UNIVERSITÄT DUISBURG-ESSEN
FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN
ABTEILUNG INFORMATIK UND ANGEWANDTE KOGNITIONSWISSENSCHAFT

Bachelorarbeit

Visualisierung des BB84 Protokolls in CrypTool 2.0

Benedict Beuscher
Matrikelnummer: 2245950

Abteilung Informatik und Angewandte Kognitionswissenschaft
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen

13. Oktober 2013

Prüfer:
Prof. Dr. Arno Wacker, Universität Kassel
Prof. Dr. Ing. Torben Weis, Universität Duisburg-Essen
Betreuer:
Nils Kopal, M.Sc.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Das Open-Source Programm CrypTool 2.0 [ct2a] wird seit 2007 in Kooperation der Universität Kassel, Duisburg-Essen, Siegen und vielen weiteren Universitäten sowie weiteren Freiwilligen entwickelt. Mit Hinblick auf mehrere unterschiedliche Zielsetzungen und Benutzerzielgruppen wird es ständig aktualisiert und verbessert, um Interessenten der Kryptologie eine Arbeitsfläche zu bieten. Zum einen eignet es sich zur Veranschaulichung sowohl simpler als auch komplexer kryptographischer Abläufe und stellt somit ein Werkzeug für die Lehre dieses Fachbereiches dar, sei es für schulische oder autodidaktische Zwecke, zum anderen kann durch Kombination verschiedener Algorithmen und Bausteine etwas neues geschaffen und somit direkt zur Forschung auf diesem Gebiet beigetragen werden. CrypTool 2.0 beinhaltet derzeit eine Vielzahl klassischer und moderne Verschlüsselungsverfahren, Methoden der Kryptoanalyse, diverse Werkzeuge sowie Vorlagen kompletter Protokolle. Im Zuge dieser Bachelorarbeit soll nun das erste Protokoll implementiert werden, welches auf quantenmechanischer Basis funktioniert: das öffentliche Schlüsselaustauschverfahren von Charles H. Bennet und Gilles Brassard, auch bekannt als "BB84-Protokoll". [BB+84] Die Implementierung soll Interessierte dazu anregen, sich mit diesem Thema zu beschäftigen und interaktiv neue Erkenntnisse zu erlangen.

1.1 Motivation

Dass der praktische Gebrauch von quantenmechanischen Phänomenen im Alltag weiter zunimmt, ist eine Tatsache, die auch außerhalb der Universität und Forschung eine große Rolle spielt. Als populärstes Beispiel dienen hier die Lasergeräte, ohne die jegliche CD, DVD und Blu-Ray-Laufwerke nicht denkbar wären, und der Transistor, auf dem der wesentliche Erfolg des Heimcomputers beruht.

Auch im Fachbereich der Kryptologie, welche sich aus dem Bereich der Kryptographie (die Wissenschaft der Verschlüsselung von Informationen) und der Kryptoanalyse (die Wissenschaft der Analyse kryptographischer Verfahren) zusammensetzt, finden sich zusehends mehr Verfahren, die auf quantenmechanischer Basis funktionieren. So ist es beispielsweise möglich, dass fundierte Verfahren, wie der RSA-Algorithmus [RSA78], auf welchem ein Großteil der Sicherheit unserer Bankensysteme beruht, bald durch den Einsatz von Quantencomputern zunichte gemacht werden. Da es unerheblich ist, wie sicher ein Verschlüsselungsverfahren theoretisch ist, wenn die Geheimhaltung in der Praxis am Austausch des geheimen Schlüssels scheitert, müssen neue Schlüsselaustauschverfahren her, deren Sicherheit nach Möglichkeit noch in weiter Zukunft gewährleistet ist. Bisher gibt es zwei Protokolle, die einen Schlüsselaustausch auf quantenmechanischer Ebene durchführen: das Ekert-Protokoll, das verschränkte Teilchen zur Übertragung verwendet [Eke91] (dieses wird am Ende der Ausarbeitung noch kurz beschrieben), sowie das BB84-Protokoll, welches auf Basis von polarisierten Photonen arbeitet. Letzteres ist Gegenstand dieser Arbeit, in der nicht nur die Umsetzung in CrypTool 2.0, sondern auch Nutzen und Schwachstellen desselben dargestellt werden sollen.

1.2 Aufgabenstellung

Ziel dieser Bachelorarbeit ist eine vollständige Implementierung des BB84-Protokolls in CrypTool 2.0. Folgende Anforderungen werden dabei gestellt:

- R1: Der Schlüsselaustausch ist in vollem Umfang zu simulieren
- R2: Ein simulierter Lauschangriff sowie Übertragungsfehler können optional aktiviert werden
- R3: Eine Sicherheitsprüfung des übertragenen Schlüssels muss durchführbar sein
- R4: Visuelle Präsentationen zum besseren Verständnis der Funktionsweise können abgespielt werden
- R5: Die Beschriftung und Dokumentation der einzelnen Komponenten erfolgt bilingual in Deutsch und Englisch

Alle Schritte von den Grundlagen über die Planung bis zur konkreten Umsetzung dieser Anforderungen sollen im folgenden Verlauf der Ausarbeitung beschrieben werden.

1.3 Aufbau der Arbeit

Die vorliegende Bachelorarbeit umfasst neben dieser Einleitung folgende Kapitel: Kapitel 2 soll zunächst einen Überblick über die benötigten Grundlagen zum Thema geben. Hierzu zählen physikalisches und kryptographisches Grundwissen sowie eine kurze Beschreibung von CrypTool 2.0

In Kapitel 3 wird das BB84-Protokoll in seiner Funktionsweise ausführlich beschrieben und anschließend in seine Bestandteile zerlegt, anhand derer dann das Konzept und Design für die anstehende Implementierung erarbeitet werden wird.

Kapitel 4 befasst sich anschließend mit der konkreten Umsetzung der Implementierung, wobei es einzelne Komponenten detailliert beschreibt und mit Grafiken anschaulich macht. Auch die visuelle Präsentation wird beschrieben und auf ihren Nutzen hin untersucht.

Stärken und Schwächen der implementieren Computersimulation werden in Kapitel 5 diskutiert, außerdem wird eine Analyse der Sicherheit des Protokolls durchgeführt.

Ein Ausblick über zukünftige Möglichkeiten und ein Fazit über die geleisteten Arbeiten folgt abschließend in Kapitel 6.

2 Grundlagen

2.1 Kryptographische und komplexitätstheoretische Grundlagen

Alle wichtigen und im Prinzip sehr sicheren symmetrischen Kryptosysteme, seien es "Rivest Cipher 5" [Riv95], der "Advanced Encryption Standard" [DR00] oder das "One-Time-Pad" [Lis], welches sogar bewiesenermaßen perfekte Sicherheit bietet, haben eine gemeinsame Schwachstelle: Selbst wenn der eigentliche Algorithmus sicher ist, so muss bei allen diesen Verfahren stets ein gemeinsamer Schlüssel existieren, den jede Partei zum Ver- und Entschlüsseln benötigt. Würde man diesen Schlüssel nun austauschen, ohne sich dafür an den gleichen Ort zu begeben, müsste man ihn über eine Leitung verschicken, womit die Gefährdung des gesamten Sicherheitssystems riskiert wird, da immer die Gefahr eines Lauschangriffs besteht. Daher zählt der sichere Schlüsselaustausch über einen öffentlichen Kanal zu einem der wichtigsten Grundinstrumente, welches die Kryptographie bereitstellen soll. Ziel dieser Verfahren ist, Informationen öffentlich auszutauschen, ohne einem eventuellen Lauscher Rückschlüsse auf den Schlüssel zu geben. Die heute größtenteils angewandten Verfahren, wie beispielsweise der Diffie-Hellman-Schlüsselaustausch [HDM80], basieren meist auf mathematischen Problemstellungen, die nur mit enormem Rechenaufwand umgangen werden könnten. Selbst die leistungsstärksten Rechner der Gegenwart wären ab einer gewissen Schlüsselgröße zu lange beschäftigt, um effizient anhand abgefangener Daten den gemeinsamen Schlüssel der beiden Parteien zu berechnen.
Um nun zu erklären, worin die Bedrohung dieser derzeitigen scheinbar sicheren Verfahren liegt, müssen diese zunächst durch die Komplexitätstheorie beschrieben werden. Diese kategorisiert Problemstellungen wie das Faktorisieren ganzer Zahlen oder das Problem des diskreten Logarithmus in die Kompexitätsklasse NP,

