsprechende Bibliotheken und Frameworks zu berücksichtigen. Dies sorgt einerseits für gewisse Einschränkungen, andererseits erleichtert die Verwendung von HTTPS-Protokollen sowie den notwendigen JSON-Formaten die Implementierung. Das ETSI-Protokoll 004 hingegen benötigt dies nicht und lässt sich somit kompakter programmieren und ist zusätzlich an jede Situation anpassbarer. [2][32]

REST-API

Durch das anfängliche schnelle Wachstum des WorldWideWebs, gab es sehr schnell infrastrukturelle Probleme. Eine Lösung hierfür wurde von Roy Fielding erarbeitet, indem die Schnittstelle „Representational State Transfer - Application Programming Interface“ oder auch REST-API erdachte, um den Informationsaustausch zwischen verteilten Systemen zu ermöglichen. Die REST-API sorgt hierbei für die Kommunikation zwischen einem Client und einem Server, im erdachten Anwendungsszenario ein Webservice.[22]
Aus der Webarchitektur lassen sich sechs Prinzipien ableiten, welche ebenfalls für die REST-API gelten, aber nicht explizit vorgeschrieben werden. Diese sind Folgende:

- **Client-Server-Modell**
  Anhand eines Client-Server-Modells, indem beide jeweils andere Rollen übernehmen, soll das Nutzerinterface von der Datenhaltung getrennt sein. Dadurch kann jede einzelne Komponente komplett unabhängig vom Rest konzipiert werden, solange die Kommunikation über die Schnittstelle sichergestellt ist.

- **Einheitliche Schnittstelle**
  Essenziell für die Kommunikation zwischen den Komponenten ist die einheitliche Verwendung der Schnittstelle, ansonsten bricht jene komplett zusammen. Informationen müssen dadurch zwar zwangsweise in eine standardisierte Form gebracht werden, was zu einer verminderten Effizienz führen kann, aber somit wird eine vereinfachte Architektur erreicht. Der Informationsfluss wird durch diese Einheitlichkeit einseh- und nachvollziehbarer.
- **Layered System**
  Die mehrschichtigen, hierarchischen Systeme, auf welche bei REST-APIs gesetzt wird, ermöglichen den transparenten Einsatz von Vermittlern („Intermediaries“), um unter anderem für eine Umsetzung der Sicherheitsmerkmal und ein Load Balancing zu sorgen.
- **Cache**
  Durch das server-seitige Kennzeichnen einer Information als „cacheable“ oder „non-cacheable“ ist es dem Client möglich, diese zu speichern und sie bei späteren, gleichartigen Anfragen wiederzuverwenden. Dadurch wird Effizienz innerhalb des Netzwerks erhöht, da zum Beispiel sowohl die Zuverlässigkeit als auch die Erreichbarkeit von Diensten gesteigert wird.
- **Zustandslosigkeit**
  Da der Client und Server zustandslos miteinander kommunizieren, ist es dem Server nicht möglich, auf vorher gespeicherte Inhalte zurückgreifen zu können. Deshalb muss bei jeder Anfrage der Client alle benötigten Informationen mitsenden. Dies steigert die Skalierbarkeit des Netzes, weil somit der Großteil der Komplexität einer Kommunikation client-seitig ausgelagert wird und ein Server somit mehr Client bedienen kann.
- **Code-On-Demand**
  Dieser Constraint ist optional und wird lagespezifisch umgesetzt. Es ermöglicht dem Server temporäre Übertragungen von ausführbaren Programmen oder Programmteilen, wie beispielsweise Plug-ins, an den Client, um Funktionen zu erweitern.

[22]

JSON

Nach [28] ist das Dateiformat JSON (JavaScript Object Notation) das beliebteste, wenn es um das Senden von API-Anfragen sowie -Antworten über das HTTP(S)-Protokoll geht und basiert auf der Programmiersprache JavaScript. JSON wird immer dann eingesetzt, wenn Funktionen auf einem Rechner ausgeführt werden sollen, wofür die Anfrage bzw. der Auftrag von einem anderen, entfernten Rechner gekommen ist. Denn hierbei ist ein

entsprechendes Protokoll notwendig, welches diese Anfragen und Antworten exakt managt. In diesem Kontext findet das Dateiformat seine Anwendung. Die versendeten Nachrichten sind Textdateien und jede einzelne stellt ein JSON-Objekt dar. [28]
Für die Veranschaulichung dient ein einfaches Beispiel für eine Anfrage mit der dazugehörigen Antwort über eine REST-API, ob eine Filiale in einer Stadt noch geöffnet ist:

Anfrage:
{'State': 'Bavaria', 'City': 'Munich'}
Antwort:
{'timestamp': '09/04/2021 11:52:23', 'status': 'open', 'State': 'Bavaria', 'City': 'Munich', 'closing time': '7pm'}

Das HTTPS-Protokoll

Das Protokoll Hypertext Transfer Protocol Secure oder abgekürzt HTTPS ist eine hinsichtlich der Sicherheit verbesserte Version des Vorgängers HTTP (Hypertext Transfer Protocol), wodurch die Vertraulichkeit sowie Integrität während des Informations- und Datenaustausches im Internet erhöht wird. Der Ansatz hierbei ist, dass zwischen den Schichten 4 (Transport Layer via TCP) und 5 (Session Layer via HTTP) des ISO/OSI-Schichtenmodells eine Zwischenschicht, der Transport Layer Secruity, einfügt wird. Hierbei wird unteranderem die Authentifizierung von den Kommunikationspartnern über digitale Zertifikate durchgeführt, wobei allerdings häufig nur die Webserver-Authentifizierung durchgeführt wird. Außerdem werden Webseiten gezwungen, dass keine Skripte Quellen geladen werden könne, die nicht als sicher gelten. Mit diesen genannten und anderen Maßnahmen wird der Web-Verkehr möglichst sicher gestaltet. Das dafür zuständige TLS-Protokoll wird im nachfolgenden Unterkapitel noch einmal näher erläutert, da dessen Funktion grundlegend ist. [10]

Das TLS-Protokoll

Transport Layer Secruity, kurz TLS, ist ein hybrides Verschlüsselungsprotokoll, welches für das obige HTTPS von größter Wichtigkeit ist. Die Kommunikation zwischen Client und Server lässt sich hierbei in zwei Phasen unterteilen. Als erstes findet der Verbindungsaufbau durch eine Authentifikation statt und um danach in die zweite Phase mit der verschlüsselten Datenübertragung überzugehen. Bei TLS basieren alle definierten Protokolle auf dem TLS Record Protocol, wodurch diesem eine zentrale Rolle zukommt. Im Folgenden werden die vier Wichtigsten kurz vorgestellt:

- **Handshake Protocol**
  Hierbei werden die verschiedenen Parameter für die Sitzung ausgetauscht.

- **Change Cipher Spec Protocol**
  Dieses ist für die Änderung der kryptografischen Sitzungsparameter zuständig.
- **Alert Protocol**
  Anhand dessen wird der Sitzungspartner über auftretende Fehlerzustände informiert.
- **Application Data Protocol**
  Mit diesem Protokoll werden die Daten zerlegt, komprimiert, danach verschlüsselt und schlussendlich übertragen.

Da das Handshake Protokoll als erstes ausgeführt wird und im Rahmen dieser Arbeit für die Kommunikation während des Quantenschlüsselaustausches von großer Wichtigkeit ist, wird dieses im Folgenden näher erläutert. [24]

Der TLS-Handshake

Während des TLS-Handshakes werden zwischen Client und Server verschiedene Nachrichten ausgetauscht, um die für einen sicheren Verbindungsaufbau benötigten Sicherheitsparameter auszuhandeln und eine Authentifizierung durchzuführen. Auf der Abbildung 2.5 ist dieser in vier Phasen unterteilbare Ablauf in einer vereinfachten Form anschaulich dargestellt.

- **Phase 1 - Verbindungsaufnahme vom Client**
  Mit der Nachricht *ClientHello* an den Server wird der Verbindungsaufbau gestartet. Hierbei werden mehrere Parameter vor, wie beispielsweise eine Session ID oder eine Liste für den Client verwendbare Cipher Suites.
- **Phase 2 - Antwort vom Server**
  Mit der Antwort des Servers *ServerHello* werden unter anderem die Session ID und die auf sich zu einigenden Parameter wie der Cipher Suite übermittelt. Die anderen drei Nachrichtenformate, *ServerCertificate*, *ServerKeyExchange* (nur wenn Wahl ein Pre-Shared Key Suite war) sowie *CertificateRequest*, sind optional. Allerdings wird in den meisten Fällen ein Zertifikat seitens des Servers aber nicht vom Client gesendet oder angefordert. Das *ServerHelloDone* am Ende dieser Phase signalisiert dem Client, dass der Server die vom Protokoll geforderten Schritte beendet hat und auf Antwort vom Client wartet.
- **Phase 3 - Antwort vom Client**
  Falls gefordert sendet der Client zunächst die Nachricht *ClientCertificate* mit dem enthaltenen Zertifikat. *ClientKeyExchange* wird direkt als Erstes oder nach Übermittlung des Zertifikats als Antwort gesendet und ist vom vorher gewählten Cipher Suite abhängig. Wenn der Client sein Zertifikat an den Server gesendet haben sollte,

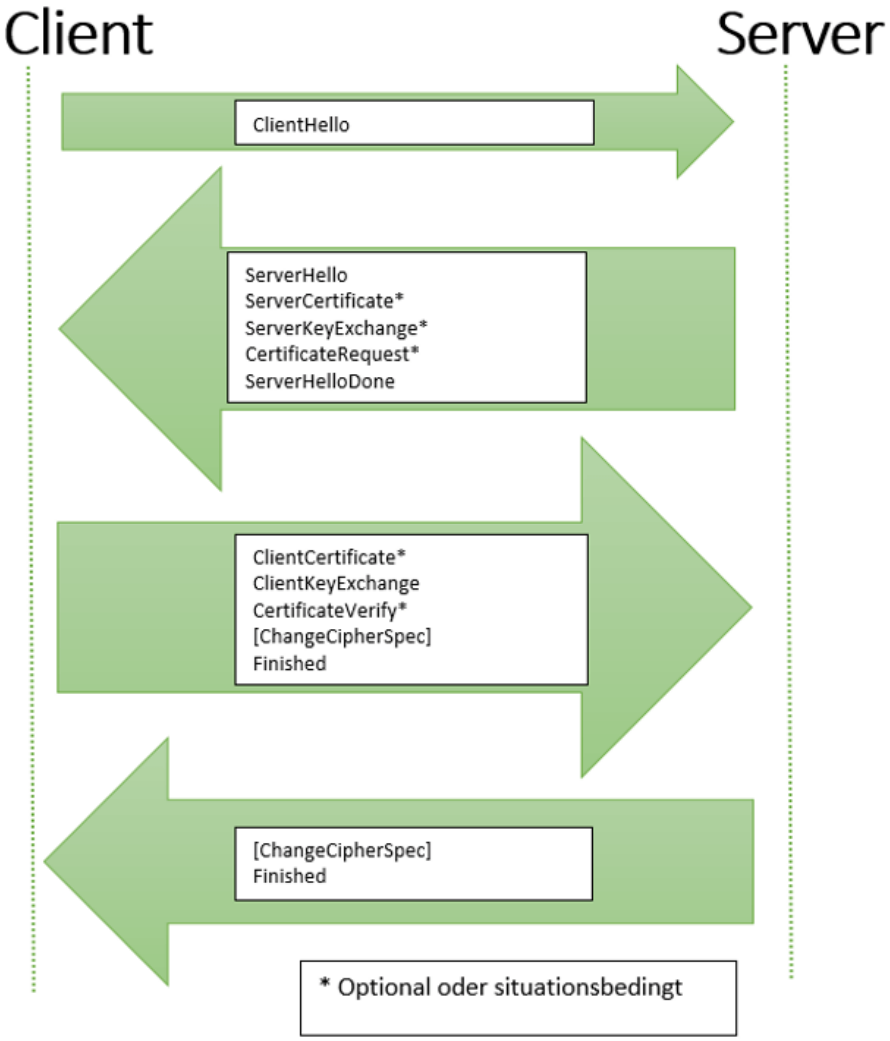


Abbildung 2.5: Vereinfachte Darstellung eines TLS-Handshake-Protokolls
Quelle: [16]

dann wird mit *CertificateVerify* eine Signatur an den Server weitergeleitet, sodass dieser sicherstellen kann, dass das Zertifikat auch zum Client gehört. Die Nachricht *ChangeCipherSpec* signalisiert dem Server, dass dessen Gegenüber von diesem Zeitpunkt an mit dem ausgehandelten, kryptographischen Verfahren kommuniziert. Mit *Finished* ist der TLS-Handshake clientseitig beendet und es wird auf die letzten Schritte des Kommunikationspartners gewartet.

- **Phase 4 - Ende des Handshakes**
  Zunächst teilt der Server ebenfalls mit der Nachricht *ChangeCipherSpec* mit, dass dieser ebenfalls den Status gewechselt hat und mit demselben Verfahren kommuniziert. Als Letztes wird noch per *Finished* der Handshake serverseitig und somit komplett als erfolgreich durchgeführt beendet.

Wenn der TLS-Handshake nach dem oben genannten Schema erfolgreich durchgeführt worden ist, können nun die entsprechenden Daten geschützt zwischen Client und Server verschickt werden. [1][3]

Kapitel 3

Anforderungsanalyse

Nachdem im vorangegangenen Kapitel die Grundlagen zum (technischen) Verständnis dieser Masterarbeit vermittelt worden sind, soll in den folgenden Abschnitten eine Anforderungsanalyse für die Umsetzung eines Schlüsselmanagementprogramms im Rahmen eines Quantenkommunikationsnetzes erarbeitet werden. Dazu wird zunächst die angewandte Methodik erläutert, um die Nachvollziehbarkeit der darauffolgenden Analyse sicherzustellen. Im Anschluss wird die entsprechende Analyse methodisch strukturiert dargelegt, damit im nächsten Kapitel hieraus ein passender Programmentwurf angefertigt werden kann.

3.1 Methodik bei dieser Anforderungsanalyse

Die nachfolgende Anforderungsanalyse orientiert sich an der objektorientierten Modellierungstechnik (OMT) nach James Rumbaugh sowie der objektorientierten Softwareentwicklung (OOSE) nach Ivar Jacobson. Die Darstellung erfolgt mit unterschiedlichen UML-Diagrammtypen, anhand dessen eine immer feingliedrigere und spezifischere Analyse durchgeführt wird. Das Ziel ist eine nachvollziehbare Beschreibung der ermittelten Anforderungen.
Dazu werden im ersten Schritt aus der Abbildung 3.1 verschiedene Kommunikationsszenarien abgeleitet und kurz die jeweilige Ausgangssituation beschrieben. Danach werden jeweils die Vor- und Nachbedingung aufgelistet, um danach den Standardablauf zu präsentieren. Da es währenddessen zu verschiedenen Fehlern kommen kann, werden zusätzlich noch die Alternativabläufe ausgearbeitet, damit ein geregeltes Verhalten festgelegt werden kann. Aus den letzten beiden Schritten werden bei jedem Szenario die anwendungsspezifischen sowie allgemeinen Anforderungen abgeleitet.
Durch diesen strukturellen Ablauf der Anforderungsanalyse soll am Ende ein Ergebnis in

der Form eines Anforderungskataloges vorliegen, aus welchem sich die Möglichkeit ergibt einen passenden Programmentwurf entwickeln zu können. Des Weiteren soll mit der darauffolgenden benutzerorientieren Analyse eine für jeden Nutzertypen geeignete Bedienbarkeit sichergestellt werden. Hierzu zählen Parameter, wie beispielsweise eine leicht bedienbare GUI, welche zwar keinen maßgeblichen Einfluss auf die Funktionalität des Programms, dafür aber auf die allgemeine Anwendbarkeit haben.
Zu der Abbildung 3.1 ist noch anmerkend zu sagen, dass für die nun folgende Anforderungsanalyse das Modell QKD-Gerät - Anwendung genutzt worden ist, obwohl das ETSI-Protokoll 014 drei Schichten (QKD-Gerät - Schlüsselmanagementanwendung - Anwendung) vorsieht. Grund für diese Vorgehensweise war, dass durch diesen Ansatz die Anforderungen an eine Schlüsselmanagementanwendung herausgearbeitet werden sollten. Anforderungen, welche eine schlüsselnutzende Anwendung bedienen muss, wie beispielsweise Verschlüsselungsverfahren, sind nicht Teil des späteren Anforderungskataloges. Ein solche Anwendung wird allerdings während der Implementierungsphase zu Demonstrationszwecken umgesetzt werden.
[4]

3.2 Kommunikationsszenarien und Use-Cases

Alle in diesem Abschnitt beschriebenen Kommunikationsszenarien sind aus der Abbildung 3.1 hergeleitet. Diese lassen sich in zwei Oberkategorien (Szenarien mit und ohne Zwischenknoten) sowie jeweils mehrere Anwendungsfällen (1:1, 1:n und n:n) unterteilen. Die Szenarien werden sowohl beschrieben als auch die entsprechenden Use-Cases aus diesen abgeleitet. Außerdem wurde auf die Schnittstellen zwischen den Geräten nicht näher eingegangen, sondern diese als gegeben gesetzt. Der Anschaulichkeit halber werden allerdings konkrete Protokolltypen genannt, können aber aufgrund der eigentlichen Protokollunabhängigkeit jederzeit ausgetauscht werden. Ebenfalls wird die Verschlüsselungsart nicht benannt, da diese keinen direkten Einfluss auf die grundlegende Anforderungsanalyse hat.

3.2.1 Problembeschreibung

In einem allgemein gehaltenem, einfachen Kommunikationsszenario möchte ein Nutzer über eine bestimmte Anwendung mit einem anderen Nutzer (über dessen Anwendung) kommunizieren. Dies geschieht über n Zwischenknoten, wobei $n \in \mathbb{N}_{\neq}$ gilt. Eine Partei beantragt einen quantenbasierten Schlüssel über dessen QKD-Gerät und übermittelt die ID über eine unsicher/öffentliche Leitung an seinen Gegenüber, damit dieser densel-

ben Schlüssel bei seinem QKD-Gerät anfordern kann. Damit diese Anforderung funktioniert, müssen vorher die QKD-Geräte über einen QKD-Link untereinander die generierten Schlüsselmaterialien austauschen. An einer bestimmten Stelle muss sich auf ein Verschlüsselungsverfahren, bei dem der Schlüssel eingesetzt werden soll, für die darauffolgende Kommunikation geeinigt werden.

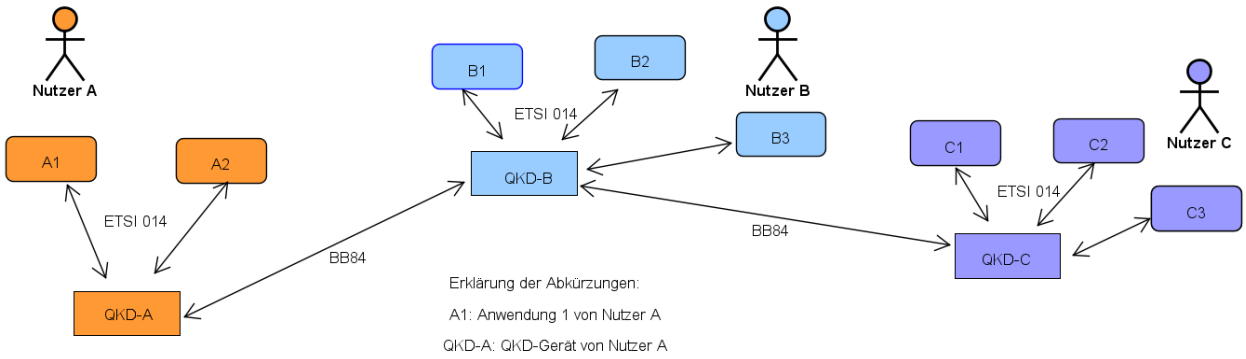


Abbildung 3.1: Darstellung eines allgemeinen Kommunikationsszenarios - Kommunikation zwischen den Anwendungen über eine unsichere/öffentliche Leitung

3.2.2 Verbindung ohne Zwischenknoten

Da es sich bei QKD-Verfahren immer um Peer-to-Peer-Architekturen handelt, kann es notwendig sein, dass zwei Netze, welche nicht miteinander verbunden sind, über ein oder mehrere Netz(e), auch als Zwischenknoten bezeichnet, hinweg kommunizieren müssen. Im Folgenden werden aber nur Szenarien betrachtet, in denen zwei Netze ohne einen oder mehreren dieser Knoten kommunizieren. Da sich die fehlerbedingten Alternativabläufe bei allen drei Szenarien gleichen, werden diese am Ende der Beschreibung aufgelistet.

Anwendungsfall 1.1: 1:1-Kommunikation

Ausgangssituation

Die Anwendung A_1 möchte mit Anwendung B_1 unter der Verwendung von Quantenschlüsseln sicher über eine unsichere/öffentliche Leitung kommunizieren.

Vorbedingung
Die unterschiedlichen Komponenten sind wie folgt miteinander verbunden:

- Die QKD-Geräte von *A* und *B* untereinander; Kommunikation über das BB84- Protokoll.
- A_1 mit QKD-*A*, B_1 mit QKD-*B*; Kommunikation jeweils über das Protokoll ETSI-014
- Die Anwendungen A_1 und B_1 über eine unsichere/öffentliche Leitung

Nachbedingung
Nach dem Kommunikationsaufbaus haben A_1 und B_1 denselben Schlüssel. Das ermöglicht es problemlos verschlüsselte Nachrichten voneinander zu entschlüsseln und weiter verarbeiten zu können. Am Ende dieses Vorganges wurde somit ein sicherer Kommunikationskanal aufgebaut und darüber Nachrichten versendet.
Außerdem ist an einer bestimmten Stelle sich auf ein Verschlüsselungsverfahren für die darauffolgende Kommunikation geeinigt worden.