was aussagt, dass diese Aufgaben von einer Nichtdeterministischen Turingmaschine in Polynomialzeit lösbar sind. Nun ist das Modell der Nichtdeterministischen Turingmaschine bis heute nicht technisch realisierbar und somit, sollte sich nicht herausstellen, dass P = NP gilt, noch keine Bedrohung [Weg05]. Es gibt jedoch erste Versuche, Computer auf Quantenbasis zu verwenden, um Probleme, die bisher exponentielle Zeit zur Berechnung benötigten, nun in Polynomialzeit zu lösen. Als Beispiel sei hier der Algorithmus von Peter Shor aus dem Jahr 1993 genannt [Sho94], der das Faktorisierungsproblem in Polynomialzeit löst, auch wenn er derzeit noch starken technischen Einschränkungen unterliegt. Für eine Auseinandersetzung mit dem Thema der Komplexitätslehre empfiehlt sich eine Lektüre der Ausarbeitung von Johannes Köbler und Olaf Beyersdorff mit dem Titel "Von der Turingmaschine zum Quantencomputer - ein Gang durch die Geschichte der Komplexitätstheorie" [KB06]
Um dieser Entwicklung vorzubeugen ist es also ratsam, neue Verfahren zu entwickeln, die auf anderen Grundlagen als mathematischem Rechenaufwand beruhen. Das BB84-Protokoll auf Basis polarisierter Photonen ist Gegenstand dieser Arbeit.

2.2 Physikalische Grundlagen

Das zentrale quantenmechanische Phänomen dieser Arbeit ist die Polarisation von Licht. Hierbei ist zunächst wichtig, dass die Schwingungsrichtung des Lichtes normal zur Ausbreitungsrichtung ist. Diese Art von Wellen, auch Transversalwellen genannt, lassen sich durch den Einsatz bestimmt ausgerichteter Filter polarisieren, was bedeutet, dass nach der Polarisation nur eine bestimmte Schwingungsebene vorhanden ist. Abbildung 2.1 zeigt die Wirkung eines horizontal ausgerichteten Filter auf unpolarisiertes Licht.

Wie zu erkennen ist, bleiben nach der Polarisation lediglich diejenigen Wellen übrig, deren Schwingung ebenfalls horizontal ausgerichtet ist. Schickt man dieses polarisierte Licht nun durch den selben Filter noch einmal, gelangt das gesamte Licht hindurch. Schickt man es jedoch durch einen um neunzig Grad gedrehten Filter, wird sämtliches Licht absorbiert (Abbildung 2.2).

Nun muss noch der Fall betrachtet werden, in welchem das polarisierte Licht nicht auf einen um neunzig, sondern um fünfundvierzig Grad gedrehten Filter trifft. In

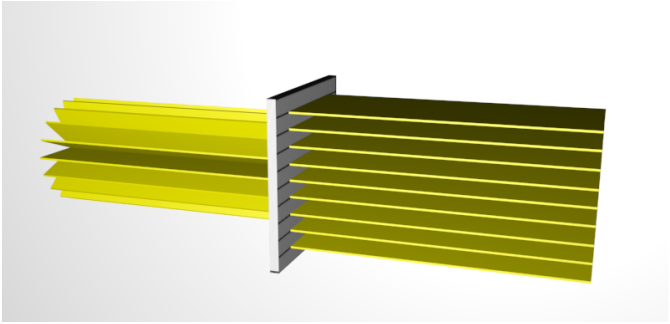

Abbildung 2.1: Polarisation durch einen horizontalen Filter

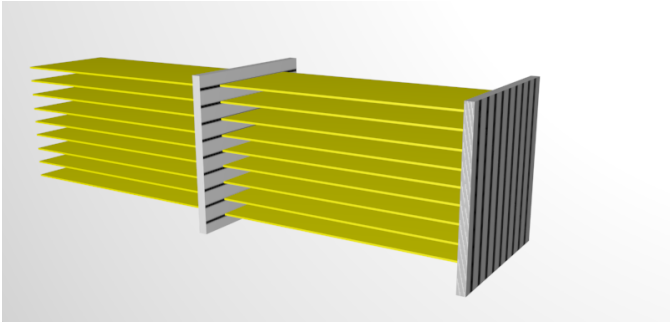

Abbildung 2.2: Polarisiertes Licht wird komplett durchgelassen bzw. absorbiert

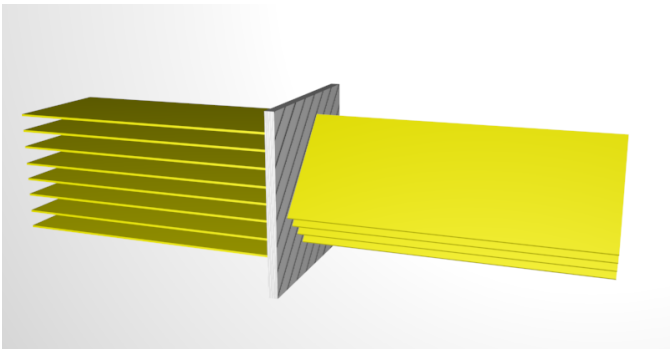

Abbildung 2.3: Die Hälfte des Lichts gelangt durch den Filter

diesem Fall gelangt 50% des Lichtes durch den Filter, die andere Hälfte wird absorbiert (Abbildung 2.3).

Dieser Fall ist besonders interessant, wenn wir nun vom Wellencharakter des Lichts zu dessen Teilchencharakter übergehen. Trifft nämlich ein einzelnes polarisiertes Photon auf einen Spalt mit einer Ausrichtung, die 45 Grad zu seiner eigenen Polarisation gedreht ist, so kann es mit einer Wahrscheinlichkeit von 50% absorbiert werden oder durch den Filter gelangen, wobei es die Ausrichtung des Spalts annimmt, durch das es geschickt wurde. Dieses Phänomen hängt also vom Zufall ab, und an dieser Stelle ist es absolut essentiell, dass der Zufall in der Quantenmechanik nicht dem des üblichen Verständnis vom Zufall entspricht. Dieser Unterschied wurde treffend von Anton Zeilinger formuliert:

> Der Zufall in der Quantenphysik ist also nicht ein subjektiver, er besteht nicht deshalb, weil wir zu wenig wissen, sondern er ist objektiv. Ganz im Sinne Heisenbergs ist es nicht unser Unwissen, von dem wir hier also sprechen, sondern die Natur selbst ist in solchen Situationen in keiner Weise festgelegt, ehe das Ereignis auftritt [...].[Zei03]

Ob das Photon durch den Filter gelangt, ist also nicht nur schwierig oder unberechenbar, sondern vollkommen unmöglich vorherzusagen, da es vor dem eigentlichen Ereignis nicht festgelegt ist. Dieser objektive oder absolute Zufall ist für die Kryptographie von enormer Wichtigkeit, da etwas, das nicht festgelegt ist, auch nicht von einem eventuellen Angreifer vorhergesagt oder geraten werden kann. Für weitere hilfreiche Informationen über die Quantenmechanik und ihre Auswirkung auf moderne Technik empfehle ich das Buch von Peter Rennert mit dem Titel "Einführung in die Quantenphysik" [RCH13]
Nachdem die Polarisation von Photonen nun beschrieben wurde, soll eine Beschreibung des Programmes, in dem diese Arbeit implementiert wird, erfolgen.

2.3 CrypTool 2.0

CrypTool 2.0 (im Folgenden: CT2) ist eine interaktive E-Learning-Plattform, die auf Basis der .NET-Technologie [LM04] und mit Hilfe der Windows Presentation Foundation (WPF) [Hub11] sowie der Programmiersprache C# [Lib09] realisiert wurde. Kernstück von CT2 ist der Arbeitsflächen-Editor (vgl. Abbildung 2.4), auf

welchem der Benutzer vorgefertigte Komponenten (oft auch als “Plugins” bezeichnet) platzieren und miteinander verbinden kann. Die so entstehenden Algorithmen können für simple Ver- und Entschlüsselungsvorgänge genutzt werden, oder auch komplexe Systeme in allen Bereichen der Kryptologie simulieren.

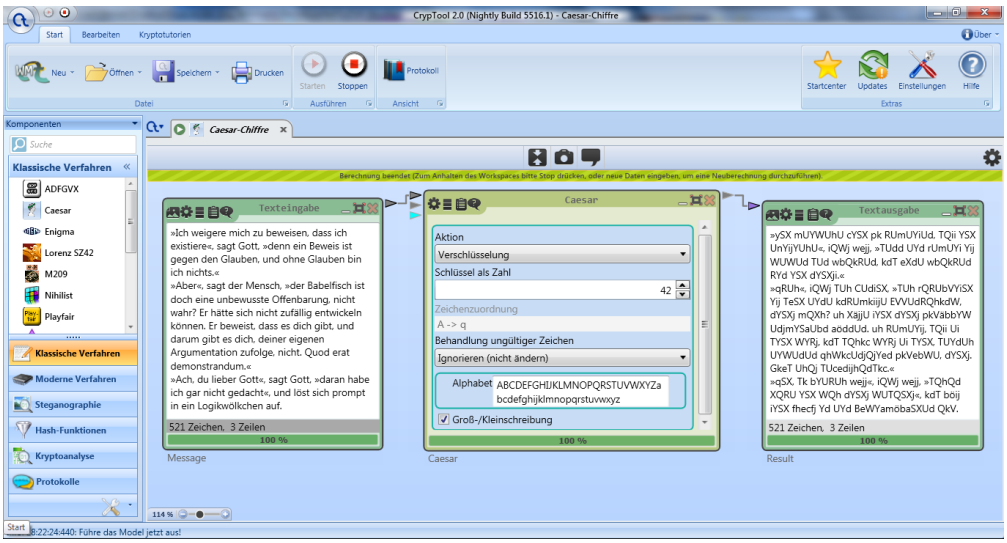

Abbildung 2.4: Screenshot der CT2-Oberfläche

Da CT2 sehr komplex ist, sollen hier nicht alle Details seines Aufbaus und seiner Möglichkeiten aufgezählt werden. Stattdessen werden die für die diese Arbeit relevantesten Elemente vorgestellt: der Arbeitsflächen-Editor, die Komponenten und die Vorlagen.

2.3.1 Der Arbeitsflächen-Editor

Wie in der Einleitung bereits erwähnt, ist der Editor das wichtigste Element von CrypTool 2.0. Er bietet die Oberfläche, auf der alle anderen Bausteine benutzt werden können, sowie Steuerungselemente, um die erstellten Prozesse auszuführen und zu kontrollieren. Abbildung 2.5 zeigt ein leeres Editor-Fenster und seine Aufteilung.

Dabei sind folgende Elemente farblich eingerahmt:

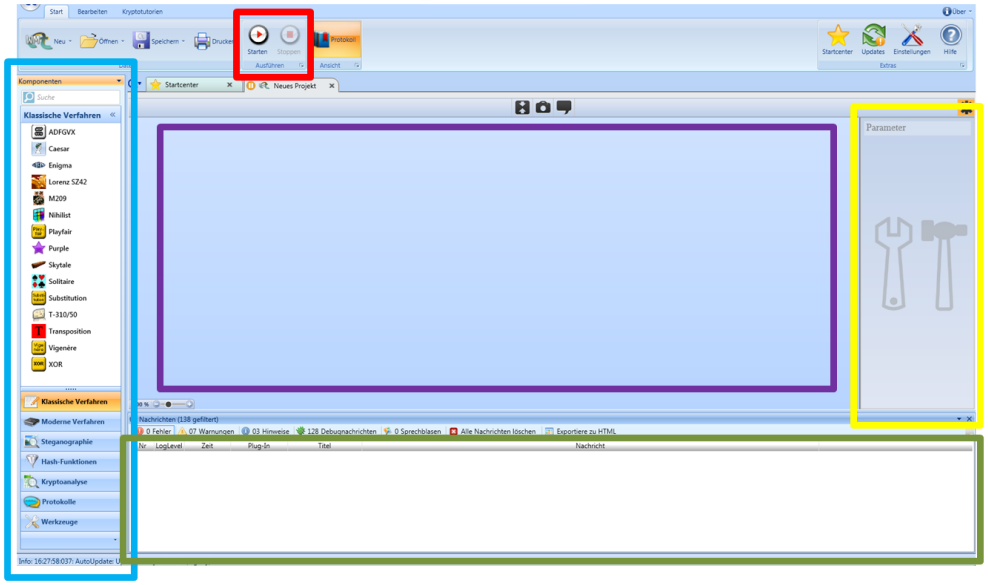

Abbildung 2.5: Aufbau des Arbeitsflächen-Editors

Die Komponenten (blau)

Die linke Leiste des Fensters enthält eine geordnete Liste aller verfügbaren Komponenten, die vom Anwender benutzt werden können. Wie zu sehen ist, werden diese Komponenten in Kategorien wie “Klassische Verfahren”, “Moderne Verfahren” oder “Kryptoanalyse” unterteilt, um leichter auffindbar zu sein. Um eine Komponente zu benutzen, muss sie lediglich mit der Maus auf die Arbeitsfläche gezogen werden.

Die Arbeitsfläche (violett)

In der Mitte des Fensters befindet sich eine große leere Fläche, auf der die ausgewählten Komponenten platziert werden können. Die Bedienung, wie z.B. Verschiebung und Vergrößerung, erfolgt intuitiv wie beim normalen Umgang mit anderen Windows-Anwendungen. Auf die Komponenten im einzelnen wird später noch eingegangen werden.

Die Start/Stop-Elemente (rot)

Am oberen Bereich des Fensters befinden sich neben den standardmäßigen Öffnen-, Speicher- und Druck-Buttons auch die beiden Schaltflächen zum Starten und

Stoppen des ausführbaren Bereichs. Wird auf den Startknopf geklickt, werden alle Komponenten aktiv und können ihre jeweilige Funktion ausführen. Sind alle Programmteile durchgelaufen, so endet die Ausführung; mit einem Klick auf den Stopp-Button kann sie jedoch auch vorzeitig beendet werden.

Die Komponenten-Einstellungen (gelb)

Die rechte Seitenleiste, welche bei Bedarf auch ausgeschaltet werden kann, zeigt alle Einstellungsmöglichkeiten für die gerade ausgewählte Komponente. Zwar können die Einstellungen auch auf der Arbeitsfläche direkt an der Komponente vorgenommen werden, jedoch ist die Fläche auf der rechten Seite meist viel größer und damit übersichtlicher.

Die Benachrichtigungen (grün)

Am unteren Bildschirmrand befindet sich ein weiterer Bereich, der ebenfalls vom Benutzer ausgeblendet werden kann. Hier werden mögliche Hinweise, Debugnachrichten, Warnungen und Fehler im vorhandenen Aufbau angezeigt. Der Benutzer hat außerdem die Möglichkeit, den Log per Knopfdruck in eine HTML-Datei zu exportieren.

2.3.2 Die Komponenten

Jede Komponente in CrypTool 2.0 verkörpert ein eigenständiges kleines Programm, welches von einfacher Textausgabe bis hin zu komplexen Kryptographischen bzw. Kryptoanalytischen Algorithmen alles beinhalten kann. Rein optisch wird eine Komponente als ein bewegliches Icon auf dem Bildschirm dargestellt. Als Beispiel soll hier die Transpositionskomponente aus CT2 dienen, die eine Verschlüsselung auf Basis von vertauschten Zeichen durchführt. Da diese nicht Bestandteil dieser Arbeit ist, nimmt ihre Beschreibung keine Aspekte vorweg.