Standardablauf

1. Anwendung A_1 beantragt über die entsprechende Schnittstelle einen quantenbasierten Schlüssel vom QKD-Gerät QKD-*A* und gibt die ID von B_1 an, um diese als Slave einzutragen
2. QKD-*A* nimmt die Anfrage entgegen, bearbeitet diese und liefert einen Schlüssel(K) mit einer zugehörigen ID an A_1
3. QKD-*A* erstellt über einen QKD-Link das Schlüsselmaterial zusammen mit QKD-*B*
4. Über eine unsichere/öffentliche Leitung wird der Slave-Anwendung B_1 die ID des bezogenen Schlüssels durch A_1 mitgeteilt
5. Anwendung B_1 beantragt wiederum den Schlüssel zu der übermittelten ID bei QKD-*B*
6. Die Anfrage wird zunächst geprüft, ob die Anwendung autorisiert ist, dann bearbeitet und der Schlüssel übermittelt
7. Bestätigungsnachricht an A_1, dass Kommunikation starten kann
8. Nun wird eine Nachricht von A_1 mit dem Schlüssel K durch Nutzung eines entsprechenden Verfahrens verschlüsselt und über die unsichere/öffentliche Leitung an B_1 gesendet
9. B_1 kann nun die Nachricht mit K entschlüsseln und den Nachrichteninhalt verwenden
10. Zur Bestätigung, dass der Prozess erfolgreich beendet worden ist, sendet B_1 eine Bestätigungsnachricht an A_1

Anforderungen für Anwendungsfall 1.1
Aus der Szenarienbeschreibung zum Anwendungsfall 1.1 lassen sich folgende anwendungsspezifische sowie allgemeine Anforderungen für ein späteres Schlüsselmanagementprogramm ableiten:

- Außerhalb dieses Szenarios, in welchem das Protokoll ETSI 014 vorausgesetzt wird, sind Schnittstellen durch das Nutzen von Agent Pattern als Protokoll-Übersetzer protokollunabhängig
- Authentifizierung erfolgt beim Verbindungsaufbau zwischen Anwendung und QKD-Gerät über das Schlüsselmanagementprogramm hinweg (leitet nur weiter, selbst nicht involviert)
- Anfragen von anderen Anwendungen an das Schlüsselmanagementprogramm können:
  - Angenommen
  - Überprüft, ob die Anwendung autorisiert ist (beispielsweise als Slave für den Schlüssel eingetragen)
  - Nach Dringlichkeit priorisiert
  - Bearbeitet (Schlüsselmaterial liefern)

  Werden
- Schlüssel werden bestimmten Anwendungen (min. zwei pro Schlüssel) zugewiesen
- Schlüssel können komplett (Schlüssel und ID) oder nur per ID anfragt werden
- Überprüfung ob Schlüssel schon in Verwendung bzw. verwendet worden ist (beispielsweise Schlüsselstrom bei ETSI 004)
- Eine Schlüssel-ID kann vom KMS gesendet, empfangen, gespeichert und verarbeitet/zugehöriger Schlüssel gesendet werden
- Sichere Übertragung der Schlüssel-IDs zwischen zwei Anwendungen über einen unsichere/öffentliche Leitung
- Erkennung von Fehlern und Bestimmung der Fehlerkategorie
- Senden und Empfangen von Fehlermeldungen und Bestätigungsnachrichten, mit entsprechender Reaktion
- Ständige Synchronisation zwischen Schlüsselmanagementprogrammen (KMS), damit Schlüsselmenge immer identisch ist
- Jede ID, egal ob von einem Schlüssel, einer Anwendung oder einem QKD-Gerät, ist immer eindeutig und dadurch entsprechend zuweisbar
- Jeder Schlüssel bekommt eine TTL (Time To Live), damit durch Störungen, wie einem Verbindungsabbruch, oder allgemein nicht genutzte Schlüssel unnötig Kapazitäten verbrauchen

Anwendungsfall 1.2: 1:n-Kommunikation

Ausgangssituation
Die Anwendung A_1 möchte zeitgleich mit den Anwendungen B_1 und B_3 unter der Verwendung von Quantenschlüsseln sicher über eine unsichere/öffentliche Leitung kommunizieren.

Vorbedingung
Die unterschiedlichen Komponenten sind wie folgt miteinander verbunden:

- Die QKD-Geräte von *A* und *B* untereinander; Kommunikation über das BB84- Protokoll.
- A_1 mit QKD-*A*, B_1 sowie B_3 mit QKD-*B*; Kommunikation jeweils über das Protokoll ETSI-014
- Die Anwendungen A_1, B_1 und B_3 über eine unsichere/öffentliche Leitung

Nachbedingung
Am Ende des Kommunikationsaufbaus hat A_1 mit B_1 sowie B_3 jeweils denselben Schlüssel. Somit können A_1 und B_1 sowie A_1 und B_3 problemlos verschlüsselte Nachrichten an einander senden sowie diese entschlüsseln und weiter verarbeiten. Am Ende dieses Vorganges wurde somit zwei sichere Kommunikationskanäle aufgebaut und genutzt.
Außerdem ist an einer bestimmten Stelle sich auf ein Verschlüsselungsverfahren für die darauffolgende Kommunikation geeinigt worden.

Standardablauf

1. Anwendung A_1 beantragt über die entsprechende Schnittstelle zwei quantenbasierte Schlüssel (K_1 und K_2) vom QKD-Gerät QKD-*A* und gibt die ID von B_1 und B_3 an, um diese als Slaves einzutragen
2. QKD-*A* nimmt die Anfrage entgegen, priorisiert und bearbeitet diese entsprechend, um dann die beiden Schlüssel mit der jeweils zugehörigen ID an A_1 zu liefern
3. QKD-*A* erstellt über einen QKD-Link das Schlüsselmaterial zusammen mit QKD-*B*
4. Über eine unsichere/öffentliche Leitung wird den Slave-Anwendungen(B_1 und B_3) von A_1 die ID des bezogenen Schlüsseles mitgeteilt:
   - ID-K_1 an B_1
   - ID-K_2 an B_3
5. Beide Anwendungen beantragen bei QKD-*B* den jeweiligen Schlüssel zu der übermittelten ID

6. QKD-B prüft beide Anfragen jeweils, ob B_1 bzw. B_3 autorisiert ist, priorisiert und bearbeitet diese danach entsprechend
7. Wenn B_1 und B_3 ihre Schlüssel bekommen haben, dann senden beide an A_1 eine Bestätigungsnachricht, dass die Kommunikation starten kann
8. A_1 verschlüsselt nun jeweils eine Nachricht mit dem Schlüssel K_1 und K_2 durch Nutzung eines entsprechenden Verfahrens und sendet diese über eine unsichere/öffentliche Leitung an B_1 bzw. B_3
9. B_1 kann diese Nachricht nun erfolgreich mit K_1 entschlüsseln und verarbeiten
10. B_1 sendet eine Bestätigungsnachricht an A_1, dass der Prozess erfolgreich abgeschlossen worden ist
11. Zeitgleich kann B_3 die seinerseits empfangene Nachricht mit K_2 entschlüsseln und weiterverarbeiten
12. B_3 sendet ebenfalls eine Bestätigungsnachricht an A_1, dass der Prozess erfolgreich abgeschlossen worden ist

Anforderungen für Anwendungsfall 1.2

Die anwendungsspezifischen sowie allgemeinen Anforderungen für den Anwendungsfall 1.2 decken sich in großen Teilen mit denen vom Anwendungsfall 1.1. Allerdings gibt dennoch weitere, durch die 1:n-Kommunikation bedingte, Anforderungen:

- Bei einer Schlüsselanfrage wird geprüft, ob die Menge lieferbar ist
  - Falls ja, dann wird das Schlüsselmaterial an die Anwendung geliefert
  - Falls nein, dann
    - Anfrage per Fehlermeldung ablehnen oder
    - trotzdem so schnell wie möglich geforderte Menge liefern, aber mit Verzögerung
- Verwaltung von mehreren Schlüsseln für die auswärtsgerichtete Kommunikation

Anwendungsfall 1.3: n:m-Kommunikation

Ausgangssituation
Die Anwendung A_1 möchte zeitgleich mit der Anwendung B_1 sowie B_3 unter der Verwendung von Quantenschlüsseln sicher über eine unsichere/öffentliche Leitung kommunizieren. Außerdem will B_2 zum selben Zeitpunkt mit A_1 sowie A_2 kommunizieren.

Vorbedingung
Die unterschiedlichen Komponenten sind wie folgt miteinander verbunden:

- Die QKD-Geräte von *A* und *B* untereinander; Kommunikation über das BB84- Protokoll.
- A_1 und A_2 mit QKD-*A*, B_1, B_2 und B_3 mit QKD-*B*; Kommunikation jeweils über das Protokoll ETSI-014
- Die Anwendungen A_1, A_2, B_1, B_2 und B_3 über eine unsichere/öffentliche Leitung

Nachbedingung
Am Ende des Kommunikationsaufbaus verfügen alle Parteien wie folgt über gemeinsame Schlüssel:

- A_1 teilt mit B_1 den Schlüssel K_1 und mit B_2 den Schlüssel K_2
- B_3 teilt mit A_1 den Schlüssel K_3 und mit A_2 den Schlüssel K_4

Daher können die Parteien wie beispielsweise A_1 und B_1, die nun mit K_1 über denselben Schlüssel verfügen, problemlos verschlüsselte Nachrichten an einander senden sowie diese entschlüsseln und weiter verarbeiten. Am Ende dieses Vorganges wurden somit mehrere sicherer Kommunikationskanäle aufgebaut und genutzt.
Außerdem ist an einer bestimmten Stelle sich auf ein Verschlüsselungsverfahren für die darauffolgende Kommunikation geeinigt worden.

Standardablauf
Der Standardablauf bei einer n:m-Kommunikation setzt sich aus mehreren 1:n-Szenarien zusammen und gleicht somit der Ablaufsbeschreibung des vorherigen Szenarios. Der Unterschied hierbei ist, dass dieser komplette Prozess zeitgleich von mehreren Anwendungen (beispielsweise A_1 und B_2) initiiert werden kann und diese somit parallel zueinander ablaufen. Die einzelnen Schritte sind gleich, bloß die Anwendungen, QKD-Geräte, Zwischen- und Zielknoten sind anders. Für die Veranschaulichung wird im Folgenden daher nur der Prozess aus der Sicht von B_2 dargestellt.

1. Die Anwendung B_2 beantragt über die entsprechende Schnittstelle zwei quantenbasierten Schlüssel(K_3 und K_4) vom QKD-Gerät QKD-*B* und gibt IDs von A_1 sowie A_2 als Slaves an

2. QKD-B nimmt die Anfrage entgegen, priorisiert und bearbeitet diese entsprechend, um dann die beiden Schlüssel mit der jeweils zugehörigen ID an B_2 zu liefern
3. Nun erstellt QKD-B über einen QKD-Link das Schlüsselmaterial zusammen mit QKD-A
4. Über eine unsichere/öffentliche Leitung wird den Kommunikationspartnern(A_1 und A_2,) von B_2 die ID des bezogenen Schlüsseles mitgeteilt:
   - ID-K_3 an A_1
   - ID-K_4 an A_2
5. Die Anwendungen beantragen bei ihrem QKD-Gerät den jeweiligen Schlüssel zu der übermittelten ID
6. QKD-A nimmt die Anfragen entgegen, prüft, ob die jeweilige Anwendung autorisiert ist, dann priorisieren und bearbeiten die Geräte diese entsprechend
7. A_1 und A_2 senden an B_2 jeweils eine Bestätigungsnachricht, dass die Kommunikation starten kann
8. B_2 verschlüsselt jeweils eine Nachricht mit dem Schlüssel K_3 und K_4 durch Nutzung eines entsprechenden Verfahrens und sendet diese über eine unsichere/öffentliche Leitung über den Zwischenknoten B an A_1 bzw. A_2
9. A_1 kann diese Nachricht nun erfolgreich entschlüsseln und verarbeiten
10. A_1 sendet eine Bestätigungsnachricht an B_2, dass der Prozess erfolgreich abgeschlossen worden ist
11. Zeitgleich kann A_2 die seinerseits empfangene Nachricht mit K_4 entschlüsseln und weiterverarbeiten
12. A_2 sendet ebenfalls eine Bestätigungsnachricht an B_2, dass der Prozess erfolgreich abgeschlossen worden ist

Anforderungen für Anwendungsfall 1.3
Die anwendungsspezifischen sowie allgemeinen Anforderungen für den Anwendungsfall 1.2 decken sich in großen Teilen mit denen von den Anwendungsfällen 1.1 und 1.2. Allerdings gibt dennoch weitere, durch die n:m-Kommunikation bedingte, Anforderungen:

- Verwaltung mehrerer Schlüssel für Ein- und Ausgangsnachrichten gleichzeitig, wenn beispielsweise eine Anwendung zwei Schlüssel als Master und einen als Schlüssel als Slave nutzt

Alternativabläufe für eine Kommunikation ohne Zwischenknoten

Der ETSI-Standard nach [32] liefert u.a. die folgenden Fehlermöglichkeiten für vom Standardvorgang abweichende Abläufe:

- **Fehlerhaftes (auf JSON basierendes) Anfrageformat**
  Die Partei, welche aufgrund des Fehlers, die JSON-Datei nicht lesen kann, sendet eine Fehlermeldung an den Kommunikationspartner. Darin ist u.a. enthalten, weshalb die Nachricht nicht lesbar war, wie beispielsweise die Einsetzung falscher Parameter.
- **Anwendung nicht autorisiert**
  Falls eine unautorisierte Anwendung, wie beispielsweise B_2, versuchen würde mit einer (abgefangenen) Schlüssel-ID den dazugehörigen Schlüssel anzufordern, würde B_2 eine Fehlermeldung bekommen. In dieser würde stehen, dass die Anwendung dafür nicht autorisiert ist.
- **Authentifizierung fehlgeschlagen**
  Wenn eine Anwendung oder ein QKD-Gerät nicht per Zertifikat authentifizierbar ist, dann soll kein Verbindungsaufbau initiiert oder eine entsprechende Anfrage hierzu angenommen werden.
- **Serverseitiger Fehler**
  Dieser Fehler tritt auf, wenn der Server, in diesem Fall das QKD-Gerät, überlastet ist, weil es zu viele Anfragen bekommt. In diesem Fall werden alle weiteren Anfragen abgelehnt und eine Fehlermeldung zurückgesendet, dass es zu einer Überlastung gekommen ist. Dieser Vorgang wird so lange wiederholt bis die restlichen Anfragen in einem ausreichendem Maß abgearbeitet worden sind.
- **Nicht genügend Schlüssel verfügbar**
  Falls nicht genügend Schlüssel verfügbar sind, werden die Anfragen entsprechend dem Zeitpunkt ihrer Anfrage in eine Warteschlange einsortiert. Sobald für eine Anforderung genug Schlüssel vorhanden sind, wird diese abgearbeitet und aus der Liste entfernt. Dies geschieht so lange bis alle Anfragen abgearbeitet sind. Zweite Lösung wäre die Anfragen so lange abzulehnen bis genug Schlüsselmaterial vorhanden ist.
- **Verbindungaufbau fehlgeschlagen**
  Wenn keine Verbindung zu einer anderen Komponente, QKD-Gerät oder Anwendung, hergestellt werden kann, dann soll eine vordefinierte Zeitspanne gewartet und dann wieder versucht werden die Verbindung aufzubauen. Ansonsten soll es zu einem Abbruch mit einer Fehlermeldung, in der die Ereignisse benannt werden, an den Nutzer hinter der Anwendung kommen.
- **Verschlüsselte Nachricht nicht zu entschlüsseln**
  Wenn für eine Anwendung, beispielsweise B_1, eine Nachricht nicht zu entschlüsseln

sein sollte, soll der Schlüssel zur vom Sender, beispielsweise A_1, übermittelten ID nochmals angefordert werden. Wenn der Fehler nochmals auftritt sollen die QKD-Geräte, vom Empfänger-QKD-Gerät (QKD-*B*) initiiert, ihre Schlüsselsätze abgleichen. Falls der Fehler immer noch nicht behoben ist, soll die Schlüssel-ID vom Sender angefordert. Als letzte Option muss der komplette Vorgang wiederholt werden.

- **Timeout bei einer Komponente**
  Wenn eine Komponente zu lange zum Antworten oder das Senden einer Bestätigungsnachricht ausbleibt, soll eine vordefinierte Zeitspanne gewartet werden und die Nachricht nochmals gesendet werden. Falls derselbe Fehler wieder auftritt, soll der komplette Kommunikationsvorgang abgebrochen werden, eine vordefinierte Zeitspanne gewartet und erneut gestartet werden. Bei einem Abbruch soll eine Fehlermeldung mit der Benennung des Ereignisses an den Nutzer hinter der Anwendung angezeigt werden.

Zusätzlich könnte es noch zu weiteren Fehlern kommen, welche hier noch nicht aufgelistet sind, aber erst bei der Testung eines Prototypens auffallen. So wäre es möglich, dass beispielsweise eine Schlüssel-ID fehlerhaft übermittelt wird und somit kein entsprechender Schlüssel vorhanden ist.

3.2.3 Verbindung mit Zwischenknoten

Da es sich bei QKD-Verfahren immer um Peer-to-Peer-Architekturen handelt, kann es notwendig sein, dass zwei Netze, welche nicht miteinander verbunden sind, über ein oder mehrere Netz(e), auch als Zwischenknoten bezeichnet, hinweg kommunizieren müssen. Im Folgenden werden nur Szenarien zwischen zwei durch n Zwischenknoten verbundenen Netzen beschrieben, wobei grundsätzlich $n \epsilon \mathbb{N}$ gilt. Zur Nachvollziehbar- und Übersichtlichkeit wird es in diesen Beispielen allerdings jeweils nur einen Zwischenknoten geben. Außerdem werden nach der Szenarienbeschreibung die fehlerbedingten Alternativabläufe aufgelistet, da diese sich bei allen dreien gleichen.

Anwendungsfall 2.1: 1:1-Kommunikation

Ausgangssituation
Die Anwendung A_1 möchte mit Anwendung C_1 unter der Verwendung von Quantenschlüsseln sicher über eine unsichere/öffentliche Leitung kommunizieren.

Vorbedingung
Die unterschiedlichen Komponenten sind wie folgt miteinander verbunden:

- Die QKD-Geräte von A und C über QKD-B; Kommunikation über das BB84- Protokoll.
- A_1 mit QKD-A, B_1 mit QKD-B, C_1 mit QKD-C; Kommunikation jeweils über das Protokoll ETSI-014
- Die Anwendungen A_1 und C_1 mit einem Zwischenknoten, in diesem Fall B, über eine unsichere/öffentliche Leitung

Nachbedingung
Nach dem Kommunikationsaufbaus haben A_1 und C_1 denselben Schlüssel. Das ermöglicht es problemlos verschlüsselte Nachrichten voneinander zu entschlüsseln und weiter verarbeiten zu können. Am Ende dieses Vorganges wurde somit ein sicherer Kommunikationskanal über den Zwischenknoten B aufgebaut und darüber Nachrichten versendet.
Außerdem ist an einer bestimmten Stelle sich auf ein Verschlüsselungsverfahren für die darauffolgende Kommunikation geeinigt worden.

Standardablauf

1. Anwendung A_1 beantragt über die entsprechende Schnittstelle die quantenbasierten Schlüssel K_1 vom QKD-Gerät QKD-*A*. Die ID von C_1 wird als Slave angegeben.
2. QKD-*A* nimmt die Anfrage entgegen, bearbeitet diese und liefert das Schlüsselmaterial mit der zugehörigen ID an A_1
3. Vorab hat QKD-*A* zusammen mit QKD-*C* über einen QKD-Link und über den Zwischenknoten *B* hinweg das Schlüsselmaterial erzeugt
4. Über eine unsichere/öffentliche Leitung wird die ID der zur Verwendung bestimmten Schlüssel an *B* gesendet, um von dort an den Zielknoten *C* weitergeleitet zu werden
5. Anwendung C_1 erhält die Schlüssel-ID und beantragt nun den dazugehörigen Schlüssel K_1 bei QKD-*C*
6. Die Anfrage wird, ob C_1 autorisiert ist, dann bearbeitet und der Schlüssel an C_1 gesendet
7. C_1 sendet eine Bestätigungsnachricht an A_1, dass die Kommunikation starten kann
8. A_1 kann nun die Nachricht verschlüsseln und durch einen sicheren Kommunikationskanal über *B* an C_1 senden
9. der Zwischenknoten *B* sendet an A_1 eine Bestätigung, dass die Nachricht erfolgreich an C_1 weitergeleitet worden ist
10. C_1 kann diese Nachricht nun erfolgreich entschlüsseln und verarbeiten
11. C_1 sendet eine Bestätigungsnachricht an A_1, dass der Prozess erfolgreich abgeschlossen worden ist

Anforderungen für den Anwendungsfall 2.1
Die anwendungsspezifischen sowie allgemeinen Anforderungen für den Anwendungsfall 2.1 decken sich in großen Teilen mit denen vom Anwendungsfall 1.1. Allerdings gibt dennoch weitere, durch die 1:1-Kommunikation mit Zwischenknoten bedingte, Anforderungen:

- Auffindung eines QKD-Endpunktes und der Anwendung, mit welcher kommuniziert werden soll, wenn beispielsweise der Standort nicht bekannt ist
  - Mögliche Umsetzung: klassisches Routing-Schema oder zentrale Managementeinheit
- Authentifizierung zwischen Knoten für eine sichere Weiterleitung der Nachrichten (Trusted Nodes)

Anwendungsfall 2.2: 1:n-Kommunikation

Ausgangssituation
Die Anwendung A_1 möchte mit Anwendung C_1 sowie C_2 unter der Verwendung von Quantenschlüsseln sicher über eine unsichere/öffentliche Leitung kommunizieren.

Vorbedingung
Die unterschiedlichen Komponenten sind wie folgt miteinander verbunden:

- Die QKD-Geräte von *A* und *C* über QKD-*B*; Kommunikation über das BB84- Protokoll.
- A_1 mit QKD-*A*, C_1 sowie C_2 mit QKD-*C*; Kommunikation jeweils über das Protokoll ETSI-014
- Die Anwendungen A_1, C_1 und C_2 mit einem Zwischenknoten, in diesem Fall *B*, über eine unsichere/öffentliche Leitung

Nachbedingung
Am Ende des Kommunikationsaufbaus hat A_1 mit C_1 sowie C_2 jeweils denselben Schlüssel (K_1 und K_2). Außerdem ist an einer bestimmten Stelle sich auf ein Verschlüsselungsverfahren für die darauffolgende Kommunikation geeinigt worden

Standardablauf

1. Anwendung A_1 beantragt über die entsprechende Schnittstelle die quantenbasierten Schlüssel K_1 und K_2 vom QKD-Gerät QKD-*A*. Die IDs von C_1 und C_2 werden als Slaves angegeben.
2. QKD-*A* nimmt die Anfrage entgegen, bearbeitet diese und liefert das Schlüsselmaterial mit den zugehörigen IDs an A_1
3. Vorab hat QKD-*A* zusammen mit QKD-*C* über einen QKD-Link und über den Zwischenknoten *B* hinweg das Schlüsselmaterial erzeugt
4. Über eine unsichere/öffentliche Leitung werden die IDs der zur Verwendung bestimmten Schlüssel an *B* gesendet und von dort an den Zielknoten *C* weitergeleitet
5. Anwendungen C_1 und C_2 erhalten jeweils die für sie bestimmte Schlüssel-ID und beantragen den dazugehörigen Schlüssel K_1 bzw. K_2 bei QKD-*C*
6. Die Anfragen werden geprüft, ob C_1 bzw. C_2 autorisiert ist, dann bearbeitet und die Schlüssel an C_1 und/oder C_2 gesendet
7. C_1 und C_2 senden jeweils eine Bestätigungsnachricht an A_1, dass die Kommunikation starten kann

8. A_1 kann nun die Nachrichten verschlüsseln und durch einen sicheren Kommunikationskanal über B an C_1 und C_2 senden
9. Der Zwischenknoten B sendet an A_1 eine Bestätigung, dass die Nachrichten erfolgreich an C_1 und/oder C_2 gesendet worden sind
10. C_1 kann diese Nachricht nun erfolgreich entschlüsseln und verarbeiten
11. C_1 sendet eine Bestätigungsnachricht an A_1, dass der Prozess erfolgreich abgeschlossen worden ist
12. Zeitgleich kann C_2 die seinerseits empfangene Nachricht mit K_2 entschlüsseln und weiterverarbeiten
13. C_2 sendet ebenfalls eine Bestätigungsnachricht an A_1, dass der Prozess erfolgreich abgeschlossen worden ist

Anforderungen für den Anwendungsfall 2.2
Die anwendungsspezifischen sowie allgemeinen Anforderungen für 2.2 decken sich mit denen von den Anwendungsfällen 2.1 und 1.2. Die benötigten Anforderungen bei einer Kommunikation mit Zwischenknoten sind beim Anwendungsfall 2.1 aufgelistet. Ansonsten handelt es sich um dasselbe Szenario wie in 1.2 beschrieben.