Wie in Abbildung 2.6 zu sehen ist, kann eine Komponente über mehrere Ein- und Ausgänge verfügen, dargestellt als ein- bzw. ausgehende Dreiecke. In diesem speziellen Fall sind zwei Eingänge (der Klartext sowie das Schlüsselwort) und ein Ausgang (der verschlüsselte Text) vorhanden. Damit die Komponente ordentlich

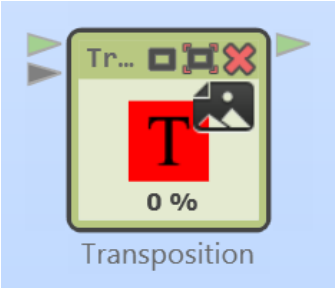

Abbildung 2.6: Beispiel einer Komponente

funktioniert, müssen weitere Komponenten hinzugefügt werden, um die benötigten Daten bereitzustellen. Die simpelste Form, in der das geschehen kann, ist in Abbildung 2.7 dargestellt. Hier wurden einfach zwei Texteingabekomponenten und eine Textausgabekomponente mit der Transposition verbunden, anschließend wurde das gesamte Projekt gestartet.

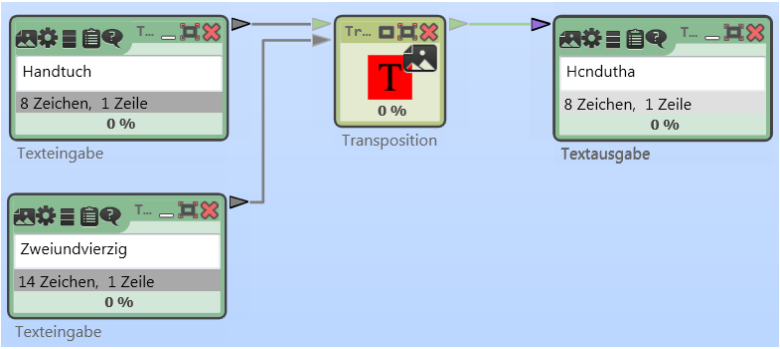

Abbildung 2.7: Anwendung der Transposition

Die Darstellung der Komponente kann auch maximiert werden, um weitere Möglichkeiten der Anzeige zu bekommen. Zum Einen kann in der oberen Leiste der maximierten Komponente das Zahnradsymbol ausgewählt werden, um die Einstellungsmöglichkeiten zu erhalten (vgl. Abbildung 2.8). Jede Komponente verfügt dabei über eigene Einstellungen, die vom Entwickler vorgegeben worden sind. Die verschiedenen Konfigurationsmöglichkeiten werden dabei durch GUI-Elemente wie Combo-Boxen, Textfelder und Slider ermöglicht, die für den Benutzer intuitiv zu bedienen sind.

Des weiteren kann für Komponenten, bei denen eine Präsentation implementiert wurde, diese über das Bildsymbol angezeigt werden. Präsentationen stellen graphische Repräsentationen der implementieren Algorithmen dar, die durch WPF-Komponenten realisiert werden. Abbildung 2.9 zeigt die Präsentation der Transpositionskomponente während der Ausführung. Durch die dargestellten Animationen

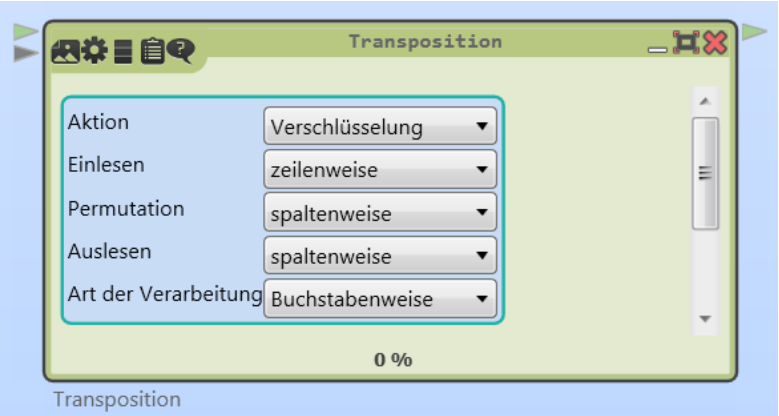

Abbildung 2.8: Einstellungen der Transpositionskomponente

ist der Arbeitsweg des Algorithmus für den Betrachter um ein Vielfaches einfacher nachzuvollziehen. Am unteren Rand des Fensters ist außerdem ein Fortschrittsbalken vorhanden, der den aktuellen Ausführungsfortschritt beschreibt.

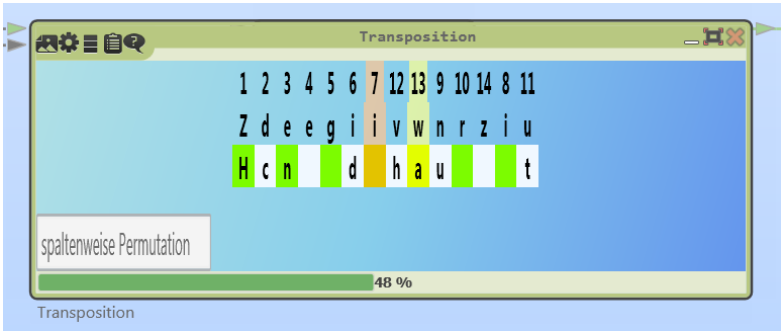

Abbildung 2.9: Präsentation der Transpositionskomponente

2.3.3 Die Vorlagen

Als Vorlage bezeichnet man in CT2 eine vollständig vorgegebene Arbeitsoberfläche, welche bereits Komponenten und deren Verknüpfung untereinander enthält. Abbildung 2.10 zeigt als Beispiel die in CT2 implementierte Vorlage zum Diffie-Hellman-Schlüsseltausch.

Zu jeder Vorlage ist eine Hilfedatei enthalten, welche die Funktionsweise und die Möglichkeiten der Einstellung detailliert beschreibt und erklärt. Vorlagen eignen sich besonders für den didaktischen Aspekt von CT2, da eventuelle Neulinge im Bereich der Kryptographie oft nicht wissen, in welcher Weise die Komponenten korrekt verbunden werden müssen, um zum Beispiel ein vollständiges Protokoll

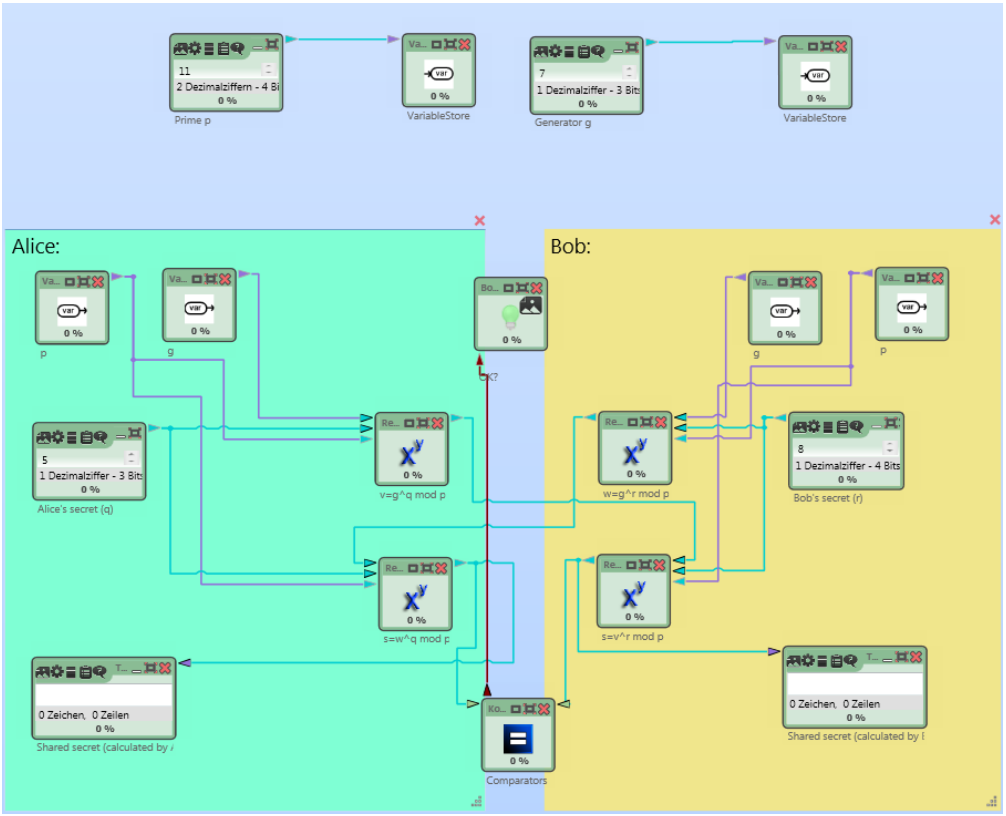

Abbildung 2.10: Diffie-Hellman-Schlüsseltausch (Vorlage)

darzustellen. Da das Ziel dieser Arbeit ebenfalls die Implementierung eines kompletten kryptographischen Protokolls ist, soll hierzu also auch eine Vorlage erstellt werden. Der Hauptvorteil einer Vorlage liegt darin, dass keine weiteren Einstellungen an der Arbeitsfläche mehr vorgenommen werden müssen und die gesamte Konstruktion nach dem Laden einfach ausgeführt werden kann. Fortgeschrittene Benutzer können jedoch auch an Vorlagen Änderungen vornehmen und diese in einer eigenen Datei abspeichern.

Da nun die Grundlagen vermittelt wurden, wird im folgenden Kapitel das Konzept der Arbeit entwickelt, bevor es in den weiteren Kapiteln zur eigentlichen Implementierung kommt.

3 Konzept und Entwurf

3.1 Konzept: Das BB84-Protokoll

Das zu implementierende Protokoll wurde 1984 von Charles H. Bennet und Gilles Brassard unter dem Titel "Quantum Cryptography: Public Key Distribution and Coin Tossing" veröffentlicht. Es beschreibt, wie mit Hilfe von polarisierten Photonen ein sicherer Schlüsselaustausch über einen öffentlichen Kanal erfolgen kann. Der komplette Vorgang des Protokolls lässt sich in folgende fünf Schritte aufteilen:

1. Photonenkodierung durch Sender
2. Versuch eines Angriffs durch Mittelsmann
3. Photonendekodierung durch Empfänger
4. Bestimmen des gemeinsamen Schlüssels von Sender und Empfänger
5. Prüfen der Nachricht auf Übertragungsfehler

Zu beachten ist, dass Schritt 2 rein optional ist, da ein Angriff nicht bei jeder Übertragung stattfinden muss. In der Ausarbeitung von Bennet und Brassard wird zunächst die abhörfreie Übertragung beschrieben bevor im weiteren Verlauf der mögliche Angriff erklärt wird. Da jedoch chronologisch betrachtet der Angriff vor dem Dekodieren durch den Empfänger stattfindet, ist er in dieser Liste der Schritte als zweiter Punkt eingeordnet.
Jeder dieser Schritte soll nun im Detail beschrieben werden. Der Sender wird dabei im Folgenden Alice genannt, der Empfänger Bob und der Mittelsmann Eve.

3.1.1 Photonenkodierung durch Sender

Alice startet zunächst mit dem Erzeugen eines Initialschlüssels in Bit-Form. Hierbei ist zu beachten, dass im weiteren Verlauf des Protokolls etwa die Hälfte des Schlüssels verworfen werden muss. Dies muss beim Erzeugen berücksichtigt werden, damit die Schlüssellänge nicht ungewollt kurz ausfällt. Nun erfolgt die eigentliche Kodierung. Für jedes einzelne Bit des Schlüssels muss ein Photon erzeugt, polarisiert und anschließend verschickt werden. Zur Polarisation können zwei verschiedene Basen genutzt werden: eine Basis, welche einen horizontalen und einen vertikalen Filter besitzt (im Folgenden der Form entsprechend als "Plus-Basis‘" bezeichnet) oder eine Basis, deren Filter jeweils um 45 Grad gedreht sind (im Folgenden als "X-Basis bezeichnet"). Die Polarisationen der X-Basis werden hier "linksdiagonal" (Filter von oben links nach unten rechts) oder "rechtsdiagonal" (umgekehrt) genannt. Zwei Dinge sind nun zu beachten: zum Einen muss für jedes einzelne Bit die Polarisationsbasis zufällig gewählt werden, zum Anderen muss die eigentliche Kodierung innerhalb einer Basis für alle beteiligten (auch für eventuelle Angreifer) offen liegen. Zu diesem Zweck wird sich am besten bereits vorher auf eine feste Tabelle geeinigt, wie beispielsweise Abbildung 3.1 zeigt.

Basis	0	1
+	−	|
×	＼	／

Abbildung 3.1: Beispieltabelle für Photonenkodierung

Alice wählt also für jedes Bit eine zufällige Basis aus und kodiert das Bit dann anhand der festgelegten Tabelle. Anschließend sendet sie die polarisierten Photonen über den öffentlichen Kanal an Bob.

3.1.2 Versuch eines Angriffs durch Mittelsmann

Bevor die gesendeten Photonen nun bei Bob ankommen, besteht die Gefahr, dass Eve den Kanal abhört und die Photonen abfängt. Da sie jedoch keine Information besitzt, welche Basen für welche Bits verwendet wurden, muss sie selbst für jedes Photon willkürlich eine Basis auswählen und es anhand dieser messen. Liegt sie mit ihrer Wahl richtig, empfängt sie das korrekte Bit, misst sie jedoch mit der

falschen Basis, polarisiert sie das empfangene Photon unabsichtlich in ihre Basis um und erhält ein falsches Ergebnis. Welche Photonen sie richtig und welche falsch gemessen hat, kann sie zu diesem Zeitpunkt jedoch nicht feststellen. Nun können die empfangenen Photonen über die offen zugängliche Tabelle dekodiert werden. Zuletzt muss Eve die von ihr abgefangenen Photonen wieder über den Kanal weiterschicken, da Bob sonst keine Daten erhält und der Lauschangriff auffällt. Jedes Photon, dass sie unabsichtlich verändert hat, wird also auch in veränderter Form weitergeschickt. Eine Kopie des Photons zu erzeugen ohne es vorher zu messen ist nach dem No-Cloning-Theorem [WZ82] nicht möglich.

3.1.3 Photonendekodierung durch Empfänger

Bob durchläuft nun die selben Schritte wie Eve: zuerst muss er für jedes Photon eine zufällige Basis zur Messung auswählen, anschließend dekodiert er die gemessenen Daten nach der bekannten Tabelle. Zu diesem Zeitpunkt weiß er nicht, ob der Kanal abgehört worden ist. Er dekodiert alle Photonen, die er erhält, egal ob sie verändert worden sind oder nicht.

3.1.4 Bestimmen des gemeinsamen Schlüssels

Wie bereits erwähnt sind die Schlüssel von Alice und Bob selbst, wenn kein Lauschangriff stattfindet, nicht gleich, da nicht immer die gleichen Basen verwendet wurden. Aus diesem Grund müssen sich die beiden nun über die von ihnen gewählten Basen austauschen und die Bits, bei denen sie unterschiedliche Basen zur Kodierung bzw. Dekodierung verwendet haben, verwerfen. Sie erhalten hierdurch den gemeinsamen Schlüssel. Da dies ebenfalls über einen öffentlichen Kanal geschieht, hat Eve nun die Möglichkeit, ihre Fehlerhaften Bits ebenfalls auszusortieren. Da ihr Lauschangriff dadurch abgeschlossen ist, muss für Alice und Bob nun noch eine Prüfung durchgeführt werden, die ihnen einen möglichen Angriff aufzeigt. Die geschieht mittels einer Fehlerprüfung.