Anwendungsfall 2.3: n:m-Kommunikation

Ausgangssituation
Die Anwendung A_1 möchte zeitgleich mit der Anwendung C_1 sowie C_2 unter der Verwendung von Quantenschlüsseln sicher über eine unsichere/öffentliche Leitung kommunizieren. Außerdem will C_3 zum selben Zeitpunkt mit A_1 sowie A_2 kommunizieren.

Vorbedingung
Die unterschiedlichen Komponenten sind wie folgt miteinander verbunden:

- Die QKD-Geräte von A und C über QKD-B; Kommunikation über das BB84-Protokoll
- A_1 und A_2 mit QKD-A, C_1, C_2 und C_3 mit QKD-C; Kommunikation jeweils über das Protokoll ETSI-014
- Die Anwendungen A_1, A_2, C_1, C_2 und C_3 mit einem Zwischenknoten, in diesem Fall B, über eine unsichere/öffentliche Leitung

Nachbedingung
Am Ende des Kommunikationsaufbaus verfügen alle Parteien wie folgt über gemeinsame Schlüssel:

- A_1 teilt mit C_1 den Schlüssel K_1 und mit C_2 den Schlüssel K_2
- C_3 teilt mit A_1 den Schlüssel K_3 und mit A_2 den Schlüssel K_4

Außerdem ist an einer bestimmten Stelle sich auf ein Verschlüsselungsverfahren für die darauffolgende Kommunikation geeinigt worden.

Standardablauf
Der Standardablauf bei einer n:m-Kommunikation setzt sich aus mehreren 1:n-Szenarien zusammen und gleicht somit der Ablaufsbeschreibung des vorherigen Szenarios. Der Unterschied hierbei ist, dass dieser komplette Prozess zeitgleich von mehreren Anwendungen (beispielsweise A_1 und C_3) initiiert werden kann und diese somit parallel zueinander ablaufen. Die einzelnen Schritte sind gleich, bloß die Anwendungen, QKD-Geräte, Zwischen- und Zielknoten sind anders. Für die Veranschaulichung wird im Folgenden daher nur der Prozess aus der Sicht von C_3 dargestellt.

1. Anwendung C_3 beantragt über die entsprechende Schnittstelle die quantenbasierten Schlüssel K_3 und K_4 vom QKD-Gerät QKD-C. Die IDs von A_1 und A_2 werden als Slaves angegeben.
2. QKD-C nimmt die Anfrage entgegen, bearbeitet diese und liefert das Schlüsselmaterial mit den zugehörigen IDs an C_3

3. Vorab hat QKD-*C* zusammen mit QKD-*A* über einen QKD-Link und über den Zwischenknoten *B* hinweg das Schlüsselmaterial erzeugt
4. Über eine unsichere/öffentliche Leitung werden die IDs der zu verwendenen Schlüssel an *B* gesendet und von dort an den Zielknoten *A* weitergeleitet
5. Die Anwendungen A_1 und A_2 erhalten jeweils die für sie bestimmte Schlüssel-ID und beantragen den dazugehörigen Schlüssel K_3 bzw. K_4 bei QKD-*A*
6. Die Anfragen werden von QKD-*A* geprüft, ob die beantragende Anwendung autorisiert ist, dann bearbeitet und die Schlüssel an A_1 und/oder A_2 gesendet
7. A_1 und A_2 senden jeweils eine Bestätigungsnachricht an C_3, dass die Kommunikation starten kann
8. C_3 kann nun die Nachrichten verschlüsseln und durch einen sicheren Kommunikationskanal über *B* an A_1 und A_2 senden
9. Der Zwischenknoten *B* sendet an C_3 eine Bestätigung, dass die Nachrichten erfolgreich an A_1 und/oder A_2 gesendet worden sind
10. A_1 kann diese Nachricht nun erfolgreich mit K_3 entschlüsseln und verarbeiten
11. A_1 sendet eine Bestätigungsnachricht an C_3, dass der Prozess erfolgreich abgeschlossen worden ist
12. Zeitgleich kann A_2 die seinerseits empfangene Nachricht mit K_4 entschlüsseln und weiterverarbeiten
13. A_2 sendet ebenfalls eine Bestätigungsnachricht an C_3, dass der Prozess erfolgreich abgeschlossen worden ist

Anforderungen für den Anwendungsfall 2.3

Die anwendungsspezifischen sowie allgemeinen Anforderungen für 2.3 decken sich mit denen von den Anwendungsfällen 2.1 und 1.3. Die benötigten Anforderungen bei einer Kommunikation mit Zwischenknoten sind beim Anwendungsfall 2.1 aufgelistet. Ansonsten handelt es sich um dasselbe n:m-Szenario wie in 1.3 beschrieben.

Alternativabläufe für eine Kommunikation ohne zwischen Knoten

Die fehlerbedingten Alternativabläufe bei einer Kommunikation mit Zwischenknoten gleichen denen des anderen Szenariotyps. Allerdings kann es zu weiteren durch Fehler hervorgerufene Abläufe kommen:

- **Authentifizierung fehlgeschlagen**
  Hierbei soll der Fokus besonders auf die Authentifizierung der Knoten liegen, damit eine sichere Route zwischen zwei Endpunkten gewährleistet werden kann. Falls sich ein Knoten nicht bei Aufforderung authentifizieren kann, dann kommt es zu einer Fehlermeldung. Entweder kann dann eine Route über andere Knoten gefunden werden oder der komplette Prozess abgebrochen werden und es kommt zu einer weiteren Fehlermeldung.
- **Verbindung zum Knoten unter- oder abgebrochen**
  Wenn die Verbindung zu einem Knoten unter- oder abgebrochen ist, bemerkbar durch das Ausbleiben von Bestätigungsnachrichten, soll eine vordefinierte Zeitspanne gewartet werden. Danach wird der vorherige Prozess, beispielsweise das Übermitteln von Schlüsselmaterial an das QKD-Gerät des Zielknoten, wiederholt. Ist die Verbindung immer noch abgebrochen wird wieder die vordefinierte Zeitspanne gewartet, das bisherige Schlüsselmaterial verworfen und der komplette Prozess neu gestartet. Wenn der Knoten immer noch nicht erreichbar ist, dann soll eine neue Route zum Zielendpunkt gefunden werden. Ansonsten wird eine Fehlermeldung gegeben, dass eine Kommunikation mit dem gewünschten Partner nicht möglich sei.

Zusätzlich könnte es noch zu weiteren Fehlern kommen, welche hier noch nicht aufgelistet sind, aber erst bei der Testung eines Prototypens auffallen. So wäre es möglich, dass beispielsweise eine Schlüssel-ID fehlerhaft übermittelt wird und somit kein entsprechender Schlüssel vorhanden ist.

3.3 Anforderungskatalog für das Schlüsselmanagement-programm

Anhand des folgenden Anforderungskatalogs soll eine komplette Übersicht sowohl über die Anforderungen aus allen Szenarien sowie aus der Benutzersicht ermöglicht werden. Hier raus wird im nächsten Kapitel der notwendige Programmentwurf abgeleitet und somit diese Ergebnisse zur Orientierung bei der Erstellung der benötigten Struktur, bestehend u.a. aus Klassen und den dazugehörigen Methoden, genutzt.
Dazu werden im Folgenden zunächst alle, während der Szenarien- und Use-Cases- Beschreibung erarbeitet Anforderungen an ein Schlüsselmanagementprogramm (KMS) im Rahmen eines Quantenkommunikationsnetzes einheitlich aufgelistet. Da es sich hierbei um funktionale Anforderungen handelt, werden diese mit einem F sowie einer Zahl zur eindeutigen Zuweisbarkeit gekennzeichnet.
Bei der benutzerorientierten Anforderungsanalyse handelt es sich nicht um eine zwingend notwendige Maßnahme und somit sind diese nicht-funktionale Anforderungen (mit NF gekennzeichnet). Dennoch sind diese Aspekte vor allem für die Handhabbarkeit aus der Sicht eines Benutzers, welcher nur ein entsprechendes Programm nutzen möchte, wichtig. Deshalb sollten diese ebenfalls im Laufe der Umsetzung eines vollumfänglichen Programmes beachtet werden, da ansonst eine Anwendung innerhalb eines Kommunikationsnetzes in der Realität schwer möglich wäre.

Tabelle 3.1: Anforderungskatalog

Anforderung	**Nr.**	**Kurzbeschreibung**	**Kommentar**
Protokollunabhängige Schnittstellen	F_1	Alle Schnittstellen des Schlüsselmanagement-programms (KMS) sind protokollunabhängig	Schnittstellen sind durch nutzen von Agent Pattern als Protokoll-Übersetzer protokollunabhängig. Damit sind folgende Schnittstellen gemeint: • Anwendung - KMS • QKD-Gerät - KMS • KMS- KMS

Authentifizierung	F_2	Authentifizierung zwischen einer Anwendung und einem QKD-Gerät	Authentifizierung erfolgt beim Verbindungsaufbau zwischen Anwendung und QKD-Gerät über das Schlüsselmanagement-programm hinweg (leitet nur weiter, selber nicht involviert)
	F_3	Authentifizierung der Knoten	Authentifizierung zwischen Knoten für eine sichere Weiterleitung der Nachrichten (Trusted Nodes)
Eindeutige Identifikation	F_4	Jede Kommunikationskomponente besitzt eine Identifikationsnummern	Jede ID, egal ob von einem Schlüssel, einer Anwendung oder einem QKD-Gerät, ist immer eindeutig und dadurch entsprechend zuweisbar
Suchfunktion für Endpunkte	F_5	Jeder Endpunkt soll auffindbar sein	Auffindung eines QKD-Endpunktes und der Anwendung, mit welcher kommuniziert werden soll, wenn beispielsweise der Standort nicht bekannt ist. Mögliche Umsetzung: klassisches Routing-Schema oder zentrale Managementeinheit

Anfragenmanagement	F_6	Umgang mit Anfragen	Anfragen von anderen Anwendungen an das Schlüsselmanagement-programm können: • Angenommen • Überprüft, ob die Anwendung autorisiert ist (beispielsweise als Slave für den Schlüssel eingetragen) • Nach Dringlich-keit priorisiert • Bearbeitet (Schlüsselma-terial liefern) Werden
Umgang mit Schlüssel-material	F_7	Schlüsselanforderung	Schlüssel können kom-plett (Schlüssel und ID) oder nur per ID bezogen werden
	F_8	Schlüsselzuweisung	Schlüssel werden be-stimmten Anwendungen (min. zwei pro Schlüs-sel) zugewiesen
	F_9	Statusprüfung Schlüssel	Überprüfung ob Schlüs-sel schon in Verwen-dung bzw. verwendet worden ist (beispiels-weise Schlüsselstrom bei ETSI 004)

Umgang mit Schlüsselmaterial	F_{10}	Bedienung Schlüsselanfrage	Bei einer Schlüsselanfrage wird geprüft, ob die Menge lieferbar ist • Falls ja, dann wird das Schlüsselmaterial an die Anwendung geliefert • Falls nein, dann – Anfrage per Fehlermeldung ablehnen oder – trotzdem so schnell wie möglich geforderte Menge liefern, aber mit Verzögerung
	F_{11}	Schlüsselverwaltung	Verwaltung mehrerer Schlüssel für Ein- und Ausgangsnachrichten gleichzeitig, wenn beispielsweise eine Anwendung zwei Schlüssel als Master und einen als Slave nutzt
	F_{12}	Umgang eines KMS mit Schlüssel-ID	Eine Schlüssel-ID kann gesendet, empfangen, gespeichert und verarbeitet/zugehöriger Schlüssel gesendet werden

Umgang mit Schlüsselmaterial	F_{13}	Übertragung Schlüssel-IDs	Sichere Übertragung der Schlüssel-IDs zwischen zwei Anwendungen über eine unsichere/öffentliche Leitung
	F_{14}	Begrenzte Nutzungszeit von Schlüsselmaterial	Jeder Schlüssel bekommt eine TTL (Time To Live), damit durch Störungen, wie einem Verbindungsabbruch, oder allgemein nicht genutzte Schlüssel unnötig Kapazitäten verbrauchen
Fortlaufende Synchronisation	F_{15}	Fortlaufende Synchronisation zwischen Schlüsselmanagementprogrammen (KMS)	Ständige Synchronisation zwischen KMS der kommunizierenden Parteien, damit die Schlüsselmenge immer identisch sind
Protokollbasiertes Fehlermanagement	F_{16}	Fehlermeldungen und Bestätigungsnachrichten	Senden und Empfangen von Fehlermeldungen und Bestätigungsnachrichten, mitentsprechender Reaktion
	F_{17}	Fehlererkennung	Erkennung von Fehlern und Bestimmung der Fehlerkategorie

Optisches	NF_1	Angepasste Benutzeroberfläche	Übersichtliche Benutzeroberfläche durch: • Klare Strukturierung • Keine Überladung mit Inhalten • Farben nur nutzen, um Wichtiges hervorzuheben und somit Anwendung sinnvoll
	NF_2	Benutzerfreundliche Sprache	Verständliche Sprache/fachspezifische Ausdrücke vermeiden, um jeden Benutzertypen zu erreichen
Bedienung des Programmes	NF_3	Bedienbarkeit	Leichte Bedienbarkeit soll gegeben sein, beispielsweise durch standardisierte Voreinstellungen
	NF_4	Individuelle Einstellungen	Trotzdem individuelle Einstellungen, wie beispielsweise Auswahl des Verschlüsselungsverfahren oder Schlüssellänge, möglich
Hilfestellungen	NF_5	Jederzeit sind Hilfestellungen für den Benutzer wahrnehmbar	Hilfestellungen zur Bedienbarkeit anbieten und gleichzeitig möglichst Benutzeroberfläche möglichst selbsterklärend gestalten

Kapitel 4

Programmentwurf

Nachdem im vorangegangenen Kapitel eine Anforderungsanalyse anhand unterschiedlicher Szenarienbeschreibungen durchgeführt worden ist, soll in den folgenden Abschnitten ein Programmentwurf für die Umsetzung eines Schlüsselmanagementprogramms im Rahmen einen Quantenkommunikationsnetzes erarbeitet werden. Zunächst wird der geplante Umfang eines solchen Programmes definiert, um daraufhin ein theoretisches Konstrukt planen zu können. Dazu wird der vorher angefertigte Anforderungskatalog, genauer gesagt die funktionalen Anforderungen, genutzt, um die benötigten Komponenten ableiten zu können. Anhand dessen werden danach sowohl die Klassendiagramme und deren Beziehungen zueinander sowie Interaktionsdiagramme entworfen. Am Ende steht somit ein fertiger Programmentwurf für ein Schlüsselmanagementprogramm im Rahmen einen Quantenkommunikationsnetzes. Im nächsten Kapitel folgt die Dokumentation des angefertigten Prototypen, welcher zeigen soll, dass sich der theoretische Entwurf auch in der Praxis umsetzen lassen würde.

4.1 Geplanter Programmumfang

Die Funktionalitäten, welche ein vollumfängliches Programm beinhalten muss und somit den Programmumfang definieren, lassen sich anhand des in Kapitel 3 erarbeiteten Anforderungskatalogs ableiten.
Für die Randbedingungen gelten, dass alle definierten Schnittstellen protokollunabhängig sein sollen und jeder Teil einer Kommunikationseinheit eine eindeutige Identifikationsnummer besitzen soll.
Der eigentliche Umfang definiert sich anhand der aktiven Funktionalitäten. So soll das Programm in der Lage sein vor einem Verbindungsaufbau zwischen einer Anwendung und einem QKD-Gerät sowie zwischen Knoten eine Authentifizierung mit Zertifikaten

durchführen zu können. Außerdem soll es möglich sein, dass jeder Endpunkt mit der entsprechenden Anwendung, für die eine Nachricht bestimmt ist, sowie eine passende, sichere Route gefunden werden kann, wenn der Endpunkt und/oder die Route unbekannt ist. Ebenfalls sollen die unterschiedlichen Anfragen, wie beispielsweise für die Lieferung von Schlüsselmaterial, zentral verwaltet und standardisiert abgearbeitet werden. Dies geschieht durch das Annehmen, überprüfen, ob Anfragender autorisiert ist, Priorisieren sowie schlussendlich dem Liefern des Schlüsselmaterials. Die wichtigste Aufgabe besteht in dem Managen der Schlüsselmaterialien. Hierzu soll es unteranderem möglich sein Schlüsselmaterial komplett (Schlüssel und ID) oder per ID anzufordern. Weitere Funktionen sind im Anforderungskatalog von F_7 bis F_{13} nachzulesen. Des Weiteren soll ein fortlaufende Synchronisation zwischen den QKD-Geräten gewährleistet werden. Als Letztes ist der Fehlermanager zu nennen, welche nicht nur Fehler erkennen, melden und Gegenmaßnahmen einleiten soll sondern auch Bestätigungsnachrichten an die entsprechenden Stellen senden soll, wenn ein Vorgang erfolgreich abgeschlossen worden ist.

4.2 Benötigte Komponenten

Somit setzt sich das spätere Programm aus sieben unterschiedlichen Komponenten zusammen, wobei sechs von sieben jeweils die funktionalen Anforderungen erfüllen und der Einfachheitshalber nach diesen benannt sind. Das protokollbasierte Fehlermanagement wird nicht als selbstständige Komponente abgebildet, sondern wird von jeder Komponente für ihren jeweiligen Bereich in Eigenarbeit vorgenommen.
Die Kommunikation sowie Zusammenarbeit dieser Komponenten untereinander sowie mit anderen Anwendungen ist beispielhaft und vereinfacht auf Abbildung 4.1 anhand eines Komponentendiagramms zu sehen. Da 4.1 nur zum Verständnis dient, ist die Syntax nicht vollkommen korrekt. Der Fokus hierbei liegt nicht auf der Funktionsweise, sondern nur auf der Kommunikation durch angebotene sowie benötigte Schnittstellen innerhalb sowie außerhalb der Anwendung. Zum besseren Verständnis dient folgende Legende:

- $-\circ$ angebotene Schnittstelle: stellt einen Dienst bereit
- $\supset$ benötigte Schnittstelle: benötigt einen Dienst
- $\square$ Port: delegiert die Schnittstellen an eine interne Klasse/Komponente
- $-$ Abhängigkeit: ähnliche wie Schnittstellendarstellung/Beziehung zueinander

Die Funktionsweise der einzelnen Komponenten wird im folgenden Abschnitt kurz erklärt. Hieraus werden im restlichen Kapitel, die für die Umsetzung des Prototypen bzw. eines vollständigen Programmes benötigten Klassen- sowie Interaktionsdiagramme erstellt.

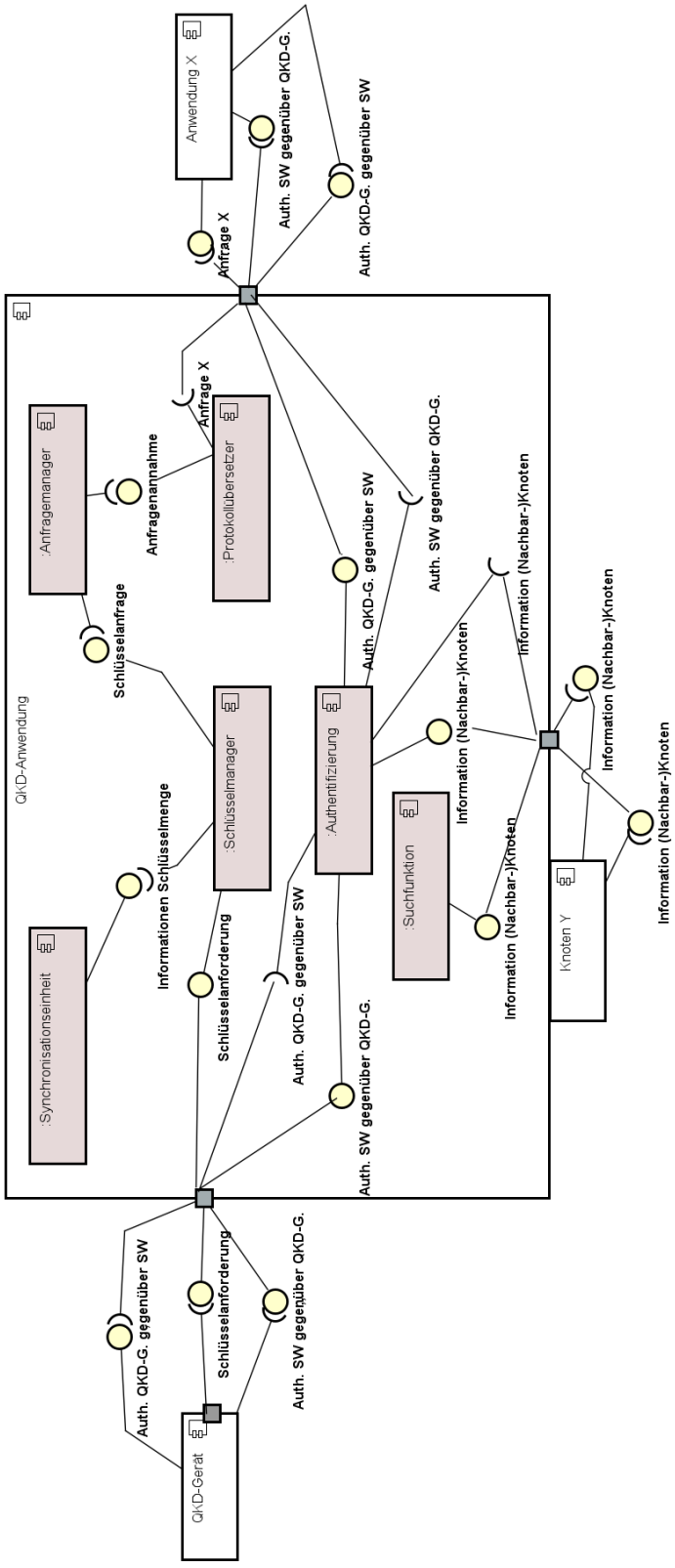


Abbildung 4.1: Vereinfachtes Komponentendiagramm zur Erläuterung des Zusammenspiels der einzelnen Komponenten

4.2.1 Protokollübersetzer

Durch die Verwendung von Agent Pattern soll eine Protokollunabhängigkeit im Bezug zu den unterschiedlichen Protokollen, welche von den anderen Anwendungen sowie QKD-Geräten verwendet werden, geschaffen werden. Diese Komponente soll alle empfangenen Nachrichtenformate (Anfragen etc.) in ein für das Schlüsselmanagementprogramm lesbares Format bringen/übersetzen. Auch zu sendenden Daten sollen in eine für den Empfänger lesbares Format zurück- bzw. umwandeln.

4.2.2 Authentifizierungsstelle

Diese Komponente ist dafür zuständig den Datenverkehr für den Authentifizierungsablauf zwischen einer Anwendung und einem QKD-Gerät weiterzuleiten, ohne dabei involviert zu sein. Authentifizierungsstelle selbst muss sich gegenüber einer Anwendung oder QKD-Gerät nicht verifizieren, da diese als sicher und vertrauenswürdig angesehen wird.
Außerdem soll über eine entsprechende Schnittstelle eine Authentifizierung zwischen Knoten durch die jeweiligen KMS möglich sein. Ziel dieses Vorganges ist die Schaffung eines Netzes von Trusted Nodes für eine sichere Weiterleitung der Nachrichten. Als sicher vermerkte Knoten werden in einer Tabelle für die Dauer der Kommunikation zwischengespeichert und danach wieder gelöscht, weil im Zeitraum zwischen zwei Kommunikationsabläufen ein Knoten den Status als Trusted Node verlieren könnte.

4.2.3 Suchfunktion

Das Schlüsselmanagementprogramm (KMS) ist mit dieser Komponente in der Lage einen QKD-Endpunkt mit der entsprechenden Anwendung, mit welcher kommuniziert werden soll, zu finden, wenn beispielsweise der Standort nicht bekannt ist. Dies soll durch ein statisches Routing-Verfahren, also in Zusammenarbeit mit anderen KMS, geschehen. Dabei wird eine Route für die Kommunikation gefunden, festgelegt und danach für den gesamten Zeitraum der Interaktion genutzt. Hierzu wird dieser festgelegte Weg von der Komponente in einer Routingtabelle für den Zeitraum der Kommunikation hinterlegt, damit nicht für jede Nachricht einzeln erneut eine Route gefunden werden muss, und danach wieder gelöscht.

4.2.4 Anfragemanager

An den Anfragenmanager werden die von außen über eine Schittstelle delegierten Anfragen von anderen Anwendungen bearbeitet. Diese können angenommen und überprüft

werden, ob die Anwendung autorisiert ist (beispielsweise als Slave für den Schlüssel eingetragen ist). Außerdem übernimmt diese Komponente das Priorisieren der Anfragen nach Dringlichkeit. Schlüsselanfragen werden an den Schlüsselmanager weitergeleitet, welcher nach der Bearbeitung das Schlüsselmaterial wieder zurück an den Anfragenmanager schickt. Dieser liefert dann das Schlüsselmaterial an die anfragende Anwendung.

4.2.5 Schlüsselmanager

Der Schlüsselmanager steht über eine direkte Schnittstelle mit dem QKD-Gerät, welches das Schlüsselmaterial erstellt, in Verbindung. Über diese Komponente können Schlüssel komplett (Schlüssel und ID) oder nur per ID über den Anfragenmanager, welcher die entsprechende Anfrage von einer Anwendung weiterleitet, bezogen werden. In eine Liste werden alle relevanten Informationen zu einem Schlüssel gespeichert. Dies wären:

- Schlüssel-ID und eigentliches Schlüsselmaterial
- Schlüsselmaterial schon in Verwendung
- Time to Live, welche bei Ablauf für das Löschen alle Informationen/dieses Eintrages sorgt
- Schlüsselnutzer, allgemein sowie welche zugewiesene Anwendung ist Master und welcher Slave

Falls die Kommunikation beendet worden ist und noch ein Rest an Nutzungszeit übrig sein sollte, dann wird der Schlüssel mit allen dazugehörigen Informationen gelöscht. Somit kann eine Schlüssel-ID von dieser Komponente an den Kommunikationspartner über eine unsichere/öffentliche Leitung gesendet werden. Falls der Schlüsselmanager nur eine Schlüssel-ID empfängt, wird ein neue Listeneintrag vorgenommen, der zugehörige Schlüssel über die QKD-Geräte-Schnittstelle bezogen und der Eintrag vervollständigt. Des Weiteren wird die sichere Übertragung der Schlüssel-IDs zwischen zwei Anwendungen über eine unsichere/öffentliche Leitung gewährleistet, wenn beispielsweise eine ID zu einer Slave-Anwendung gesendet werden soll, damit diese den entsprechenden Schlüssel beziehen kann. Außerdem soll diese Komponente in der Lage sein zu prüfen, ob eine angefragte Schlüsselmenge lieferbar ist oder nicht. Je nachdem wie die Anfrage priorisiert worden ist, wird diese entweder abgelehnt, wenn nicht genug Schlüssel in ausreichender Menge vorhanden sind, oder so schnell wie möglich die geforderte Menge geliefert, wenn auch mit Verzögerung.

4.2.6 Synchronisationseinheit

Die Synchronisationseinheiten der miteinander kommunizierenden Schlüsselmanagementprogrammen stehen über die dafür vorgesehene Schnittstelle in einem ständigen Austausch

damit die Schlüsselmengen immer identisch sind. Vor jedem Verbindungsaufbau zwischen zwei Anwendungen gleichen die jeweiligen KMS die Schlüsselmenge ab und ergänzen diese wenn notwendig. Falls während einer Kommunikation neues Schlüsselmaterial benötigt und erzeugt werden sollte, dann soll es zu der Synchronisation zwischen den KMS mit dem oben angesprochenen Ziel kommen. Hierfür wird eine Schnittstelle zwischen der Synchronisationseinheit und dem Schlüsselmanager genutzt, um die benötigten Informationen über die Schlüsselmenge beziehen zu können.

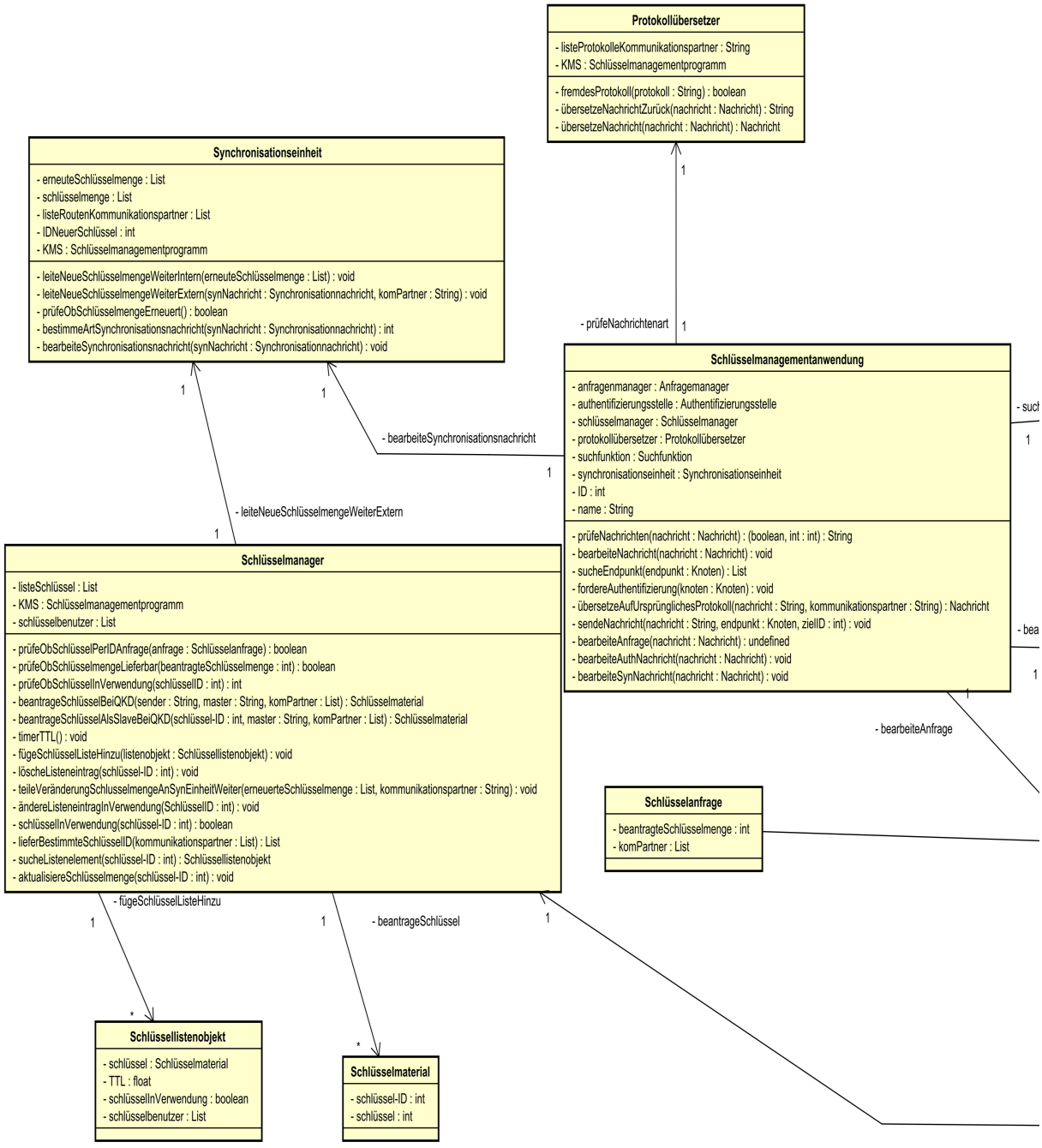
4.2.7 Übergeordnete Managementeinheit

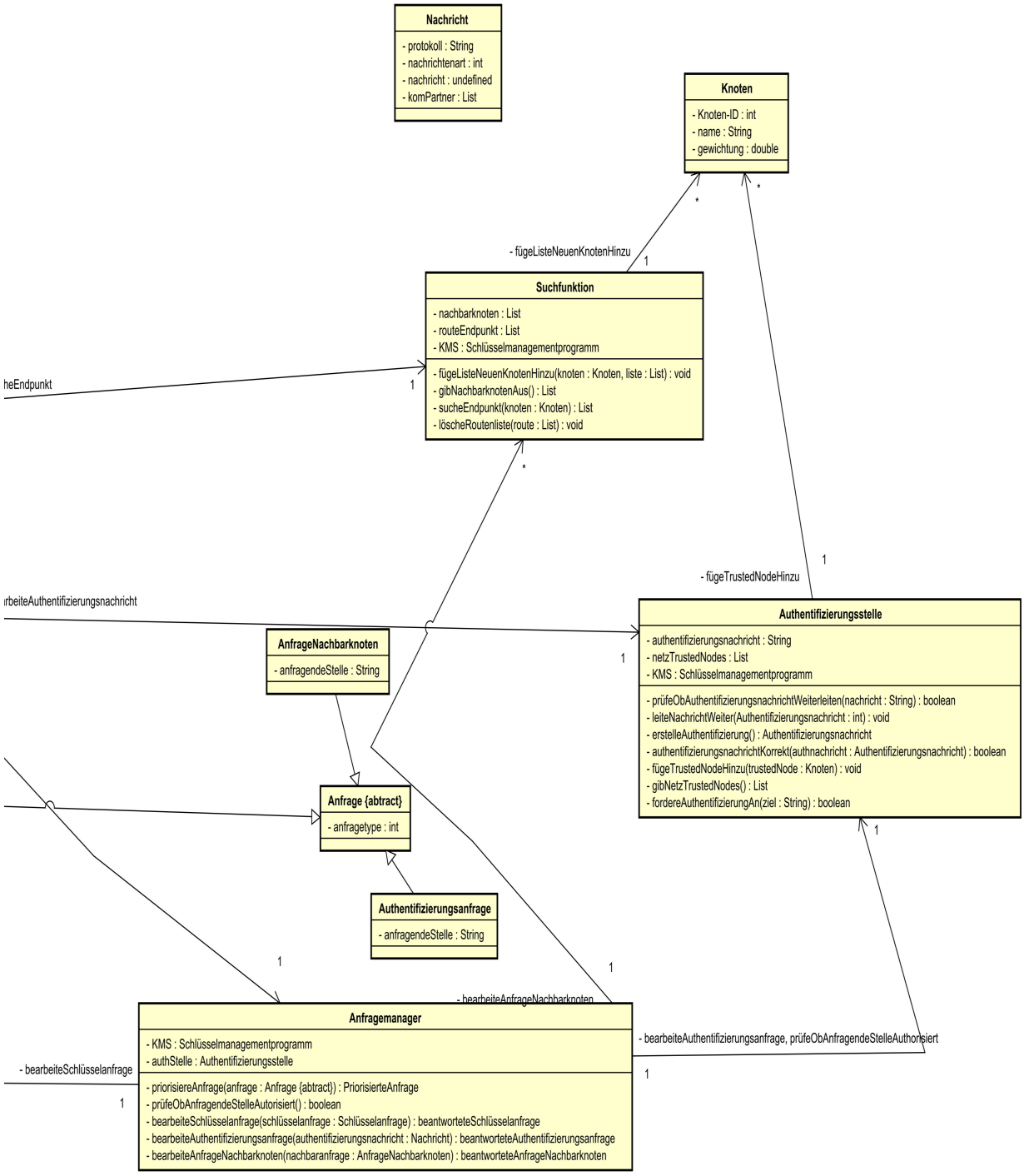
Diese Komponente, welche auch als spätere de facto Hauptklasse angesehen werden kann, dient dem Managen der bisher beschriebenen Komponenten innerhalb des eigentlichen Programms. Die Hauptaufgabe besteht darin Daten, Informationen etc. zu verteilen und Aufgaben an die jeweils dafür zuständigen Komponenten für die weitere Bearbeitung zu delegieren. Die Ergebnisse werden von der Managementeinheit zusammengeführt, um diese dann an die Schlüsselmanagement-Anwendung des Kommunikationspartners zu übermitteln.

4.3 Beschreibung von Klassen und deren Beziehungen

Im Folgenden werden die aus den vorher definierten Komponenten abgeleiteten Klassen, welche zum Verhindern von Unklarheiten dieselben Namen besitzen, vorgestellt. Einzige Ausnahme bezüglich der Namensfindung bildet die Klasse *Schlüsselmanagementanwendung*, welche den übergeordnete Management-Komponente abbildet.
Auf der Abbildung 4.3 ist das gesamte Klassendiagramm dargestellt. Auf die wichtigsten bzw. die funktionellen Klassen wird jeweils separat näher eingegangen. Damit das gesamte Diagramm nicht zu unübersichtlich wird, sind auf der Abbildung 4.3 nur alle Attribute sowie die wichtigsten Methoden der funktionellen Klassen zu sehen. Einfache getter-Methoden wurden ebenfalls für eine bessere Übersicht ausgeblendet.
Zum besseren Verständnis des erstellten Klassendiagrammes dient folgende Legende:

- Beziehungen: Zeigen an welche Komponenten das System benötigt und wie sie sich gegenseitig beeinflussen
  - Assoziationen: Beschreiben explizit Beziehung zwischen Klassen durch gerichtete ($\longrightarrow$ bzw. $\longleftarrow$, $\longleftrightarrow$) oder ungerichtete Verbindungen ($-$)
  - Vererbung: Zeigt die Beziehung zwischen einer generelleren und spezialisierten Klasse an (z.B. Anfrage - Schlüsselanfrage); Pfeil mit hohlem Pfeilkopf, leider nicht zeichenbar
- Multiplizitäten: gibt an wie viele Instanzen eine Klasse ausbilden darf, wobei $* = n$ mit $n \epsilon \mathbb{N}$ gilt
- Sichtbarkeit: gibt an wer eine Methode oder ein Attribut sehen und darauf zugreifen kann; hier nur private Sichtbarkeit ($-$), sodass nur die eigene Klasse dazu befugt ist
- Struktur der Klassendarstellung:
  1. Klassenname
  2. Auflistung der Attribute mit Name und Datentyp
  3. Auflistung der Methoden mit Name sowie ggf. Name und Datentyp der Übersowie Rückgabewerte


Protokollübersetzer
- listeProtokolleKommunikationspartner : String
- KMS : Schlüsselmanagementprogramm
- fremdesProtokoll(protokoll : String) : boolean
- übersetzeNachrichtZurück(nachricht : Nachricht) : String
- übersetzeNachricht(nachricht : Nachricht) : Nachricht
Synchronisationseinheit
- erneuteSchlüsselmenge : List
- schlüsselmenge : List
- listeRoutenKommunikationspartner : List
- IDNeuerSchlüssel : int
- KMS : Schlüsselmanagementprogramm
- leiteNeueSchlüsselmengeWeiterIntern(erneuteSchlüsselmenge : List) : void
- leiteNeueSchlüsselmengeWeiterExtern(synNachricht : Synchronisationnachricht, komPartner : String) : void
- prüfeObSchlüsselmengeErneuert() : boolean
- bestimmeArtSynchronisationsnachricht(synNachricht : Synchronisationnachricht) : int
- bearbeiteSynchronisationsnachricht(synNachricht : Synchronisationnachricht) : void
- prüfeNachrichtenart
Schlüsselmanagementanwendung
- anfragenmanager : Anfragemanager
- authentifizierungsstelle : Authentifizierungsstelle
- schlüsselmanager : Schlüsselmanager
- protokollübersetzer : Protokollübersetzer
- suchfunktion : Suchfunktion
- synchronisationseinheit : Synchronisationseinheit
- ID : int
- name : String
- prüfeNachrichten(nachricht : Nachricht) : (boolean, int : int) : String
- bearbeiteNachricht(nachricht : Nachricht) : void
- sucheEndpunkt(endpunkt : Knoten) : List
- fordereAuthentifizierung(knoten : Knoten) : void
- übersetzeAufUrsprünglichesProtokoll(nachricht : String, kommunikationspartner : String) : Nachricht
- sendeNachricht(nachricht : String, endpunkt : Knoten, zielID : int) : void
- bearbeiteAnfrage(nachricht : Nachricht) : undefined
- bearbeiteAuthNachricht(nachricht : Nachricht) : void
- bearbeiteSynNachricht(nachricht : Nachricht) : void
- bearbeiteSynchronisationsnachricht
- leiteNeueSchlüsselmengeWeiterExtern
Schlüsselmanager
- listeSchlüssel : List
- KMS : Schlüsselmanagementprogramm
- schlüsselbenutzer : List
- prüfeObSchlüsselPerIDAnfrage(anfrage : Schlüsselanfrage) : boolean
- prüfeObSchlüsselmengeLieferbar(beantragteSchlüsselmenge : int) : boolean
- prüfeObSchlüsselInVerwendung(schlüsselID : int) : int
- beantrageSchlüsselBeiQKD(sender : String, master : String, komPartner : List) : Schlüsselmaterial
- beantrageSchlüsselAlsSlaveBeiQKD(schlüssel-ID : int, master : String, komPartner : List) : Schlüsselmaterial
- timerTTL() : void
- fügeSchlüsselListeHinzu(listenobjekt : Schlüssellistenobjekt) : void
- lösche Listeneintrag(schlüssel-ID : int) : void
- teileVeränderungSchlusselmengeAnSynEinheitWeiter(erneuerteSchlüsselmenge : List, kommunikationspartner : String) : void
- ändereListeneintragInVerwendung(SchlüsselID : int) : void
- schlüsselInVerwendung(schlüssel-ID : int) : boolean
- lieferBestimmteSchlüsselID(kommunikationspartner : List) : List
- sucheListenelement(schlüssel-ID : int) : Schlüssellistenobjekt
- aktualisiereSchlüsselmenge(schlüssel-ID : int) : void
- fügeSchlüsselListeHinzu
- beantrageSchlüssel
Schlüsselanfrage
- beantragteSchlüsselmenge : int
- komPartner : List
Schlüssellistenobjekt
- schlüssel : Schlüsselmaterial
- TTL : float
- schlüsselInVerwendung : boolean
- schlüsselbenutzer : List
Schlüsselmaterial
- schlüssel-ID : int
- schlüssel : int


Nachricht
- protokoll : String
- nachrichtenart : int
- nachricht : undefined
- komPartner : List
Knoten
- Knoten-ID : int
- name : String
- gewichtung : double
- fügeListeNeuenKnotenHinzu
Suchfunktion
- nachbarknoten : List
- routeEndpunkt : List
- KMS : Schlüsselmanagementprogramm
- fügeListeNeuenKnotenHinzu(knoten : Knoten, liste : List) : void
- gibNachbarknotenAus() : List
- sucheEndpunkt(knoten : Knoten) : List
- löscheRoutenliste(route : List) : void
heEndpunkt
- fügeTrustedNodeHinzu
rbeiteAuthentifizierungsnachricht
Authentifizierungsstelle
- authentifizierungsnachricht : String
- netzTrustedNodes : List
- KMS : Schlüsselmanagementprogramm
- prüfeObAuthentifizierungsnachrichtWeiterleiten(nachricht : String) : boolean
- leiteNachrichtWeiter(Authentifizierungsnachricht : int) : void
- erstelleAuthentifizierung() : Authentifizierungsnachricht
- authentifizierungsnachrichtKorrekt(authnachricht : Authentifizierungsnachricht) : boolean
- fügeTrustedNodeHinzu(trustedNode : Knoten) : void
- gibNetzTrustedNodes() : List
- fordereAuthentifizierungAn(ziel : String) : boolean
AnfrageNachbarknoten
- anfragendeStelle : String
Anfrage {abtract}
- anfragetype : int
Authentifizierungsanfrage
- anfragendeStelle : String
- bearbeiteAnfrageNachbarknoten
Anfragemanager
- KMS : Schlüsselmanagementprogramm
- authStelle : Authentifizierungsstelle
- bearbeiteAuthentifizierungsanfrage, prüfeObAnfragendeStelleAuthorisiert
- bearbeiteSchlüsselanfrage
- priorisiereAnfrage(anfrage : Anfrage {abtract}) : PriorisierteAnfrage
- prüfeObAnfragendeStelleAutorisiert() : boolean
- bearbeiteSchlüsselanfrage(schlüsselanfrage : Schlüsselanfrage) : beantworteteSchlüsselanfrage
- bearbeiteAuthentifizierungsanfrage(authentifizierungsnachricht : Nachricht) : beantworteteAuthentifizierungsanfrage
- bearbeiteAnfrageNachbarknoten(nachbaranfrage : AnfrageNachbarknoten) : beantworteteAnfrageNachbarknoten

4.3.1 Schlüsselmanagementanwendung

Auf der Abbildung 4.2 ist die Hauptklasse *Schlüsselmanagementanwendung* zu sehen, welche mit den entsprechenden Komponenten die Gesamtanwendung eines quantenbasierten Schlüsselmanagementanwendung repräsentiert. Diese Klasse ist die Schnittstelle nach außen zu anderen Schlüsselmanagementanwendungen und nimmt somit Nachrichten, egal ob eine Anfrage oder Authentifizierungsaufforderung, an. Durch die Methode *prüfeNachrichtenart()* wird festgestellt um was für eine Nachricht es sich handelt, um diese dann entsprechend an die dafür vorgesehene Komponente zur Weiterverarbeitung zu delegieren. Nach diesem Prozess wird, falls vorgesehen, das Ergebnis der Anforderung wieder an den Sender zurückgesendet. Somit entfallen auf diese Klasse hauptsächlich delegierende bzw. das Verteilen der Aufgaben, wodurch sich diese als Manager der gesamten Anwendung beschreiben lässt.