3.1.5 Prüfung auf Übertragungsfehler

Zwar hat Eve im Falle korrekter Ausführung ihres Angriffs nun einen Teil des Schlüssels abgehört, jedoch hat sie in 25% aller Fälle einen Fehler in der Übert-

ragung verursacht. Diese Zahl kommt zustande, da sie beim Messen mit einer Wahrscheinlichkeit von 50% eine falsche Basis wählt, das so verfälschte Photon aber nur mit ebenfalls 50% Wahrscheinlichkeit von Bob richtig gemessen wird. Zur Überprüfung auf einen Lauschangriff müssen Alice und Bob nun also die Fehlerrate ihrer Übertragung kontrollieren. Dies geschieht am besten, indem man sich auf eine bestimmte Sequenz innerhalb des Schlüssels einigt, welche anschießend verglichen wird, um die Fehlerrate der gesamten Übertragung abzuschätzen. Die benutzte Sequenz muss anschließend natürlich aus dem Schlüssel entfernt werden, da die Überprüfung ebenfalls abgehört worden sein könnte. Des weiteren muss bei jedem Schlüsseltausch eine andere Sequenz von Zeichen ausgewählt werden, damit sich ein Mittelsmann diesen Bereich nicht für das nächste Mal merken kann. Aber auch bei einer nicht angegriffen Übertragung können Fehler auftreten, was auf die Leitungen und Messgeräte zurückzuführen ist. Aus diesem Grund sollte sich hier auf einen Fehlergrenzwert geeinigt werden, welcher zwischen normalen Übertragungsfehlern und Veränderung durch einen Angreifer unterscheidet. In der Ausarbeitung von Bennet und Brassard wird der genaue Betrag dieses Schwellwerts zwar nicht festgelegt, da statistisch gesehen jedoch bei einem Lauschangriff jedes vierte Photon verändert wird, könnte er beispielsweise bei 25% angesetzt werden.

3.1.6 Einschub: Schlüsselwiederherstellung

Die im vorherigen beschriebenen Übertragungsfehler können für einen ausgetauschten Schlüssel fatale Konsequenzen haben. Weicht auch nur eine Stelle im gemeinsamen Schlüssel beider Parteien ab, so wird er meist unbrauchbar, da er beispielsweise für ein symmetrisches Verschlüsselungsverfahren nicht mehr genutzt werden kann. Daher müssen in der Praxis die oben genannten Schritte durch einen weiteren ergänzt werden: der Korrektur und Wiederherstellung des Schlüssels. Diese muss über einen öffentlichen Kanal erfolgen und darf einem Angreifer möglichst keine Informationen über den Schlüssel offenbaren. Obwohl dieser Schritt nicht zum BB84-Protokoll gehört, sollen hier zwei Möglichkeiten dieser Korrektur kurz beschrieben werden.

3.1.6.1 Austausch von Paritätsbits

Eine simple Möglichkeit, um ausgetauschte fehlerhafte Schlüssel wiederherzustellen, ist der Austausch von Paritätsbits. Hierbei teilen Alice und Bob ihren Schlüssel zunächst in Blöcke aus je zwei Bits auf. Anschließend bestimmen sie von jedem der Blöcke die Parität und vergleichen diese Paritätsbits miteinander. Sind sie unterschiedlich, so ist ein Übertragungsfehler aufgetreten und der entsprechende Block wird verworfen. Bei Übereinstimmung löschen beide jeweils eines der beiden Bits, wobei die Entscheidung darüber, welches Bit verworfen wird, bereits im Voraus gemeinsam getroffen wurde. Das übrige Bit verbleibt im Schlüssel. So kann ein Mittelsmann durch das Abhören der Parität nicht auf das finale Bit des Schlüssels schließen.
Der Einfachheit dieses Ansatzes stehen zwei Dinge entgegen: zum einen besteht in jedem Block die Möglichkeit, dass beide Bits verfälscht wurden, was das Paritätsbit wieder den scheinbar richtigen Wert annehmen lässt. Zum anderen muss erneut ein großer Teil des Schlüssels verworfen werden: selbst im Falle einer fehlerfreien Übertragung gehen bei n Bits $n/2$ Stellen verloren, bei einer Fehlerrate von 50% (gleichverteilt) wird sogar der gesamte Schlüssel verworfen. Ein effizienterer Algorithmus ist also wünschenswert.

3.1.6.2 Das Cascading-Protokoll

Dieser Ansatz zur Schlüsselwiederherstellung ist aus dem Paper "Secret-Key Reconciliation by Public Discussion" von Gilles Brassard und Louis Salvail[BS94] entnommen.
In diesem Protokoll finden mehrere Iterationen statt, in denen die Schlüssel, ähnlich wie im vorherigen Ansatz, in Blöcke aufgeteilt und diese auf ihre Parität verglichen werden. Stellt man eine unterschiedliche Parität fest, so wird eine binäre Suche ausgeführt, um den Fehler zu finden und zu korrigieren. Wird ein Fehler in einem Block einer vorangegangenen Runde gefunden, in der die Parität korrekt war, so muss ein weiterer Fehler in diesem Block vorhanden sein, der ebenfalls korrigiert wird. Dieser Prozess wird rekursiv (kaskadenartig) wiederholt. Nachdem alle Blöcke verglichen wurden, ordnen Alice und Bob ihre Schlüssel auf gleiche, zufällig vorher ausgewählte Art und es beginnt eine neue Runde. Nach mehreren Runden haben Alice und Bob mit hoher Wahrscheinlichkeit den selben Schlüssel.

Es ist jedoch zu beachten, dass bei Ausführung dieses Protokolls für einen lauschenden Mittelsmann viele Informationen bereitgestellt werden. Daher ist es zu empfehlen, nach dessen Ausführung beide Schlüssel zu verändern. Dies geschieht zum Beispiel durch Anwendung einer zufällig ausgewählten universellen Hashfunktion aus einem gleichen, vorher festgelegten Pool an Funktionen.
Für die praktische Ausführung des BB84-Protokolls ist ein fehlerkorrigierender Ansatz dieser Art essentiell.

3.2 Entwurf: Die Komponenten

Um die Architektur der Protokollsimulation zu planen, muss zuerst entschieden werden, welche Inhalte des Protokolls sich in welche Komponenten aufteilen lassen. Anschließend müssen diese Komponenten den Teilnehmern des Protokolls zugeteilt werden. Die folgenden Angaben sind Plattform-unabhängig und dienen dem Zweck der allgemeinen Planung einer späteren Implementierung.
Folgende Komponenten werden benötigt:

- Der Photonenbasis-Generator
- Der Photonenkodierer
- Der Mittelsmann-Simulator
- Der Photonendekodierer
- Der Schlüsselgenerator
- Der Fehlerdetektor

Für die Beschreibung der Schlüssel, der Photonen und der Filterbasen werden Strings verwendet, da sie intuitiver benutzbar sind und sich für die Beschreibung einer fließtextartigen Struktur sehr gut eignen.

3.2.1 Der Photonenbasis-Generator

Der Photonenbasis-Generator stellt die am häufigsten benötigte Komponente dar, weil sie von jeder Partei des Protokolls benötigt wird. Als Information benötigt dieser Baustein die Länge des Schlüssels bzw. des Photonenstroms und erzeugt einen

Strom aus zufällig generierten Photonenbasen der gleichen Länge. Alice benötigt diese Komponente, um den Initialschlüssel zu kodieren, Bob und Eve jeweils um den erhaltenen Photonenstrom zu dekodieren. Wie überall in der Kryptographie ist hier ein guter Zufallsgenerator sehr wichtig, da, wenn Vorhersagen über die nächsten Basen gemacht werden können, die Sicherheit des gesamten Schlüsseltausches in Gefahr ist.

3.2.2 Der Photonenkodierer

Der Photonenkodierer benötigt die Eingabe eines Schlüssels in Bit-Form, sowie die des Basenstroms des Photonenbasis-Generators. Nun wird die zu verschickende Sequenz durch eine Tabelle aufgelöst. Für jedes zu verschickende Photon muss das entsprechende Schlüsselbit sowie die Basis des Photonenbasis-Generators betrachtet werden. Zur Darstellung des Photonenstroms kann ein String verwendet werden, dessen Länge gleich ist mit der des Schlüssels und des Basenstroms. Diese muss anschließend ausgegeben werden.
Zusätzlich zur Funktionalität dieser Komponente soll sie ebenfalls eine Animation beinhalten, welche die Polarisation der einzelnen Photonen und deren Sendung beschreibt. Die Geschwindigkeit dieser Animation soll variabel und konfigurierbar sein.