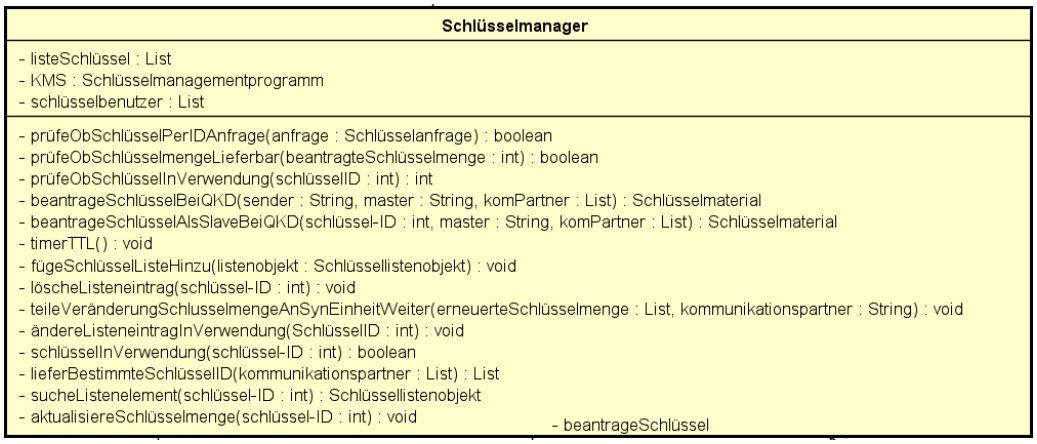


Abbildung 4.2: Darstellung der Klasse Schlüsselmanagementanwendung

4.3.2 Protokollübersetzer

Die Klasse *Protokollübersetzer* repräsentiert die gleichnamige Komponente und ist dafür zuständig fremde Protokolle zu übersetzen sowie Nachrichten, welche gesendete werden sollen, in ihre ursprünglichen Protokolle umzuwandeln. Dafür sind die beiden Methoden *übersetzeNachricht()* sowie *übersetzeNachrichtZurück()* verantwortlich. *fremdesProtokoll()* gibt einen boolean-Wert zurück, welche angibt ob die übergebene Nachricht in einem zu übersetzendem Fremdprotokoll verfasst ist oder nicht.

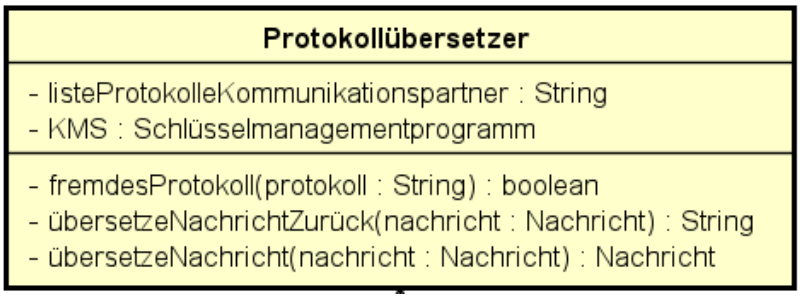


Abbildung 4.3: Darstellung der Klasse Protokollübersetzer

4.3.3 Authentifizierungsstelle

Diese Klasse prüft mit der Methode *prüfeObAuthentifizierungsnachtichtWeiterleiten()* zunächst ob es sich um eine Aufforderung zur Authentifizierung von einem anderen Knoten oder um eine weiterzuleitende Nachricht während eines externen Authentifizierungsprozesses handelt. Bei einem externen Authentifizierungsprozess dient der eigene Knoten nur zur Weiterleitung einer Authentifizierungsanforderungen von einem Knoten A an einen anderen Knoten C. Falls keine Authentifizierungsaufforderung zu beantworten ist, wird diese Nachricht einfach mit *leiteNachrichtWeiter()* an den adressierten Empfänger weitergeleitet. Ansonsten diese Anforderung an die Klasse *Anfragemanager* zur weiteren Verarbeitung der Anfrage weitergeleitet. Anhand der Methode *fordereAuthentifizierungAnKnoten()* kann zum Aufbau bzw. der Erweiterung eines Netzes von Trusted Nodes eine Authentifizierungsnachricht an ein anderen Knoten gesendet werden. Mit *authentifizierungsnachrichtKorrekt()* kann die aufgeforderte Authentifizierung eines anderen Knotens überprüft werden.

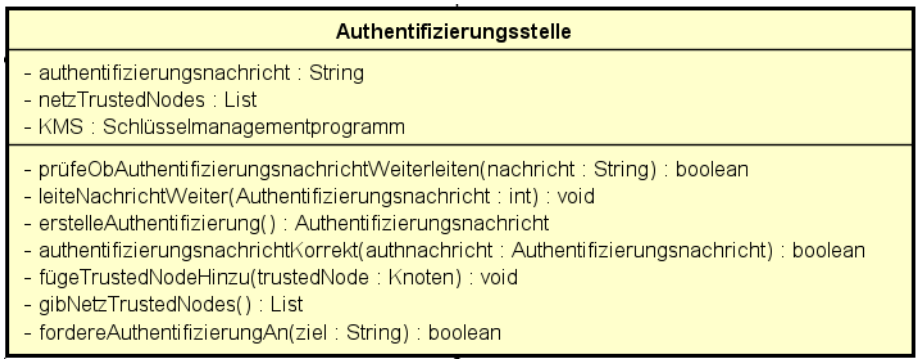


Abbildung 4.4: Darstellung der Klasse Authentifizierungsstelle

4.3.4 Suchfunktion

Mit der Klasse *Suchfunktion* wird die Anforderung an die gleichnamige Komponente erfüllt, indem mit der Methode *sucheEndpunkt()* nach einem übergebenen Knoten gesucht und gleichzeitig die dadurch erarbeitete Route für die darauffolgende Kommunikation gespeichert werden kann. Außerdem kann per *gibNachbarknotenAus()*-Aufruf eine Liste aller Nachbarknoten geliefert werden, wenn beispielsweise eine Schlüsselmanagementanwendung eines anderen Knotens eine Route zu einem Endpunkt suchen sollte.

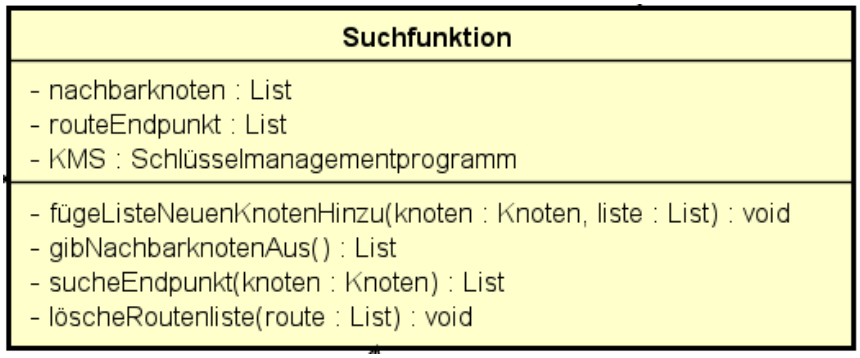


Abbildung 4.5: Darstellung der Klasse Suchfunktion

4.3.5 Anfragemanager

Der *Anfragemanager* ist für das delegieren der Anfragen zu den jeweiligen Komponenten, das Priorisieren der Anfragen sowie das übermitteln der Ergebnis an die Hauptklasse zuständig. Zunächst werden durch die Methoden *prüfeAnfrageart()* mit einem Rückgabewert vom Typ Integer, und *prüfeObAnfragendeStelleAutorisiert()* mit einem zurückgegebenen boolean-Wert grundlegende Informationen ermittelt. Letztere überprüft dabei ob die anfragende Partei überhaupt berechtigt ist eine bestimmte Anfrage, wie beispielsweise eine Schlüsselanfrage, zu stellen. Die Methode *priorisiereAnfrage()* priorisiert die übergebene Anfrage, welche dabei zum Typ PriorisierteAnfrage umgewandelt wird. *bearbeiteSchlüsselanfrage()* und *bearbeiteAuthentifizierungsnachricht()* sind unteranderem die beiden Methoden welche nach der Überprüfung der Nachricht, diese zur weiteren Bearbeitung an die entsprechenden Komponenten delegieren.

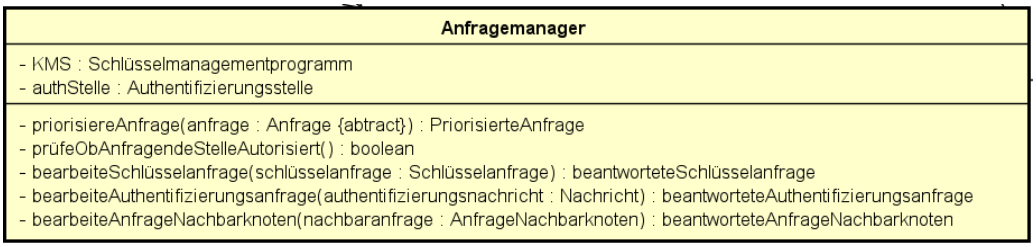


Abbildung 4.6: Darstellung der Klasse Anfragemanager

4.3.6 Schlüsselmanager

Diese Klasse bildet die Hauptkomponente der gesamten Anwendung, denn diese übernimmt sowohl das Beziehen des Schlüsselmaterials vom QKD-Gerät als auch das komplette Management dieser Daten. Dabei prüfen die Methoden *prüfeObSchlüsselmengeLieferbar()* sowie *prüfeObSchlüsselPerIDAnfrage()* zunächst ob die angeforderte Schlüsselmenge lieferbar ist und bei der Anfrage das komplette Schlüsselpaket oder nur der Schlüssel gefordert wird. Mit *beantrageSchlüsselBeiQKD()* kann ein komplettes Schlüsselpaket, beispielsweise als Master und Initiator einer Kommunikation, beim QKD-Gerät bezogen werden. Die Methode *beantrageSchlüsselAlsSlaveBeiQKD()* wiederum ermöglicht das Beziehen eines Schlüssels per Schlüssel-ID. In den meisten Fällen geschieht dies als Slave, welcher die benötigten IDs vom Master übermittelt bekommt, wodurch sich der Name der Methode ableitet.
Das Management der Schlüsselmaterialien findet mit einer tabellarischen Liste statt wozu

die restlichen Methoden der Klasse *Schlüsselmanager* dienen. Beispielsweise wird mit *timerTTL()* ein Timer bei jedem Bezug eines Schlüssels als Master gestartet, welche dafür sorgt, dass jeder Schlüssel nur eine maximale Nutzungszeit besitzt und danach nicht mehr für die weitere Kommunikation zur Verfügung steht. Dazu wird mit *löscheListeneintrag()* der bzw. die entsprechenden Schlüssel mit allen Parameter aus der Auflistung zu entfernen und somit nicht mehr für die weitere Kommunikation genutzt. Zeitgleich werden der Synchronisationskomponente die IDs der nun erneuerten Schlüsselmenge mitgeteilt und durch diese an die Slaves weitergeleitet, damit diese ihre Schlüsselmenge ebenfalls aktualisieren können.

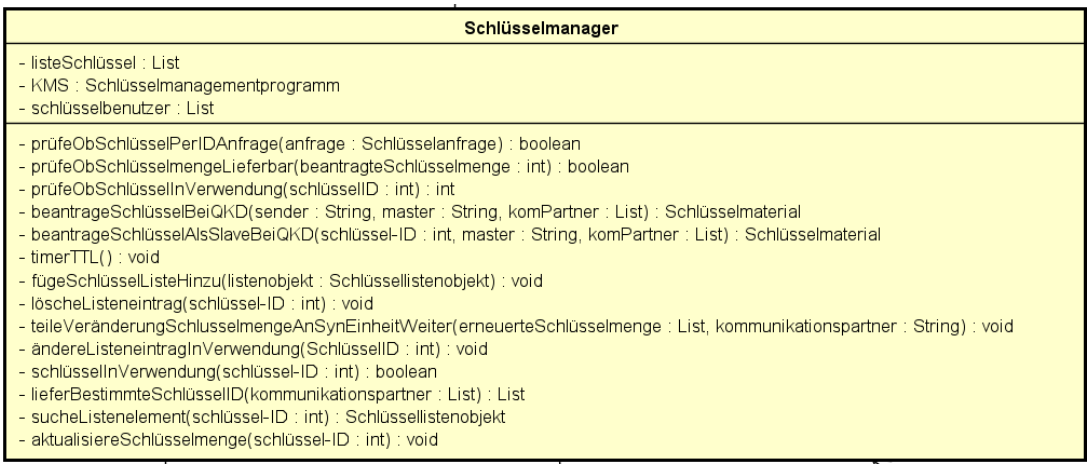


Abbildung 4.7: Darstellung der Klasse Schlüsselmanager

4.3.7 Synchronisationseinheit

Die Klasse *Synchronisationseinheit* sorgt bei allen an einer Kommunikation beteiligten Schlüsselmanagementanwendungen für eine gleichbleibende Schlüsselmenge. Dies kann sich ändern, wenn beispielsweise neues Schlüsselmaterial während einer Kommunikation gebraucht, bezogen sowie genutzt werden soll. Anhand der Methode *leiteNeueSchlüsselmengeWeiterExtern()* werden die von der *Schlüsselmanager*-Klasse bezogenen, neuen Schlüssel-IDs an die *Synchronisationseinheit* des Kommunikationspartners weitergeleitet. Durch die Methode *bestimmeArtSynNachricht()* kann durch *bearbeiteSynNachricht* die Art der Synchronisationsnachricht bestimmt werden, damit diese entsprechend bearbeitet werden kann.

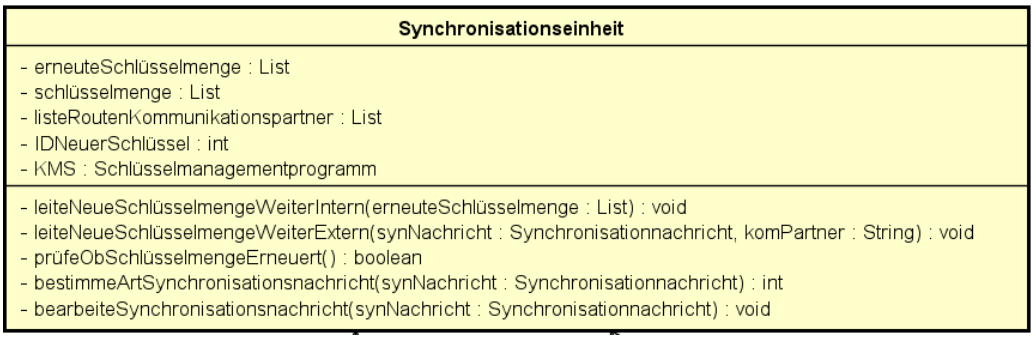


Abbildung 4.8: Darstellung der Klasse Synchronisationseinheit

4.4 Verhaltensdarstellung durch Sequenzdiagramme

Im vorherigen Abschnitt wurde die Programmstruktur mit den einzelnen Klassen und deren Beziehung anhand eines Klassendiagramm präsentiert. Im Folgenden wird nun durch die sequenzielle Darstellung des Programmablaufs mit den aufeinander folgenden Befehlen während verschiedener Ereignisse das Zusammenspiel der verschiedenen Klassen dargestellt.
Diese Abläufe gleichen denen bei einer späteren Umsetzung des aufgestellten Entwurfs für eine Schlüsselmanagementanwendung. Für diese Darstellungsart werden Interaktionsdiagramme, genauer gesagt Sequenzdiagramme, genutzt. Zunächst wurden alle Anforderungen auf daraus resultierende, abbildbare Ereignisse überprüft sowie die eventuelle Anzahl der sich ergebenen Sequenzdiagramme. Dazu sind in der folgenden Auflistung alle Anforderungen mit den sich daraus ergebenen Diagrammen bzw. deren Namen aufgeführt.

- Protokollunabhängige Schnittstellen
  - Protokollübersetzung durch die Klasse *Protokollübersetzer*
- Authentifizierung
  - Weiterleitung einer Authentifizierungsnachricht an eine Anwendung oder ein QKD-Gerät
  - Authentifizierungsanforderung an die entsprechende Komponente einer anderen KMS
  - Beantwortung einer Authentifizierungsaufforderung
- Eindeutige Identifikation
  - Kein Diagramm erstellbar, da jeder Schlüssel, jede Anwendung und jedes QKD-Gerät eine entsprechende ID als Attribut aufweist
- Suchfunktion für Endpunkte
  - Befehlsabfolge beim Aufruf der Such-Methode mit einem übergebenen Endpunkt
- Anfragenmanagement
  - Prüfung der Anfrageart durch den Anfragenmanger (Schlüssel- oder Authentifizierungsanfrage)
  - Priorisierung einer Anfrage
  - Überprüfung ob die anfragende Stelle für die Anfrageart autorisiert ist, beispielsweise bei Schlüsselanfrage
  - Komplette Bearbeitung einer Anfrage
- Umgang mit Schlüsselmaterial
  - Bearbeitung einer Schlüsselanforderung eines kompletten Schlüsselpaketes (ID + Schlüssel)
  - Bearbeitung einer Schlüsselanforderung per Schlüssel-ID
  - Hinzufügung eines Schlüssel mit allen relevanten Informationen (z.B. Nutzer) in die tabellarische Auflistung
  - Entfernung eines Schlüssel mit allen relevanten Informationen aus der tabellarischen Auflistung, wenn beispielsweise TTL abgelaufen
  - Überprüfung ob ein Schlüssel zur übergebenen ID schon für ein Kommunikation verwendet wird
  - Änderung des Listeneintrages, wenn Schlüssel verwendet werden soll
  - Übertragung von Schlüssel-IDs an den Kommunikationspartner
- Fortlaufende Synchronisation

  - Weiterleitung der Schlüssel-IDs der erweiterten Schlüsselmenge an die Synchronisationeinheit de Kommunikationspartners
  - Weiterleitung der empfangenen Schlüssel-IDs der erweiterten Schlüsselmenge an den Schlüsselmanager zur Anpassung der bisherigen Schlüsselmenge
- Protokollbasiertes Fehlermanagement
  - Kein Diagramm erstellbar, da jede Komponente auftretende Fehler während des Protokollablaufs selber erkennt und darauf reagiert

Somit ergeben sich nach dieser Methodik insgesamt 18 zeichenbare Sequenzdiagramme, wovon allerdings nicht alle, welche eintreten können, abgebildet werden, sondern nur einige repräsentative Beispiele. Außerdem können teilweise die in den Diagrammen abgebildeten Abfolgen der Methodenaufrufe in umfangreicheren Aufrufen dargestellt werden. Beispielsweise kann das Sequenzdiagramm für die Schlüsselzuweisung in der Darstellung der Befehlsabfolge zu einer Schlüsselanforderung abgebildet werden. Daher ergeben sich faktisch nur noch 11 relevante Diagramme.
Insgesamt sind die drei folgenden Beispieldiagramme gezeichnet worden:

- Bearbeitung einer (Schlüssel-)Anfrage durch den Anfragenmanager (hier Anforderung komplettes Schlüsselpaket)
- Authentifizierungsaufforderung von einem anderen Knoten ausgehend
- Suche nach einem übergebenen Endpunkt sowie Abspeicherung der entsprechenden Route

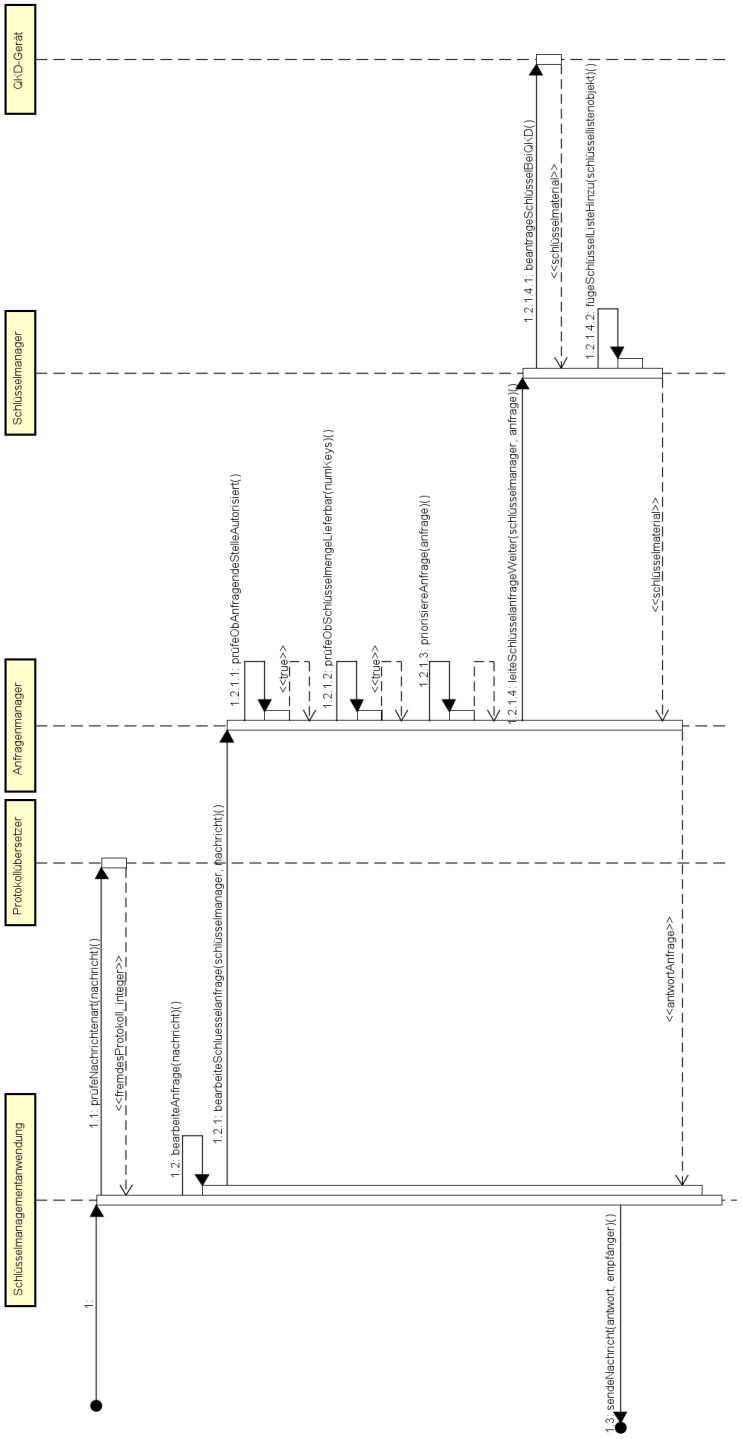


Abbildung 4.9: Darstellung eines sequenziellen Ablaufs bei einer Schlüsselanforderung zur Erläuterung des Zusammenspiels der einzelnen Komponenten

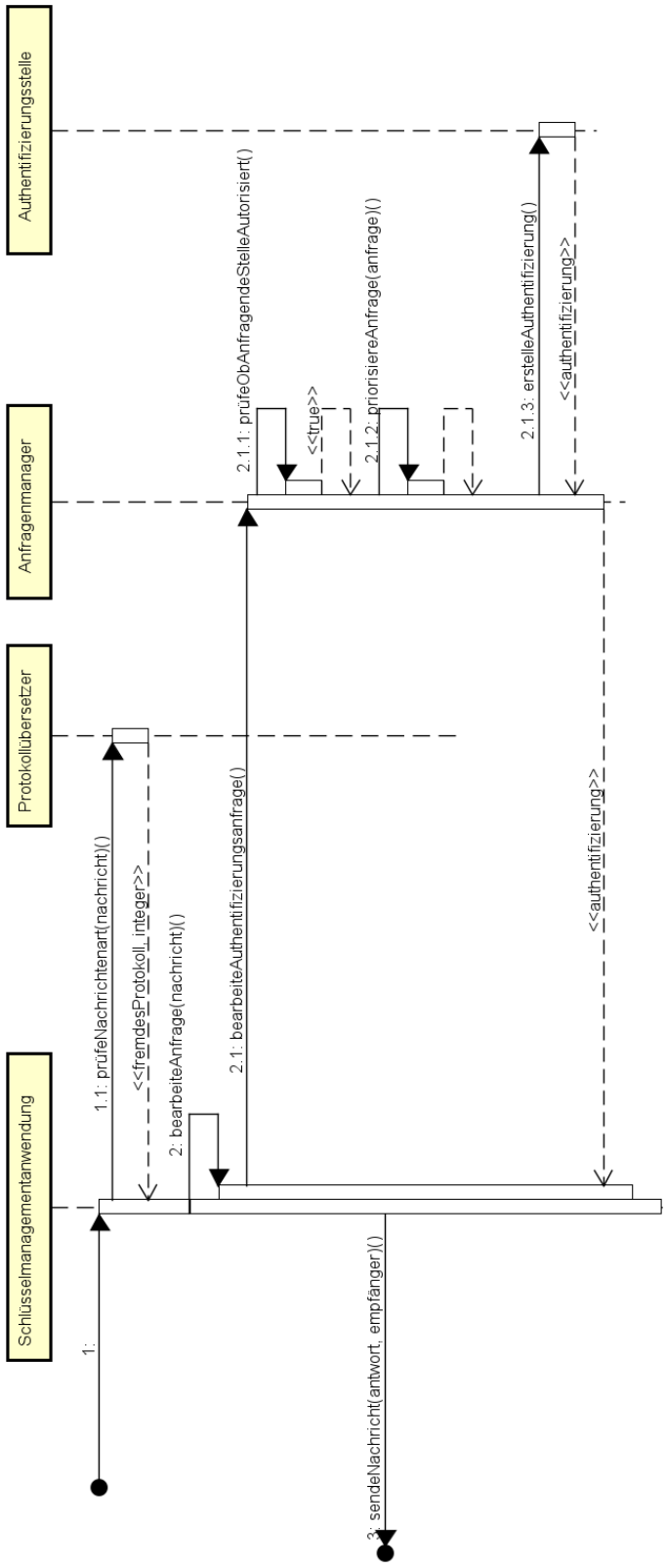


Abbildung 4.10: Darstellung eines sequenziellen Ablaufs bei einer Authentifizierungsanfrage mit den nacheinander ablaufenden Methodenaufrufen

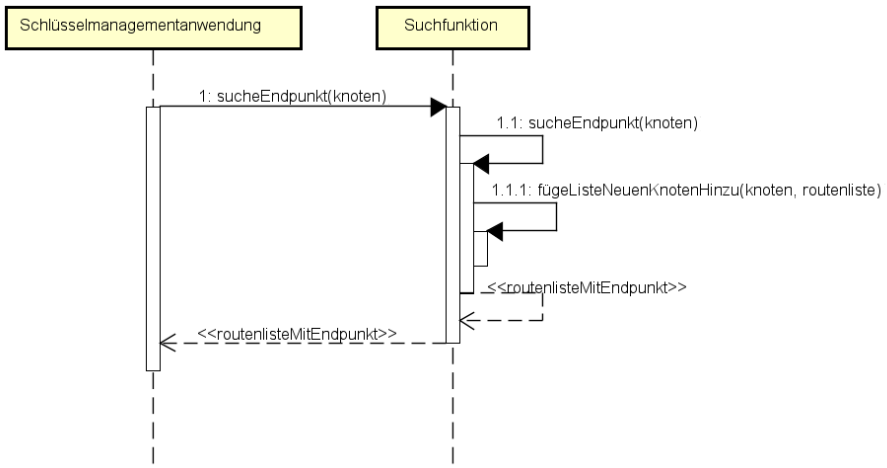


Abbildung 4.11: Darstellung eines sequenziellen Ablaufs bei dem Aufruf der Suchfunktion mit einem übergebenen Endpunkt

Kapitel 5

Umsetzung eines Prototypen

Anhand des im vorherigen Kapitel erarbeitetem Programmentwurf soll nun der Prototyp, welcher die Umsetzbarkeit dieses Entwurfs zeigen soll, implementiert sowie zur Nachvollziehbarkeit der Umsetzung dokumentiert werden.
Dazu wird zunächst die verwendete Programmiersprache Python und die Entwicklungsumgebung PyCharme kurz vorgestellt sowie begründet warum sich für diese entschieden worden ist. Danach wird der fertige Prototyp mit seinen Funktionalitäten anhand von Codeabschnitten kurz vorgestellt. Vorab werden einige Anmerkungen bezüglich weiterer implementierter Strukturen gemacht, welche für die Testung sowie Debugging des Prototypens benötigt worden sind. Bei der darauffolgenden Codebeschreibung werden nur die wichtigsten Methoden im Rahmen einer Schlüsselanforderung für eine bessere Veranschaulichung beschrieben.

5.1 Vorstellung Programmiersprache und Entwicklungsumgebung

Für die Implementierung des Prototypens wurde die Programmiersprache Phyton benutzt, da diese für eine gut lesbare sowie knappe Programmstruktur förderlich ist. Des Weiteren findet diese Sprache im Rahmen der Projekt MuQuaNet bis dato große Verwendung, weshalb durch diese Wahl eine spätere Verwendung des implementierten Prototypens im Rahmen des Projektes realistischer scheint. Außerdem lassen sich Strukturen schnell an Anforderungen anderer Programme anpassen sowie ist ein weiterer Ausbau des Prototypens einfacher realisierbar.
PyCharm ist eine Python-Entwicklungsumgebung und wurde als kostenlose Entwicklungsumgebung gewählt, wobei durch den Studentenstatus die Aufwertung zur Professional-

Version möglich war und auch entsprechend genutzt worden ist. Diese liefert viele verschiedene, nützliche Features, wie beispielsweise Codevervollständigung, Codeanalysen und Refactorings. Hierbei ist in der Professional-Version die Unterstützung für moderne Webentwicklungs-Frameworks wie Flask ermöglicht worden, welche in Hinsicht entscheidend für die Kommunikationsstruktur des Prototypens ist, da diese auf Flask basiert. [27]

5.2 Vorstellung des Prototypen und seiner Funktionalitäten

Während der Implementierung des Prototypens anhand des vorher erstellten Klassendiagrammes war es notwendig weitere Infrastrukturen, beispielsweise für die Kommunikation mehrerer Instanzen einer Schlüsselmanagementanwendung untereinander, zusätzlich einzupflegen. Deshalb werde diese vorab kurz benannt und in ihrer Funktionsweise erklärt, ohne dabei näher auf den Quellcode einzugehen.

5.2.1 Voranmerkungen zur Implementierung des Prototypens

Für eine vereinfachte Kommunikation zwischen den Anwendungen, welche einen Schlüssel für die Verschlüsselung benutzen wollten, und Instanzen der Schlüsselmanagementanwendungen wurde auf zwei Hilfsmittel gesetzt. Eine lokale Datenbank, um die IP-Adresse sowie die Ports der Empfänger einer Nachricht zu ermitteln, sowie die Nutzung von Sockets für den Netzverkehr.

Lokale Datenbank mongoDB

Bei mongoDB handelt es sich um eine dokumentenorientierte NoSQL-Datenbank, wodurch Verwaltung und Zugriff der Daten anhand von JSON-Objekten geschieht. Im Rahmen der Implementierung des Prototypens wurde sich für eine lokale Datenbank entschieden.
Die Datenbank wurde genutzt, um die IP-Adressen und Ports der unterschiedlichen Schnittstellen zu hinterlegen und zugreifbar zu machen. Für das Managen der Datenbank wurde extra eine Klasse geschrieben, welche durch einen entsprechenden Methodenaufruf die gewünschten Daten liefert. Somit ist es jederzeit den entsprechenden Kommunikationspartnern möglich diese Daten durch folgenden Aufruf zu beziehen:

```
1 def sucheObjekt(self, knotenName):
2     obj = self.mycol.find_one({'Knoten': knotenName})
3     return obj
```

Listing 5.1: Methode zum Auslesen eines bestimmten Datensatztes

mycol.fond_one() ist eine Methode, welche von mongoDB gestellt wird und durch den Import des Packets *pymongo* genutzt werden kann.
Jede Schnittstelle wird bei der Initialisierung mit den entsprechenden Informationen wie IP-Adresse, Port, Name sowie einer ID eingefügt. Falls ein Datensatz schon in der Datenbank enthalten sein sollte, da diese nicht jedes Mal neu befüllt werden muss, wird einfach eine entsprechende Information in der Konsole ausgegeben, wie am Quellcode zu sehen:

```
1  def dbAufsetzen(self, knotenName, id, HOST, PORT):
2      try:
3          self.myclient=pymongo.MongoClient("mongodb://localhost:27017/")
4          self.mydb = self.myclient["KMS"]
5          self.mycol = self.mydb["Knotenpunkte"]
6          if id == 1:
7              idDB = id + len(knotenName)
8          elif id == 2:
9              idDB = id + len(knotenName)
10         knoten = {"_id": idDB, "Knoten": knotenName, "HOST": HOST, "
               PORT": PORT}
11         self.knoten = self.mycol.insert_one(knoten)
12     except Exception:
13         error = "Knoten {} schon eingefuegt"
14         print(error.format(knotenName))
```

Listing 5.2: Methode zum Einfügen einer Schnittstelle in die Datenbank

Netzkommunikation via Sockets mit zur Hilfenahme von Threads

Für die Kommunikation bzw. den Versand der Nachrichten innerhalb des Netzes zwischen den Schnittstellen der Anwendungen und Schlüsselmanagement-Instanzen wurden Sockets verwendet. Damit der Server, welcher auf das Empfangen einer Nachricht wartet, im Hintergrund laufen kann, Threading benutzt, um parallel mehrere Programmabläufe zu ermöglichen.
Es wurden jeweils für die drei folgenden Schnittstellen eigene Socket-Server verwendet:

- Schnittstelle Schlüsselmanagementanwendung nach außen
- Schnittstelle zur Schlüsselannahme der anfordernden Stelle

- Schnittstelle zur allgemeinen Kommunikation der schlüsselnutzenden Anwendungen

Die Implementierung des Servers sowie des Clients ist anhand von Tutorials erstellt worden und nur das Verarbeiten der empfangenen Nachrichten ist somit in Eigenarbeit entstanden. Zur Veranschaulichung dient folgende Code-Beispiel, in dem die Aufbereitung einer Nachricht an eine Instanz des Schlüsselmanagementprogramm nachzuvollziehen ist:

```
1  def bereiteDatenAuf(self, data):
2      nachricht = None
3      if data['Typ'] == 0:            # 0 = Schluesselanfrage
4          nachricht = Nachricht(data['Protokoll'], data['Typ'], [data['
               Menge']], [data['Master'], data['Slaves']])
5      elif data['Typ'] == 1:          # 1 = Authentifizierungsanfrage
6          nachricht = Nachricht(data['Protokoll'], data['Typ'], data['
               Authentifizierungsgrund'], data['Sender'])
7      elif data['Typ'] == 2:          # 2 = Nachbarknotenanfrage
8          nachricht = Nachricht(data['Protokoll'], data['Typ'], data['
               Ausgenommene Knoten'], data['Sender'])
9      elif data['Typ'] == 3:          # 3 = Authentifizierungsnachricht (
           weiterleiten)
10         nachricht = Nachricht(data['Protokoll'], data['Typ'], data['
               Authentifizierungsgrund'], [data['Sender'], data['Empfaenger
               ']])
11     elif data['Typ'] == 4:          # 4 = Synchronisationsnachricht
12         if data['SynTyp'] == 0:
13             nachricht = Nachricht(data['Protokoll'], data['Typ'], [data
                   ['Menge'], data['Schluessel-ID']],
14                                   [data['Master'], data['Slaves']])
15         elif data['SynTyp'] == 1:
16             nachricht = Nachricht(data['Protokoll'], data['Typ'], data[
                   'Schluessel-ID abgelaufener TTL'], [data['Master'], data
                   ['Slaves']])
17     else:
18         print("Fehlerhafter Nachrichtentyp angegeben")
19     self.uebergeordneteStelle.bearbeiteNachricht(nachricht)
```

Listing 5.3: Methode zur Datenaufbereitung empfangener Nachrichten für die Bearbeitung

In der Variable *uebergeordneteStelle* ist die jeweilige Instanz der Schlüsselmanagementanwendung hinterlegt, damit eine Bearbeitung der empfangenen Nachricht eingeleitet werden kann.

5.2.2 Beschreibung unterschiedlicher Codefragmente

Um die Beschreibung unterschiedlicher Codefragmente anschaulicher zu gestalten, wird der Ablauf einer Schlüsselanforderung anhand der Funktionalität des implementierten Codes beschrieben.
Viele Typen, beispielsweise zur Bestimmung der Nachrichtenart, sind in Form von einem Integer-Wert gespeichert, da dies eine einfache, unkomplizierte und leicht anzupassende Lösung ist.

Methode bearbeiteNachricht

Die Methode *bearbeiteNachricht()* stellt den Kopf der Bearbeitungskette dar und untersucht zunächst durch den Methodenaufruf *pruefeNachricht()* ob der Sender ein fremdes Protokoll benutzt und um welche Nachrichtenart es sich handelt. Falls notwendig wird die Nachricht in eine verarbeitbare Form durch den Protokollübersetzer übersetzt, um entsprechend der Nachrichtenart verarbeitet zu werden. Wenn beispielsweise die Nachricht eine Schlüsselanfrage von einer Anwendung ist, dann diese zur Bearbeitung an den Anfragenmanager weitergeleitet, um das zurückgelieferte Ergebnis mit *sendeNachricht()* an die anfragende Stelle zurückzusenden.

```
1  def bearbeiteNachricht(self, nachricht):
2      nachrichtenart = self.pruefeNachricht(nachricht)
3      self.sender = nachricht.getKomPartner()[0] + " Schluessel"
4      if nachrichtenart[0] == True:
5          self.protokolluebersetzer.uebersetzeNachricht(nachricht)
6      else:
7          if nachrichtenart[1] == 0 or nachrichtenart[1] == 1 or
               nachrichtenart[1] == 2:
8              bearbeiteteAnfrage = self.bearbeiteAnfrage(nachricht)
9              antwort = {"Status": "Bearbeitet", "Beantragte Schluessel":
                   bearbeiteteAnfrage.getBeantragtesSchluesselmaterial()}
10             self.sendeNachricht(antwort, self.sender)
11         elif nachrichtenart[1] == 3:
12             self.bearbeiteAuthNachricht(nachricht)
13         elif nachrichtenart[1] == 4:
14             self.bearbeiteSynNachricht(nachricht)
15         else:
16             print("Fehler, unbekannter Nachrichtentyp")
```

Listing 5.4: Methode zur Annahme und zum Delegieren der Bearbeiung einer Nachricht

Methode sendeNachricht

Dazu wird das Ergebnis der Bearbeitung in ein JSON-Objekt eingefügt und der Methode *sendeNachricht()* übergeben, welche bei der Datenbank sowohl die IP-Adresse als auch den Port der anfragenden Stelle anfragt. Anhand dessen kann die Nachricht über das Socket-System versendet werden.

```
1 def sendeNachricht(self, nachricht, empfaenger):
2     hostEmpfaenger = self.dbhandler.sucheObjekt(empfaenger)['HOST']
3     portEmpfaenger = self.dbhandler.sucheObjekt(empfaenger)['PORT']
4     self.socketClient.sendeNachrichtAnGegenueber(hostEmpfaenger,
          portEmpfaenger, nachricht)
```

Listing 5.5: Versenden von der bearbeiteten Anfrage an die anfragende Stelle

Methode bearbeiteAnfrage

Die Schlüsselmanagement untersucht durch den Aufruf der Methode *bearbeiteAnfrage()* mit der ursprünglichen Nachricht als Übergabeparameter die Anfrage. Dieser Schritt dient zur besseren Koordinierung des Anfragemanagers. Wenn es sich beispielsweise um eine Schlüsselanfrage handelt, kann diese direkt per Methodenaufruf *bearbeiteSchluesselanfrage()* vom Anfragemanager bearbeitet werden.

```
1  def bearbeiteAnfrage(self, nachricht):
2      anfrageart = nachricht.getNachrichtenart()
3      if anfrageart == 0:
4          beantragterSchluessel = self.anfragemanager.
               bearbeiteSchluesselanfrage(self.schluesselmanager, nachricht
               )
5          return beantragterSchluessel
6      elif anfrageart == 1:
7          auth = self.anfragemanager.bearbeiteAuthentifizierungsanfrage()
8          return auth
9      elif anfrageart == 2:
10         nachbarknoten = self.anfragemanager.
               bearbeiteAnfrageNachbarknoten()
11         return nachbarknoten
12     else:
13         print("Fehler bei Anfragenbearbeitung")
```

Listing 5.6: Delegieren der Aufgaben für den Anfragenmanager

Methode bearbeiteSchluesselanfrage

Eine Ebene tiefer in diesem Verarbeitungsprozess befindet sich der Anfragemanager, welcher von der übergeordneten Ebene die Aufgabe zur Beantwortung einer Schlüsselanfrage erteilt sowie die ursprüngliche Nachricht als Übergabeparameter bekommen hat. Außerdem ist die vom Schlüsselmanagementanwendung erstellte Instanz des Schlüsselmanagers ebenfalls ein Übergabeparameter, damit jedes Mal auf dieselbe Instanz zugegriffen werden und somit zu keinen Komplikationen durch beispielsweise unterschiedliche Instanzen innerhalb einer Schlüsselmanagementanwendung kommt.
Als ersten Schritt werden zwei Hilfsvariablen erstellt, wobei eine den boolean-Wert speichern soll, ob es sich um eine Schlüsselanfrage per Schlüssel-ID handelt, und die andere Variable die beantragte Schlüsselmenge speichert. Die weitere Bearbeitung findet anhand einer if-Abfolge statt, bei der zunächst geprüft wird, ob die anfragende Stelle für eine Schlüsselanfrage autorisiert ist und danach, ob die beantragte Schlüsselmenge lieferbar ist. Wenn beide Bedingungen erfüllt sind, wird die Anfrage nach der Maxime bearbeitet, ob Schlüssel per ID angefragt werden (boolean-Wert der Hilfsvariable True) oder nicht.
In diesem Fall handelt es sich um eine Anfrage für ein Schlüsselpaar (Schlüssel + ID). Als nächstes wird nun eine konkrete Schlüsselanfrage anhand der gleichnamigen Klasse erstellt, um der Klasse *Schluesselmanager* die benötigten Informationen für sowohl den Schlüsselbezug vom QKD-Gerät als auch zur Verwaltung des Schlüsselpaares erhält.
Das Beziehen des Schlüsselpaares vom *Schlüsselmanager* geschieht per while-Schleife, welche für den Wert der Hilfsvariable *numKeys* entsprechend per Methodenaufruf *beantrageSchluesselBeiQKD()* die benötigte Anzahl bezieht und in einer Hilfsliste zwischenspeichert. Diese wird an die Methode an die übergeordnete Ebene als beantragte Schlüsselmenge geliefert. Jeder Eintrag besteht sowohl aus der Schlüssel-ID als auch dem eigentlichen Schlüssel.

```
1 def bearbeiteSchluesselanfrage(self, schluesselmanager, nachricht):
2     schluesselanfragetyp = schluesselmanager.
          pr feObSchl sselPerIDAnfrage(nachricht.getNachricht())
3     numKeys = nachricht.getNachricht()[0]
4     if self.pruefeOAnfragendeStelleAuthorisiert():
5         if schluesselmanager.pruefeObSchl sselmengeLieferbar(numKeys):
6             if schluesselanfragetyp == 0:
7                 self.KMS.sender = "Anwendung 2 Schluessel"
8                 schluesselanfrage = SchluesselanfragePerID(nachricht.
                      getNachrichtenart(), nachricht.getNachricht()[0],
                      nachricht.getKomPartner(), nachricht.getNachricht()
                      [1])
9                 schluessel = schluesselmanager.
                      beantrageSchluesselBeiQKDAlsSlave(schluesselanfrage.
                      getKeyID(), schluesselanfrage.getKomPartner()[0],
```

```
                    schluesselanfrage.getKomPartner()[1])
10                  antwortIDAnfrage = [schluessel]
11                  return antwortIDAnfrage
12              else:
13                  schluesselanfrage = Schluesselanfrage(nachricht.
                        getNachrichtenart(), nachricht.getNachricht()[0],
                        nachricht.getKomPartner())
14                  listeSchluessel = []
15                  i = 1
16                  while i <= numKeys:
17                      listeSchluessel.append(schluesselmanager.
                            beantrageSchluesselBeiQKD(self.KMS.getID(),
                            schluesselanfrage.getKomPartner()[0],
                            schluesselanfrage.getKomPartner()[i]))
18                      i = i+1
19                  antwortAnfrage = listeSchluessel
20                  return antwortAnfrage
21          else:
22              print("Schluessel nicht lieferbar")
23      else:
24          print("Nicht Authorisiert")
```