3.2.3 Der Mittelsmann-Simulator

Der Mittelsmann-Simulator empfängt die Ausgabe des Photonenkodierers und misst zunächst die Länge des erhaltenen Strings. Diese empfangene Länge gibt er nun in seinen Photonenbasis-Generator ein, um die Basen zu erhalten, die er für die Dekodierung benötigt. Anschließend benutzt er die öffentliche Kodierungstabelle, um den empfangenen String in einen Schlüssel zu verwandeln. Hierbei kann der Fall auftreten, dass die Polarisation eines Photons nicht mit der benutzten Basis übereinstimmt. Geschieht dies, so muss das Zeichen im String durch ein zufälliges der zugehörigen Base ersetzt werden und erst dann mit der Tabelle abgeglichen und übersetzt werden. Dies simuliert, dass der Mittelsmann ein Photon nicht messen kann, ohne es durch sein Messgerät zu schicken. Außerdem muss nach der Dekodierung der gesamte String inklusive der veränderten Zeichen weitergeschickt werden, um zu Bob zu gelangen.

Für den Benutzer der Simulation muss es möglich sein, den Mittelsmann-Simulator ein- und auszuschalten, um Schlüsseltauschvorgänge mit und ohne Angriff zu simulieren. Wird die Komponente in den inaktiven Zustand gesetzt, so muss ein alternatives Programm ablaufen, das den empfangenen String einfach weiterschickt, ohne ihn zu verändern.
Auch diese Komponente sollte über eine Animation verfügen, die das ankommende Photon, dessen Abhörung und Weiterschickung zeigt. Die Animationsgeschwindigkeit soll wieder einstellbar sein.

3.2.4 Der Photonendekodierer

Der Photonendekodierer empfängt die originale oder veränderte Zeichenkette vom Mittelsmann-Simulator. Des weiteren ist hier erneut ein Photonenbasis-Generator nötig, dessen Ausgabe zusammen mit dem Photonenstring anhand der Kodierungstabelle aufgelöst werden kann. Außerdem wäre es wünschenswert, an dieser Stelle eine Möglichkeit zum Ein- und Ausstellen von Übertragungsfehlern zu implementieren. Wird diese Funktion eingestellt, kommt es bei einem gewissen Prozentsatz von Dekodierungsvorgängen zu einer fehlerhaften Auflösung des Schlüssels, was beispielsweise Störungen im Messgerät simulieren soll.
Diese Komponente soll die Animation der Photonensendung vervollständigen. Sie zeigt, wie die Photonen beim Empfänger ankommen und durch den Filter gelangen. Bei Verwendung des falschen Filters für ein Photon wird visuell über den Fehler aufgeklärt und gezeigt, wie das nun falsch polarisierte Photon gemessen wird.

3.2.5 Der Schlüsselgenerator

Der Algorithmus des Schlüsselgenerators soll den gemeinsamen Schlüssel von Alice und Bob bestimmen. Daher benötigt er als Eingabe den vom Photonendekodierer dekodierten Schlüssel, den Basenstring vom Alices Photonenbasis-Generator und denjenigen von Bob. Nun vergleicht er diese beiden Strings und wählt als Ausgabe die entsprechenden Stellen des Schlüssels, an welchen Alice und Bob gemeinsame Basen verwendet haben. Textauszug 3.1 zeigt die Implementierung in Pseudocode.

Textauszug 3.1: Schlüsselgenerator

```
1 for i=0 to key.length
2   if firstBases[i] == secondBases[i]
3     commonKey += key[i]
4 return commonKey
```

Geschieht dies bei Alice und Bob, so erhalten sie beide den gleichen Schlüssel.

3.2.6 Der Fehlerdetektor

Der Fehlerdetektor verfolgt das Ziel, einen eventuell erfolgten Lauschangriff aufzudecken. Zu diesem Zweck muss er die gemeinsamen Schlüssel von Alice und Bob vergleichen und die Fehlerrate berechnen, aus der sich bestimmten lässt, ob der Schlüsselaustausch sicher erfolgt ist oder nicht. Als Eingabe erhält diese Komponente die beiden finalen Schlüssel und die Ausgabe besteht in einer Nachricht über das Ergebnis der Fehlerberechnung. Welcher Bereich des Schlüssels verglichen wird, muss variabel einstellbar sein. Außerdem muss der Fehlerschwellwert für eine sichere Übertragung vom Benutzer verändert werden können.

3.2.7 Verteilung und Verbindung der Komponenten

Das gesamte Protokoll lässt sich in drei Parteien einteilen: den Sender Alice, den Empfänger Bob, und dem Angreifer Eve. Basierend auf dieser Aufteilung können die soeben beschriebenen Komponenten den Parteien zugeteilt werden. Des weiteren müssen zusätzliche Komponenten für die Ein- und Ausgabe von Daten hinzugefügt werden. Die komplette Struktur des Protokolls ist in Abbildung 3.2 aufgeführt. Hierbei stehen die orangefarbenen Elemente für die neu erstellten Komponenten, die grauen Elemente stellen Ein- und Ausgabemöglichkeiten von Strings dar.

Bei Alice beginnend wird der Initialschlüssel zunächst in den Photonenkodierer eingegeben, da die Bits dort kodiert werden müssen. Des weiteren findet auch eine Weitergabe an den Photonenbasis-Generator statt, da diesem die Länge des

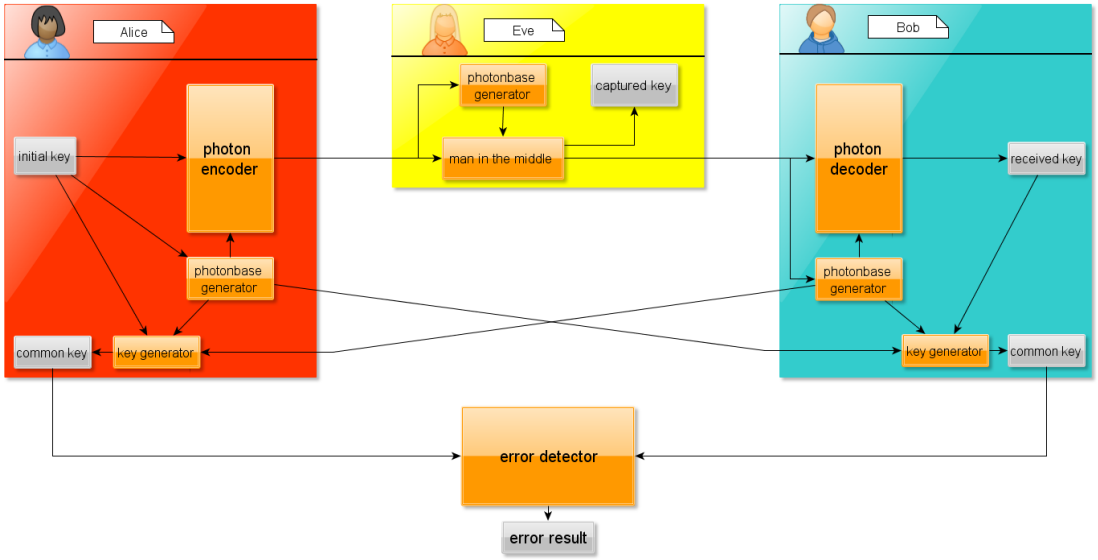

Abbildung 3.2: Vollständige Struktur des Protokolls

Schlüssels bekannt sein muss, um dementsprechend viele Photonenbasen zu erzeugen. Auch der Schlüsselgenerator benötigt diese Eingabe, um später die irrelevanten Bits zu streichen und den gemeinsamen Schlüssel zu erzeugen. Wie zu sehen ist benötigt Eves Photonenbasis-Generator nur den abgefangenen Photonenstrom, da dessen Länge der des Schlüssels entspricht. Sie erzeugt also dieselbe Anzahl zufälliger Photonenbasen und versucht anhand dieser, den Photonenstrom zu dekodieren. Es folgt die Weitergabe der Photonen an Bob, der nun ebenfalls den Strom durch seinen Photonenbasis-Generator und den Photonendekodierer laufen lässt, um schließlich seinen empfangenen Schlüssel zu erhalten.
Der nun folgende Schritt, die Erzeugung des gemeinsamen Schlüssels, geschieht bei beiden Parteien simultan und benötigt auf beiden Seiten sowohl die Ausgabe des Photonenbasisgenerators von Alice als auch die von Bob, da die Basen miteinander verglichen werden müssen, um die nicht übereinstimmenden Stellen im Schlüssel zu streichen. Zuletzt erfolgt eine Eingabe beider gemeinsamer Schlüssel in den Fehlerdetektor, in welchem sich auf eine Untersequenz des Schlüssels geeinigt werden muss, die anschließend auf Gleichheit überprüft wird. Die berechnete Fehlerrate wird nun noch für den Benutzer sichtbar ausgegeben.