Listing 5.7: Bearbeitung einer Schlüsselanfrage durch den Schlüsselmanager

Methode beantrageSchluesselBeiQKD

Die unterste Ebene des Bearbeitungsprozess stellt die Klasse *Schlüsselmanager* mit, in diesem Fall, dem Methodenaufruf *beantrageSchluesselBeiQKD()* dar.
Als Erstes wird ein Hilfsvariable mit dem beantragenden Master sowie den Slaves, welche später den oder dieselben Schlüssel nutzen können, erstellt. Die Beantragung eines Schlüssels beim QKD-Gerät geschieht per get-Request, wozu der benötigte Link so angepasst wird, dass der für die vorgesehene Kommunikation passende Schlüssel bezogen werden kann. Mit dem nun angepassten https-Link kann nun sowohl der Schlüssel als auch die passende Schlüssel-ID bezogen werden und in der Hilfsvariable result zwischengespeichert werden. Aus dieser wird nun das JSON-Objekt, welches das Schlüsselpaar enthält, extrahiert, um daraus eine Instanz der Klasse *Schluesselmaterial* erstellen zu können, welche später als Antwort an die nächst höhere Bearbeitungsebene weitergeleitet wird.
Allerdings wird zunächst das Setzen der vorgesehenen Time-to-Live für den Schlüssel gesetzt, in Rahmen der Umsetzung 15 Sekunden. Die Verwaltung der Schlüssel geschieht durch die Variable listeSchluessel vom Typ List, in der sich Objekte der Klasse *Schluessellistenobjekt* befinden. Das Nutzen dieser Klasse vereinfacht das Verwalten der gesamten Schlüsselmenge und beinhaltet folgende Parameter:

- Schlüsselpaar vom Typ Schluesselmaterial
- Time-to-Live vom Typ Float
- Angabe ob sich der Schlüssel aktuelle in Benutzung befindet vom Typ Bool
- Liste von den Schlüsselnutzern mit Master und Slaves vom Typ JSON

Damit die Schlüsselmanagementanwendungen der Slave-Anwendungen ebenfalls die benötigten Schlüssel beziehen können, werden im nächsten Schritt allen Slaves die Schlüssel-IDs der verwendeten Schlüssel mitgeteilt. Dazu wird per while-Schleife für jeden Slave eine entsprechende Synchronisationsnachricht an die Synchronisationseinheit weitergeleitet, damit eine Anpassung der Schlüsselmenge stattfinden kann. Als letztes wird ein Timer gestartet, um zu bestimmen, wann die Time-to-Live eines Schlüssels abgelaufen ist, damit dieser nicht weiterverwendet wird.

```
1  def beantrageSchluesselBeiQKD(self, sender, master, komPartner):
2      listeBenutzer = {"Master": master, "Slaves": komPartner}
3      httpsLink = 'https://127.0.0.1:5000/api/v1/keys/ETSI{}/enc_keys'
4      if str(sender) in "Anwendung 1":
5          httpsLinkFertig = httpsLink.format("B")
6      elif str(sender) in "Anwendung 2":
7          httpsLinkFertig = httpsLink.format("A")
8      else:
9          print("Fehler link")
10     result = requests.get(httpsLinkFertig, verify=False)
11     key = result.json()
12     newKeyMaterial = Schluesselmaterial(key["keys"][0]["key_ID"], key["
           keys"][0]["key"])
13     TTL = 15
14     self.fuegeSchluesselListeHinzu(Schluessellistenobjekt(
           newKeyMaterial, TTL, True, listeBenutzer))
15     i = 0
16     while i < len(komPartner):
17         synNachricht = {"Protokoll": "ETSI014", "Typ": 4, "SynTyp" : 0,
               "Menge": 1, "Schl ssel-ID": newKeyMaterial.getID(), "
               Master": master,
18                         "Slaves": [komPartner[i]]}
19         self.KMS.getSynStelle().leiteNeueSchl sselmengeWeiterExtern(
               synNachricht, komPartner[i])
20         i = i + 1
21     self.actKey = newKeyMaterial.getID()
22     t = Timer(interval=TTL, function=self.timerTTL).start()
23     return newKeyMaterial
```

Listing 5.8: Methode um ein Schlüsselpaar (Schlüssel und ID) zu beziehen

5.2.3 Übersicht der Implementierungsergebnisse

An dieser beispielhaften Beschreibung von Teilen des implementierten Codes ist es möglich ebenfalls die Funktionsweise des restlichen Codes herzuleiten. Die anfänglichen Bearbeitungsschritte durch die Methoden *bearbeiteNachricht()* und *bearbeiteAnfrage()* oder simultan *bearbeiteSynchronisationsnachricht()* bzw. *bearbeiteAuthentifizierungsnachricht()* sind jedes Mal gleich.
Während der Beschreibung wurde teilweise von verschiedenen Bearbeitungsebenen gesprochen. Hierbei handelt es sich um die unterschiedlichen Schichten des implementierten Prototypens, welche anhand eines Schichtenmodells auf der Abbildung 5.1 zu sehen sind. Diese Abbildung ist ebenfalls unterstützend für die Herleitung der Funktionsweise des restlichen Codes.

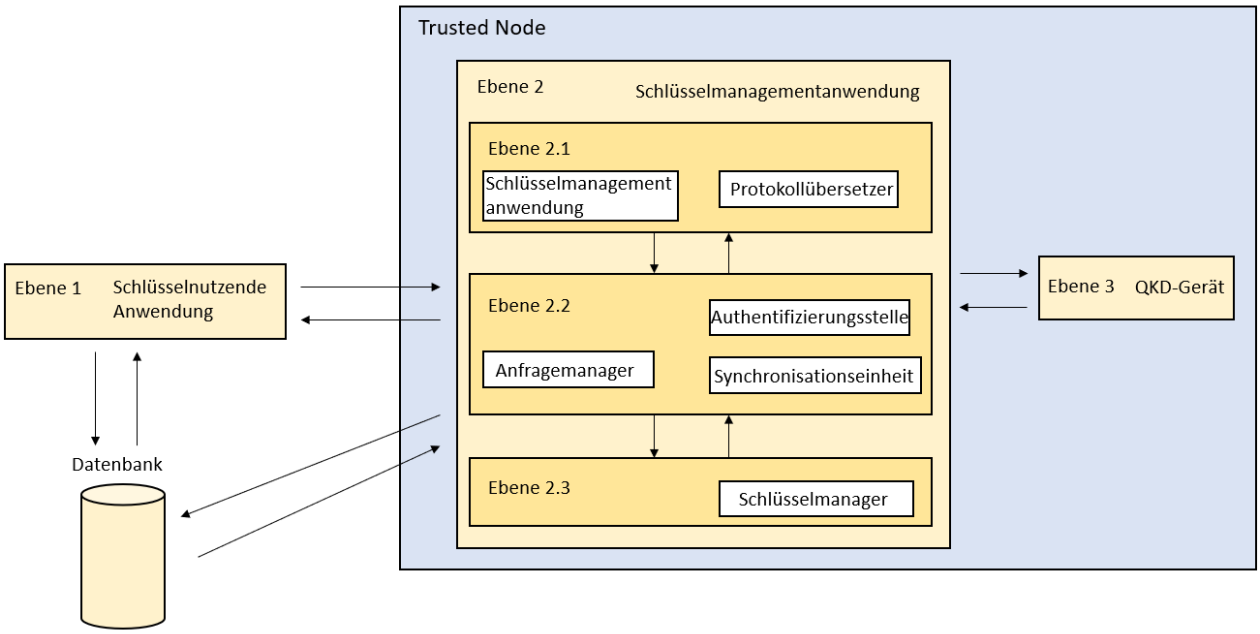


Abbildung 5.1: Darstellung des Modells für die Schichtenarchitektur des Prototypens

Dieses Modell soll nicht die Interaktion zwischen den einzelnen Komponenten zeigen, sondern zwischen den einzelnen Ebenen/Schichten aufzeigen. Hierbei steht besonders die Struktur innerhalb der Schlüsselmanagementanwendung während der Bearbeitung einer Nachricht im Fokus.
Für einen besseren Überblick welche Anforderungen wie umgesetzt worden sind, wurde

eine entsprechende Auflistung vorgenommen. Dieser kann entnommen werden ob und in welchem Umfang die entsprechende Anforderung umgesetzt worden ist. Zur Angabe der Umsetzung sind folgende Symbole benutzt worden:

- ✓ : Umgesetzung
- (✓): Rudimentäre Umsetzung
- × : Keine Umsetzung

Die Beschreibung des Umsetzungsumfanges einer Anforderung befindet sich in kurzer Fassung in der Kommentarspalte.

Tabelle 5.1: Abgleich der Implementierung mit dem Anforderungskatalog

Anforderung	**Nr.**	**Umsetzung**	**Kommentar**
Protokollunabhängige Schnittstellen	F_1	(✓)	Rudimentär umgesetzt, da nur mit dem ETSI 014-Protokoll gearbeitet worden ist
Authentifizierung	F_2	✓	Entsprechende Authentifizierungsnachrichten werden einfach weitergeleitet
	F_3	✓	Einfache Authentifizierung anhand einer kurzen Nachricht.
Eindeutige Identifikation	F_4	✓	Jede Instanz einer Schlüsselmanagementanwendung oder Synchronisationeinheit hat eine eindeutige ID
Suchfunktion für Endpunkte	F_5	×	Eine konkrete Suchfunktion wurde nicht umgesetzt, sondern eine Übergangslösung durch das Nutzen einer lokalen Datenbank genutzt
Anfragenmanagement	F_6	(✓)	Das Bearbeiten einer Anfrage wurde bis auf den Punkt der Priorisierung (da zeitgleich nur eine Anfrage) umgesetzt

Umgang mit Schlüsselmaterial	F_7	✓	Vollständig umgesetzt
Umgang mit Schlüsselmaterial	F_8	✓	Umgesetzt anhand einer Verwaltungsliste
	F_9	✓	Einfach Umsetzung anhand einer Durchsuchung der Verwaltungsliste
	F_{10}	(✓)	Grundlegende Implementierung mit einer einfachen Abfrage, ob angefragte Menge lieferbar ist und einer Konsolenausgabe wenn nicht
	F_{11}	✓	Durch das Nutzen der Verwaltungsliste gelöst
	F_{12}	✓	Durch das Nutzen der Verwaltungsliste gelöst
	F_{13}	×	Da in dem Szenario keine Angriffe oder unsichere Leitungen vorherrschten, wurde dieser Punkt nicht weiter beachtet
Umgang mit Schlüssel	F_{14}	(✓)	Implementierung anhand eines einfach Timers, welcher im Hintergrund läuft und nach Ablauf der Zeit den Schlüssel löscht und die Schlüsselmenge synchronisiert

Fortlaufende Synchronisation	F_{15}	✓	Implementiert, sodass bei Bezug eines Schlüssels durch den Master die Slaves die IDs mitgeteilt bekommen; ansonsten eine Synchronisation wie bei F_{14}
Protokollbasiertes Fehlermanagement	F_{16}	(✓)	Einfache Reaktionen auf einige Fehler durch Konsolenausgaben
	F_{17}	(✓)	Konsolenausgabe der Fehler/-der Fehlerkategorien
Optisches	NF_1 - NF_5	×	Alle nicht-funktionalen Anforderungen sind im Rahmen dieses einfachen Prototypens nicht umgesetzt worden

Außerdem wurde eine Testumgebung mit einer kleinen Demo-Anwendung implementiert, um den Prototypen testen zu können. Die Testungen wurden nicht einem realen QKD-Gerät sondern an einer Imitation durchgeführt, allerdings sind trotzdem HTTPS-Requests genutzt worden, um Schlüsselmaterial anzufordern. Für das Simulieren einer Netzstruktur sind einfache Sockets mit Threads genutzt, sodass es möglich war, dass eine Demo-Anwendung einen Schlüssel anfordern, eine Nachricht verschlüsseln und an eine andere Demo-Anwendung senden konnte. Somit sind innerhalb der Testumgebung zwei Schlüsselmanagementanwendungen und zwei Demo-Anwendungen erzeugt worden und folgende Funktionen werden selbstsändig nacheinander ausgeführt:

- Schlüsselnaforderung als Master (Anwendung 1)
- Schlüsselbezug als Slave (Anwendung 2)
- Verschlüsselung einer Nachricht und das Senden des verschlüsselen Textes an Anwendung 2
- Empfang und Entschlüsselung der Nachricht
- Zur Kontrolle wird der verschlüsselte und entschlüsselte Text sowie der Originaltext in der Konsole ausgegeben

Die gesamte Testung lief auf einem Rechner und als IP-Adresse wurde jedes mal 127.0.0.1 genutzt.

Kapitel 6

Diskussion der Ergebnisse

Auf Grundlage der vorherigen Kapitel soll sowohl die Sicherheit des Protokolls ETSI014 diskutiert werden als auch die des implementierten Prototypens. Außerdem soll dessen Zweckmäßigkeit in Bezug auf repräsentative Aussagekraft für den vorher erstellten Anforderungskatalog sowie die Klassen- und Sequenzdiagramme beurteilt werden.
Zu diesem Zweck werden zunächst die unterschiedlichen Sicherheitsaspekte des Protokolls ETSI014, welches als Grundlage für die Entwicklung eines Schlüsselmanagementprogrammes und der Implementierung eines entsprechenden Prototypens diente, näher betrachtet. Hieraus leiten sich die Argumente für die darauffolgende Beurteilung der Sicherheit dieses Protokolls innerhalb des Anwendungsrahmen. Darauf folgt eine Untersuchung des implementierten Prototypens auf dessen Zweckmäßigkeit die vorher stattgefundene Softwareentwicklung repräsentieren und als Beleg für dessen Umsetzbarkeit dienen zu können. Am Schluss auch der Prototyp einer kritischen Sicherheitsüberprüfung unterzogen, um auf Sicherheitslücken hinzuweisen, welche bei einer kompletten Umsetzung beachten werden müssen, und diese Lücken in den Kontext des Ziels der Implementierung zu setzen.

6.1 Kritische Sicherheitsüberprüfung des verwendeten Protokolls

Das ETSI-Protokoll 014 behandelt einige sicherheitsrelevante Aspekte nicht, da sie außerhalb des vom Protokoll gesetzten Rahmens liegen. Dies wäre beispielsweise die Schlüsselverwaltung sowie die sichere Übertragung von Schlüsselmaterial zwischen Schlüsselmanagementanwendung und schlüsselnutzende Anwendung. Unter anderem lagen deshalb diese beiden Aspekte während der Erarbeitung eines Softwareentwurfs für eine Schlüssel-

managementanwendung im Fokus. Dies schränkt den sicherheitsrelevanten Bereich, welchen ETSI 014 abdecken soll, erheblich ein und muss bei einer kritischen Sicherheitsüberprüfung beachtet werden. Dennoch gibt das Protokoll ETSI 014 einige Maßnahmen vor, um die Kommunikation sicherer zu gestalten.
Eine vom Standard vorgegebene Maßnahme ist, dass HTTPS-Protokolle (mit einer TLS Version 1.2 oder höher) für den Datenaustausch zwischen einer schlüsselnutzenden Anwendung (Secruity Application Entity) und einer Schlüsselmanagementanwendung (Key Management Entity) genutzt werden soll. Somit ist diese Kommunikation potenziell genauso sicher wie die innerhalb des World Wide Webs, welche ebenfalls auf HTTPS basiert.
Auf der anderen Seite nimmt ETSI 014 an, dass jeder innerhalb eines Netzes agierende Knoten als sicher gilt. Dieser Zustand muss sich bei bloß einem Knoten ändern oder dieser fälschlicherweise als solche in das Netz eingefügt werden und einem potentiellen Angreifer bietet sich eine relevante Sicherheitslücke.
Zusammenfassend lässt sich sagen, dass eine entworfene Schlüsselmanagementanwendung, welche diesen Standard nutzt, dadurch nicht automatisch sicher ist. Denn es müssen, wie schon angesprochen, Lösungen für die nicht berücksichtigten Sicherheitsaspekte gefunden werden. Dies gilt insbesondere für eine Weiterentwicklung des Prototypens oder eine vollumfängliche Implementation einer solche Anwendung bei Nutzung des ETSI 014-Standards.
[32]

6.2 Beurteilung des Prototypen auf dessen Zweckmäßigkeit

Von vorne herein war es nicht vorgesehen ein Programm zu implementieren, welches alle Funktionen vollumfänglich im Sinne des Entwurfes bereitstellt, sondern sollten nur die wichtigsten Funktionalitäten anwendbar sein. Ziel war es eine funktionstüchtige Umsetzbarkeit des vorher durchgeführten Softwareentwurfes zeigen zu können. Ob diese Zielsetzung am Ende erreicht worden ist, soll nun aufgearbeitet und beurteilt werden.

6.2.1 Umsetzungstiefe des Softwareentwurfes

Wie einleitend schon erwähnt sollten nur die wichtigsten Funktionen, wie beispielsweise das Beziehen, Verwalten und Synchronisieren einer Schlüsselmenge, vollumfänglich im Sinne des Entwurfs umgesetzt werden. Dies wurde während der Implementierung des Prototypens, wie in 5.2.2 beschrieben, erreicht und lässt sich somit in dem gegebenen Sze-

nario anwenden.
Andere Funktionalitäten, wie ein Suchalgorithmus für die Pfadfindung zu einem Knoten, dessen Position unbekannt ist, wurden aufgrund ihrer Irrelevanz nicht implementiert, sondern nur als leere, funktionslose Hülle eingebunden. Grund hierfür war Kontext einer 1:1-Kommunikation ohne Zwischenknoten. Wiederum andere Funktionen, wie die vorgesehene Authentifizierungsstelle, wurden nur rudimentär umgesetzt, da in dem gegebenen Szenario kein vollumfänglicher Authentifizierungsprozess notwendig war. Dies liegt daran, dass beispielsweise keine Angriffe von Eve während der Testungen vorgesehen waren. Ebenfalls wurde auch nicht-funktionale Anforderungen an eine Schlüsselmanagementanwendung nicht weiter berücksichtigt und es bei Ausgaben innerhalb der Konsole belassen, da zweckdienlich war und ansonsten einen unverhältnismäßigen Mehraufwand bedeutet hätte.
In Bezug auf die Umsetzungstiefe der in den Kapiteln 3 und 4 erarbeiteten Softwareentwicklung lässt sich zusammenfassend sagen, dass der Prototyp das Ziel der Zweckmäßigkeit erfüllt hat. Denn die wichtigsten funktionellen Anforderungen sind erfüllt worden und lassen sich demonstrieren. Einzige Anmerkung hierbei ist, dass die Prozesse innerhalb der Schlüsselmanagementanwendung noch besser parallelisiert werden könnten, was besonders während eines Szenarios mit mehr Aktivitäten sinnvoll wäre.

6.2.2 Verwendung von Sockets mit zur Hilfenahme von Threads

Damit sowohl die einzelnen Instanzen einer Testanwendungsklasse sowie des Prototypens einer Schlüsselmanagementanwendung sowohl innerhalb des eigenen als auch mit externen Knoten kommunizieren können, wurde diese Kommunikation durch die Verwendung von Sockets, welche in 5.2.1 beschrieben worden ist, realisiert. Damit die Socket-Server im Hintergrund auf das Empfangen einer Nachricht warten können, wurden Threads mit eingebunden. Beides sollte mit möglichst wenig Umfang umgesetzt werden und gleichzeitig eine stabile Kommunikation ermöglichen.
Die Anwendung von Threads ist in dem Kontext der Zielvorgaben bezüglich der Implementierung eines Prototypens zweckmäßig und erfüllt gesetzten Anforderungen. Die Socket-Server sind in zwar in der Lage mehrere Verbindungen nacheinander anzunehmen, sodass nicht nach jeder Kommunikation mit einem Client der Server komplett neu gestartet werden muss. Allerdings können nicht mehrere Nachrichten gleichzeitig empfangen und für die weitere Verarbeitung aufbereitet werden. Dies wäre zweckmäßiger, weil beispielsweise nicht zwei Socket-Server für eine Schnittstelle benötigt werden, sodass zeitgleich sowohl das Beziehen eines Schlüssels als auch das Empfangen einer verschlüsselten Nachricht möglich sind. Ansonsten erfüllt auch die Verwendung von Sockets ihren Zweck und ermöglicht eine stetige Kommunikation.

6.2.3 Einbindung einer lokalen Datenbank

Um das Problem bezüglich der Ermittlung der IP-Adressen und Ports der jeweiligen Schnittstellen zu lösen, wurde eine lokale Datenbank von mongoDB wie in 5.2.1 eingebunden. In dieser sind alle Schnittstellen mit entsprechender IP-Adresse, Port sowie einer eindeutigen ID hinterlegt. Dies stellte eine einfache sowie zweckmäßige Lösung dar und reichte aus, um die Funktionsweise der grundlegenden Funktionen des Prototypens demonstrieren zu können. Eine weitere Möglichkeit wäre unteranderem eine verstärkte Umsetzung der Suchfunktion, welche alle Nachbarknoten mit der jeweiligen Möglichkeit diese anzusprechen liefern könnte.
Aber in diesem begrenztem Anwendungsszenario war keine Notwendigkeit für einen solchen Schritt gegeben, weshalb auf die einfachere Lösung zurückgegriffen worden ist. Bei einem größeren Kontext, wie beispielsweise die Kommunikation über Zwischenknoten, sollte der aufwendigere Ansatz umgesetzt werden, da dieser in diesem Fall zweckmäßiger wäre.

6.2.4 Zusammenfassende Beurteilung der Zweckmäßigkeit des Prototypens

Zusammenfassend lässt sich über die Betrachtung dreier verschiedener Aspekte der Implementierung eines Prototypens auf seine Zweckmäßigkeit sagen, dass das Ziel mit wenigen unbedeutenden Abstrichen erreicht worden ist. Der Prototyp ist in dem gegebenen Anwendungsszenario in der Lage die grundlegenden funktionellen Anforderungen an eine Schlüsselmanagementanwendung zu erfüllen und diese auch zu demonstrieren.

6.3 Kritische Sicherheitsüberprüfung des Prototypen

In diesem Abschnitt soll die Sicherheit des implementierten Prototypens näher betrachtet sowie beurteilt werden. Ziel hierbei ist es insbesondere auf bestehende, aufgrund des Kontextes der Implementierung auftretende, Sicherheitslücken zu benennen, damit diese in einer späteren Fortentwicklung geschlossen werden können. Hauptgrund für die meisten Risikofaktoren in Bezug auf die Sicherheit ist, dass keine Angriffe seitens Eves sowie die Verbindung zwischen Testanwendung und Schlüsselmanagementanwendung als sicher angenommen worden sind.

6.3.1 Sicherheitsüberprüfung im Allgemeinen

Wie schon einleitend erwähnt hat der Prototyp aufgrund des vorgegebenem Anwendungsszenario mehrere Sicherheitslücken. Diese werden nach folgender Methodik herausgearbeitet, wobei kein Anspruch auf Vollständigkeit besteht: Das Schichtenmodell aus der Abbildung 5.1 wird von links bzw. oben beginnend schichtweise mit dem Ziel analysiert Sicherheitslücken aufzudecken und zu beschreiben.
So werden die Schlüssel-IDs unchiffriert über eine potentiell abhörbare Leitung verschickt, wodurch jederzeit diese abgefangen und für weitere Angriffe verwenden könnte. Verstärkend kommt noch ein rudimentäres Authentifizierungssystem hinzu, aufgrund dessen Eve theoretisch in der Lage wäre den zur Schlüssel-ID passenden Schlüssel beziehen zu können. Deshalb müssen besonders an dieser Stelle bei einer Weiterentwicklung des Prototypens geeignete Lösungen erarbeitet und umgesetzt werden. So wäre eine verschlüsselte Übertragung der IDs ein Ansatz sowie ein vollumfängliches Authentifizierungssystem. Außerdem muss eine Lösung für den Übertragungsweg schlüsselbeziehende Anwendung - Schlüsselmanagementanwendung gefunden werden, da es sich hierbei um wahrscheinlich die für die Sicherheit bedeutendste Lücke handelt.
Ein weiterer Aspekt ist die Schnittstelle Schlüsselmanagementanwendung - QKD-Gerät. Diese kann bei den meisten Szenarien, welche anhand eines Prototypens abgebildet werden sollen, als sicher angesehen werden. Aber in echten Anwendungsfällen ist diese Gegebenheit nicht zwangsläufig gegeben und entsprechende Lösungsansätze müssen entwickelt werden. Ebenfalls muss bei einer Weiterentwicklung des Prototypens sichergestellt werden, dass nur autorisierte Schlüsselmanagementanwendungen Schlüssel von einem QKD-Gerät beziehen können.
Außerdem müssen die Zugriffsregeln die unterschiedlichen Schichten/Ebenen innerhalb der Schlüsselmanagementanwendung geregelt werden. Ansonsten wäre es anderen Schichten unerlaubterweise möglich auf die Inhalte anderer zuzugreifen, was ebenfalls eine potenzielle Gefahrenquelle darstellt. Außerdem muss die Anwendung, außer über die gegebenen Schnittstellen, für Zugriffe von außen geschützt werden.
Zusammenfassend lässt sich sagen, dass derartige Sicherheitslücken in dem Szenario, welches der Prototyp bedienen sollte, vernachlässigbar sind. Dennoch ändert sich dies mit einer Weiterentwicklung des bisherigen Entwicklungsstandes. Weitere Sicherheitslücken, welche die Nutzung von Sockets und (lokalen) Datenbanken entstehen können, werden in den folgenden zwei Paragrafen näher betrachtet.

6.3.2 Nutzungsweise von Sockets innerhalb der Implementierung

Sockets sind grundlegend nicht unsicher, da Möglichkeiten bestehen eine Kommunikation durch Sockets sicher zu gestalten. Beispielsweise wäre dies durch die Verwendung

vom Secure Sockets Layer (SSL) möglich, um Nachrichten bei einer Übertragung schützen zu können. Allerdings wurden diese bei der Implementierung des Prototypens nur so grundlegend umgesetzt, dass ein stabiles Kommunikationsnetz sichergestellt ist und kein Sicherheitsprotokolle dabei berücksichtigt.
Aus diesem Grund tut sich hier Sicherheitslücke auf, welche so ausgenutzt werden kann, dass während des Datenverkehrs im aufgebauten Netz die Nachrichten mitgelesen werden können. Verstärken tut diesen Effekt, das im vorherigen Absatz angesprochene, nur rudimentär implementierte Authentifizierungssystem. Auch an dieser Stelle kann zusammenfassend gesagt werden, dass diese Sicherheitslücke bei einer Weiterentwicklung des Prototypens für größere Szenarien geschlossen werden muss. Entweder durch die Einpflegung des SSL-Protokolls oder ein Verzicht von Sockets als Lösung, um eine Kommunikation zu gewährleisten.

6.3.3 Einsatz einer lokale Datenbank für Netzdaten

Das Nutzen einer Datenbank birgt neben unterschiedlichen Vorteilen auch potenzielle Risikofaktoren. Je nach Relevanz der dort hinterlegten Daten kann die Gefahr, welche von einer Sicherheitslücke bezüglich des Zugriffsrechts ausgeht, gering oder hoch sein. Da der Zugriff auf die während der Implementation eingebundene Datenbank nicht passwortgeschützt ist, ist es theoretisch jeder Stelle im Netz möglich jederzeit auf den Inhalt der Datenbank zuzugreifen.
In Szenarien, in denen Angriffe nicht ausgeschlossen und gegebene Verbindungen als sicher angesehen werden, müssen bei der Verwendung einer Datenbank der Zugriff auf die dort hinterlegten Daten sicher gestaltet werden. Derartige Angriffe wurden während der Implementation des Prototypens als nicht anzunehmend behandelt und diese Lücke entsprechend nicht geschlossen. Auch sollten bei einer Weiterentwicklung die angesprochenen Punkte, welche während der Beurteilung bezüglich der Zweckmäßigkeit des Prototypens aufgegriffen worden sind, beachtet und je nach Szenario umgesetzt werden.

Kapitel 7

Zusammenfassung und Ausblick

Im folgenden Kapitel werden noch einmal die wichtigsten Aussagen und Erkenntnisse dieser Arbeit zusammengefasst sowie dazu Stellung genommen. Am Ende wird ein Ausblick auf die Möglichkeiten bei einer tiefergehenderen Implementation für einen umfangreicheren Prototypen sowie einem vollumfänglichen Programm gewährt.

7.1 Zusammenfassung

Zunächst wurden Begriffsdefinitionen vorgestellt, um eine grundlegende Wissensbasis für das Verständnis der folgenden Kapitel zu schaffen, sodass besonders die Diskussion der Ergebnisse dieser Arbeit nachvollzogen werden kann. Um den thematischen Rahmen, in welchem diese Masterarbeit entstanden ist, zu erläutern, ist das Projekt MuQuaNet im Wesentlichen kurz vorgestellt worden. Als erste technische Grundlage sind auf einer allgemeinen Erklärung der Kryptographie die drei unterschiedlichen Verfahren ((a-)symmetrische und hybride Verschlüsselung) sowie die Quantenkryptographie erklärt worden. Darauffolgend sind die Funktionsweise und Ziele eines Quantenschlüsselaustausches ausgearbeitet worden, indem anschließend beispielhaft das Protokoll BB84, welches einen Quantenschlüsselaustausch zum Ziel hat, erläutert worden ist. Der Schwerpunkt des Grundlagen Kapitels lag auf den ETSI-Protokollen 004 und 014, wozu unteranderem zum besseren Verständnis REST-API, JSON, HTTPS-Protokolle sowie das TLS-Protokoll erklärt worden sind. Denn die Wahl des Protokolls war entscheidend für die Struktur sowohl des späteren Softwareentwurfes als auch für den implementierten Prototypen. Für den späteren Programmentwurf wurde das Protokoll ETSI 014 gewählt, da sowohl schon die entsprechenden Geräte vorhanden sind als auch eine einfachere Implementierung durch das Nutzen von Webservices ermöglicht wird.
Auf dieser Wissensbasis aufbauend ist eine Anforderungsanalyse für eine Schlüsselma-

nagementanwendung im Rahmen eines Quantenkommunikationnetzes durchgeführt worden. Das Ergebnis war ein Anforderungskatalog mit mehreren funktionalen sowie nichtfunktionalen Anforderungen für eine derartige Anwendung. Wobei die funktionalen Anforderungen Authentifizierungsprozesse, den Umgang mit Schlüsselmaterial, protokollbasiertes Fehlermanagement und Ähnliches umfassen. Nicht-funktionale Anforderungen zielen bespielsweise auf eine angepasste Benutzeroberfläche und die Bedienbarkeit des Programmes ab.
Hierauf aufbauend ist ein Programmentwurf erarbeitet worden, indem zunächst die benötigten Komponenten anhand des Kataloges hergeleitet worden sind und durch ein stark vereinfachtes Komponentendiagramm auf der Abbildung 4.1 in ihrem Zusammenspiel erklärt worden sind. Diese bildeten die Grundlage für Klassendiagramm der Schlüsselmanagementanwendung, um die notwendigen Klassen und deren Beziehungen zueinander abbilden zu können und sich mit den vorher erarbeitetn Komponenten decken. Während dieses Prozesses sind die folgenden Klassen identifiziert worden:

- Schlüsselmanagementanwendung: Schnittstelle und übergeordnete Managementeinheit, welche beispielsweise den gesamten Bearbeitungsprozess einer Anfrage leitet
- Protokollübersetzer: übersetze Nachricht in ein nutzbares Format, falls der Kommununikationspartner ein fremdes Protokoll benutzt
- Authentifizierungsstelle: kann Authentifizierungen von anderen Knoten fordern, verwaltet ein Netz von Trusted Nodes und leitet Authentifizierungsnachrichten weiter
- Suchfunktion: sucht anhand eines Routingverfahrens die kürzeste Strecke zu einem unbekannten Knoten
- Anfragemanager: bearbeitet alle Anfragetypen
- Schlüsselmanager: bezieht von einem QKD-Gerät Schlüsselmaterial und verwaltet dieses
- Synchronisationseinheit: sorgt dafür, dass die Schlüsselmenge bei allen Kommunikationspartnern gleich bleibt

Mit Hilfe dieser vorgegebenen Struktur ist ein Prototyp implementiert worden, wobei das Ziel die Bereitstellung der grundlegenden Funktionalitäten war, sodass manche Funktionen nur stark vereinfacht verwirklicht worden sind. Im Rahmen dieser Umsetzung sind keine Angriffe auf die Kommunikation, die Schlüsselmanagementanwendung und ähnliche Strukturen angenommen bzw. erwartet worden, weshalb zusätzlich kein Angreifer Eve implementiert worden ist. Es wurde die Schlüsselmanagementanwendung und eine Demo-Anwendung implementiert, um eine einfache 1:1-Kommunikation auf einem lokalen Rechner durchführen zu können.
Daran anschließend sind die Ergebnisse, Sicherheit des verwendeten Protokolls ETSI 014 sowie die Sicherheit und Zweckmäßigkeit des Prototypens diskutiert worden. Aus der Be-

trachtung dieser drei Aspekte ergab sich, dass das verwendete Protokoll ETSI 014 nur wenige Vorgaben der Sicherheit vorschreibt, sondern die Lösung der meisten Probleme den Nutzern und Entwicklern überlässt. Dies hat nochmals die Notwendigkeit unterstrichen, dass entsprechende Ansätze im Rahmen dieser Masterarbeit entwickelt worden sind. Ansonsten sind Sicherheitslücken des Prototypens mit den Hinweisen benannt worden, dass diese im gegebenen Szenario nicht von Relevanz waren. Denn es wurde von keinen Angriffen auf den Prozess oder das System ausgegangen, da nur die grundlegende Funktionsweise aufgezeigt werden sollte. Zusätzlich wurde der Prototyp für die an ihn gestellten Aufgaben als zweckmäßig bewertet. Für die Abdeckung anderer, umfangreicherer Szenarien ist eine Erweiterung von Funktionen allerdings notwendig.

7.2 Ausblick und weiterführende Arbeiten

Zum Schluss soll sowohl ein Ausblick bezüglich weitergehender Funktionen gewährt werden, welche durch eine Weiterentwicklung des Prototypens möglich sind. Außerdem Fragestellungen für weiterführende Arbeiten aufgezeigt, welche sich während der Beschäftigung mit dem Thema einer Schlüsselmanagementanwendung im Rahmen einen Quantenkommunikationsnetzes ergeben haben.

7.2.1 Ausblick

Ein Teil, den diese Arbeit nicht abdeckt, ist die Implementation einer vollumfänglichen Schlüsselmanagementanwendung oder alternativ eines umfangreicheren Prototypens. Deshalb soll ein kurzer Ausblick gewährt werden, welche Erweiterungen sinnvollerweise als nächstes umgesetzt werden sollten. Wobei diese Empfehlung von den geplanten Szenarien, welche abgedeckt werden sollen, abhängig ist. Im folgenden Abschnitt werden vier mögliche Weiterentwicklungen mit dem entsprechenden Kontext vorgestellt.

Anschluss an ein QKD-Gerät

Wenn im nächsten Schritt geplant ist, dass der Prototyp mit einem echten QKD-Gerät kommunizieren soll, dann muss die Schnittstelle KMS - QKD-Gerät im ersten Schritt angepasst werden, falls diese nicht funktionsfähig ist. Diese Anpassung sollte schnell umgesetzt sein, da der jetzige Prototyp schon mit HTTPS-Requests arbeitet und jediglich das Nutzen der entsprechenden Zertifikate aussteht.

Beliebig große Schlüsselmenge

Durch den Anschluss ein ein QKD-Gerät ist es möglich mehrere Schlüssel zu beziehen, sodass die Testung des Schlüsselmanages besser überprüft und notfalls entsprechend angepasst werden kann. Somit wäre eine Weiterentwicklung bzw. Verbesserung der Abläufe in der Klasse *Schlüsselmanager* möglich und im Rahmen eines Anschlusses an ein QKD-Gerät zu empfehlen.

Komplexere Netze mit n Knoten

Wenn komplexere Netze mit n Knoten simuliert werden soll, sodass auch über Zwischenknoten kommuniziert werden soll, muss zwingend erforderlich die Komponente *Suchfunktion* vollständig implementiert werden. Ebenfalls muss das Weiterleiten von Nachrichten funkionieren, was bisher nicht von Nöten gewesen ist. Die Klasse *Authentifizierungsstelle* muss weiterentwickelt werden, wenn mit Angriffen auf die Kommunikation bzw. das Netz gerechnet wird, damit gleichzeitig die Schaffung eines Netzes aus Trusted Nodes möglich ist.

Simulation von Angriffen

Bei einer Simulation von Angriffen müssen, je nach Komplexität dieser, mehrere Funktionalitäten entwickelt werden. Besonders die Übertragung des Schlüssel von der KMS zu einer anfragenden Stelle liegt hierbei im Fokus und muss durch kryptographische Verfahren abgesichert werden. Außerdem ist eine voll funktionsfähige *Authentifizierungsstelle* notwendig, sodass beispielsweise unberichtigte Stelle keine Schlüssel mit abgefangenen Schlüssel-IDs beantragen können.

7.2.2 Weiterführende Aufgaben

Multiplexverfahren

Wie können Multiplexverfahren für die verschiedenen Dienste, wie der Schlüsselbezug, einer Schlüsselmanagementanwendung implementiert werden?
Dies könnte eine Forschungsfrage mit dem Ziel sein den Softwareentwurf derart zu erweitern, dass ein oder mehrere Multiplexverfahren genutzt werden können. Dies würde die Effizienz, das Leistungspotential und die Anwendbarkeit der entworfenen Schlüsselmanagementanwendung verbessern.

Kryptographische Lösungen

Der bisherige Programmentwurf weißt noch kryptographische Probleme in der Form auf, dass u.a. eine Lösung für die post-quantensichere Kommunikation zwischen einer schlüsselnutzenden Anwendung und der Schlüsselmanagementanwendung noch nicht vorhanden ist. An dieser Stelle bietet sich die Möglichkeit für ein eigenständiges Forschungsthema zur Entwicklung einer oder mehrere entsprechenden Lösungen für dieses kryptographische Problem.
Ein weiteres Thema in diesem Kontext wäre die Entwicklung einer Lösung für die sichere Übertragung der Schlüssel über die Schnittstelle Schlüsselmanagementanwendung - QKD-Gerät. Denn dies birgt ein weiteres kryptographisches Problem sowie einen potenziellen Angriffspunkt.

Hybride Kommunikation

Eine weitere Herausforderung, welche sich bei der Planung der Kommunikation herausstellte, ist die Kompatibilität einer Schlüsselmanagementanwendung mit sowohl homogenen als auch hybriden Kommunikationskanälen. So muss es möglich sein, Nachrichten, beispielsweise mit Schlüsselmaterial, nicht nur über klassische oder quantenbasierte Kommunikationskanäle zu versenden. Denn auf einer Kommunikationsstrecke, ob über mehrere Zwischenknoten oder per Direktverbindung, kann es zu einem Wechsel der Kanäle zwischen klassisch und quantenbasiert kommen. Trotzdem muss sichergestellt werden, dass die Nachrichten ohne Datenverlust oder ähnliche Probleme zwischen zwei oder mehreren Kommunikationspartnern versendet werden können.

Schlusswort

Zum Schluss lässt sich somit über die Ergebnisse dieser Masterarbeit die Aussage treffen, dass der implementierte Prototyp die Erwartungen erfüllt. Außerdem bietet der erstellte Anforderungskatalog eine gute Grundlage den Prototypen sowohl weiter auszubauen als auch in ein echtes QKD-Netz integrieren zu können. Mit den in Kapitel 7 vorgestellten nächsten Entwicklungsschritte sowie fortführende Forschungsfragen kann eine Schlüsselmanagementanwendung für komplexere Netzstrukturen sowie realitätsnahe Anwendungsfälle realisiert werden.

Literaturverzeichnis

[1] Arfaoui Ghada, Bultel Xavier, Fouque Pierre-Alain Nedelcu Adina Onete Cristina: *The privacy of the TLS 1.3 protocol*. Proceedings on Privacy Enhancing Technologies, Seiten 190–210, 2019.

[2] Ayuso Vicente Martin, Mkinsi Hakim. 2020.

[3] Badach Anatol, Schulte Heinz: *Transport Layer Security*. https://www.researchgate.net/profile/Anatol-Badach/publication/292995131_TLS_-_Transport_Layer_Security/links/56b4e18b08ae44bb33055fb5/TLS-Transport-Layer-Security.pdf. Eingesehen am 02.04.21.

[4] Christian, Kanhäuser: *Unified Modeling Language (UML) - Eine Einführung*. https://ti.tuwien.ac.at/ecs/teaching/courses/wissarb/docs/uml_einfuehrung.pdf. Eingesehen am 16.04.2021.

[5] computerweekly.com: *Was ist Quantenkryptographie?* https://www.computerweekly.com/de/definition/Quantenkryptographie. Eingesehen am 11.04.2021.

[6] Diamanti Eleni, Lo Hoi-Kwong, Qi Bing Yuan Zhiliang: *Practical challenges in quantum key distribution*, 2016.

[7] Dietmar, Wätjen: *Kryptographie - Grundlagen, Algorithmen, Protokolle*, Seiten 3–4, 49–50, 67–70. Springer-Verlag GmbH, 2018.

[8] Dipl.-Ing. Luber, Stefan: *Was ist Kryptographie?* https://www.security-insider.de/was-ist-kryptographie-a-642288/. Eingesehen am 23.03.2021.

[9] etsi.org: *Quantum Key Distribution (QKD)*. https://www.etsi.org/technologies/quantum-key-distribution. Eingesehen am 08.04.2021.

[10] Felt Adrienne Porter, Barnes Richard, King April Palmer Chris Bentzel Chris Tabriz Parisa: *Measuring HTTPS adoption on the web*. In: *26th USENIX Security Symposium (USENIX Security 17)*, Seiten 1323–1338, 2017.

[11] Gisin Nicolas, Ribordy Grégoire, Tittel Wolfgang Zbinden Hugo: *Quantum cryptography*. Reviews of modern physics, Seiten 147–152, 2002.

[12] Gösta, Fürnkranz: *Vision Quanten-Internet: ultraschnell und hackersicher*, Seiten 165–171. Springer-Verlag GmbH, 2019.

[13] Helmbrecht, Udo: *MuQuaNet – Das Quanten-Internet im Großraum München*. https://dtecbw.de/home/forschung/unibw-m/projekt-muquanet/projekt-muquanet. Eingesehen am 19.03.2021.

[14] .itwissen.info: *QoS (quality of service)*. https://www.itwissen.info/QoS-quality-of-service-Dienstguete.html. Eingesehen am 08.04.2021.

[15] Johannes, Buchmann: *Einführung in die Kryptographie*, Seiten 55–57. Springer-Verlag GmbH, 1999.

[16] Kryptowissen.de: *Transport Layer Security (TLS)*. https://www.kryptowissen.de/transport-layer-security-tls.php. Eingesehen am 02.04.2021.

[17] Lo Hoi-Kwong, Curty Marcos, Tamaki Kiyoshi: *Secure quantum key distribution*, 2014.

[18] Luber, Dipl.-Ing. (FH) Stefan: *Was ist HTTPS (Hypertext Transfer Protocol Secure)?* https://www.ip-insider.de/was-ist-https-hypertext-transfer-protocol-secure-a-691192/. Eingesehen am 02.04.2021.

[19] Luber, Dipl.-Ing. (FH) Stefan: *Was ist TLS (Transport Layer Security)?* https://www.security-insider.de/was-ist-tls-transport-layer-security-a-673066/. Eingesehen am 02.04.2021.

[20] Martens Wim, Schmidt Kai Philipp: *Quantenkryptographie*, Seiten 267–270. Springer-Verlag GmbH, 2012.

[21] Martin, Witkowski: *Quantenkryptographie einfach erklärt: Das BB84 Protokoll*. https://itsecblog.de/quantenkryptographie-einfach-erklaert-das-bb84-protokoll/. Eingesehen am 09.04.2021.

[22] Masse, Mark: *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*, Seiten 2–6. O'Reilly Media, Inc., 2011.

[23] Matt Christian, Maurer Ueli: *The one-time pad revisited*. Seite 2706. IEEE, 2013.

[24] McKay Kerry, Cooper David: *Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations*. Technischer Bericht, 2017.

[25] Nils, Gählert: *Quantenkryptographie*, 2013.

[26] Nurhadi Ali Ibnun, Syambas Nana Rachmana: *Quantum key distribution (QKD) protocols: a survey*. Seiten 1–5, 2018.

[27] Owen, Hughes: *Why PyCharm - All the Python tools in one place*. `https://www.jetbrains.com/pycharm/`. Eingesehen am 15.06.2021.

[28] Pezoa Felipe, Reutter Juan L., Suarez Fernando Ugarte Martín Vrgoč Domagoj: *Foundations of JSON schema*. In: *Proceedings of the 25th International Conference on World Wide Web*, Seiten 263–273, 2016.

[29] Scarani Valerio, Bechmann-Pasquinucci Helle, Cerf Nicolas J. Dušek Miloslav Lütkenhaus Norbert Peev Momtchil: *The security of practical quantum key distribution*, 2009.

[30] Schirmer, Kris: *Quantenschlüssel unter der Verwendung von Kerberos*, 2021.

[31] Srocke, M.A. Dirk: *Was ist eine REST API?* `https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-611116/`. Eingesehen am 25.03.2021.

[32] Tanizawa Yoshimichi, Compans Sonia. 2019.

[33] visualstudio.microsoft.com: *PyCharm*. `https://de.wikipedia.org/wiki/PyCharm`. Eingesehen am 15.06.2021.

[34] Wolfgang, Willems: *Codierungstheorie und Kryptographie*, Seiten 67–80. Birkhäuser Verlag AG, 2008.

Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden.

Ferner habe ich vom Merkblatt über die Verwendung von Bachelor/Masterabschlussarbeiten Kenntnis genommen und räume das einfache Nutzungsrecht an meiner Bachelor/-Masterarbeit der Universität der Bundeswehr München ein.

Neubiberg, 30.06.21	Arne Klee
Ort, Datum	Unterschrift