Die vorangegangene Beschreibung der gewünschten Komponenten ist sehr allgemein gehalten, um die Möglichkeiten der Implementierung möglichst offen zu lassen. Im folgenden Kapitel wird jedoch die konkrete Umsetzung in der Entwicklungsumgebung .NET und der Programmiersprache C# ausführlich beschrieben.

4 Implementierung

Die konkrete Implementierung dieser Arbeit erfolgt mit Hilfe der Microsoft Windows Presentation Foundation (WPF) in der Entwicklungsumgebung Microsoft Visual Studio 2010 und mit Hilfe der Programmiersprache C#.
Bevor die implementieren einzelnen Programmteile folgen, muss hier zunächst der grundsätzliche Aufbau einer Komponente bzw. eines Plugins in CrypTool 2.0 beschrieben werden.

4.1 Aufbau eines Plugins

Um in CT2 eine neue Komponente zu erzeugen, muss ein vorgefertigter Projekttyp, ein sogenanntes "CrypTool 2.0 Plugin", in die vorhandene CT2-Arbeitsmappe eingefügt werden. Dies stellt das Grundgerüst für alle Komponenten dar, die später im Arbeitsflächeneditor auswählbar sind. Folgende Elemente sind in jedem Plugin vorhanden bzw. können eingebunden werden:

Die Hauptklasse

Die C# -Datei, welche sinnvollerweise den Namen des Plugins tragen sollte, stellt das Herzstück jedes Plugins dar. Sie erbt vom bereitgestellten Interface *IComponent* und in ihr werden die benötigten Variablen, Ein- und Ausgänge, sowie die gesamte Ausführungslogik implementiert.

Die Konfigurationsklasse

Eine C# -Datei, welche mit dem Namen des Plugins beginnen und auf das Wort "Settings" enden sollte, beinhaltet die Klasse mit jeglichen Einstellungsmöglichkeiten, die der Benutzer während der Laufzeit von CT2 vornehmen kann. Diese

Klasse muss vom Interface *ISettings erben*. Ein Objekt dieser Konfigurationsklasse wird in der Hauptklasse erzeugt und benutzt, um auf alle eingestellten Werte zugreifen zu können.

Die Dokumentationsdatei

In einer XML-Datei wird die gesamte Beschreibung und Anleitung für den Benutzer dargelegt. Sie beinhaltet eine Einführung, den grundsätzlichen Gebrauch des Plugins, Erklärungen zur Präsentation, die Beschreibung aller Ein- und Ausgänge, die in den Einstellungen veränderbaren Werte und eventuell weitere Referenzen auf die Komponente. Jegliche Dokumentation geschieht auf Deutsch und Englisch, abhängig davon, welche Sprache in den Optionen von CT2 eingestellt wurde.

Die Präsentationsdateien (optional)

Für eine optionale visuelle Komponentenpräsentation wird eine *.xaml*-Datei für die graphische Oberfläche benötigt, sowie eine weitere C# -Datei. Die in dieser befindliche Teilklasse erbt von der Klasse *UserControl* und kann somit vorgegebene Steuerelemente in einer Gruppe zusammenführen. Die grafische Zusammensetzung geschieht dabei in der von Microsoft entwickelten allgemeinen Auszeichnungssprache XAML (*Extensible Application Markup Language*) [xam] , die Steuerung der Elemente bzw. die Darstellungslogik in C#.

Die Ressourcendateien

Im Unterordner “Properties” befinden sich zwei Dateien des Typs *.resx*. Diese bestehen aus Einträgen im *Extensible Markup Language* (XML)-Format [xml], welche die Übersetzung des Programms in verschiedene Sprachen ermöglichen. Außerdem liegt die Datei *AssemblyInfo.cs* in diesem Ordner, welche Informationen über die erstellte Assembly erhält, wie den Namen, die Beschreibung und die Version.

Verschiedenes

Zu den genannten Elementen kommen noch Ressourcendateien, welche Übersetzungen von angezeigten Beschreibungen beinhalten, eventuell benutzte Grafiken oder andere Einbindungen, sowie die Verweise auf benutzte Bibliotheken. Außerdem können wie in den meisten Entwicklungsumgebungen Ordner zur besseren Strukturierung der Projekte selbst angelegt werden.

Eine ausführliche Anleitung für die Erstellung eines neuen Plugins und dessen Einbindung in CT2 wird online direkt auf der Website des Programms angeboten [ct2b]. Dies soll den Einstieg in die Mitarbeit an diesem Projekt für Anfänger vereinfachen.

4.1.1 Programmablauf eines Plugins

In der Klasse eines jeden Plugins sind bestimmte Methoden vorgegeben, die für alle Plugins implementiert werden müssen. Das folgende Flussdiagramm (vgl. Abbildung 4.1) zeigt den Ablauf dieser Methoden und die Bedingungen für ihre Ausführung.

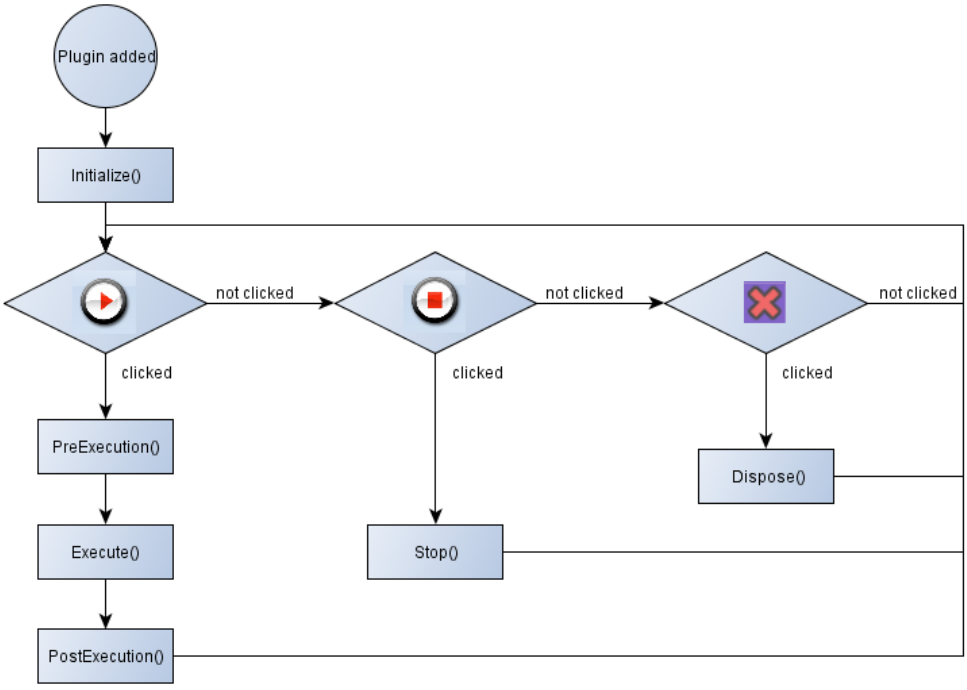

Abbildung 4.1: Ablauf eines CT2-Plugins

Wie zu sehen ist, wird nach der Initialisierung durch *Initialize()* der eigentliche Programmablauf durch die Methoden *PreExecution()*, *Execute()* und *PostExecution* beschrieben, sobald der Startknopf in CT2 betätigt wurde. Die Methoden *Stop()* und *Dispose()* werden jeweils bei manuellem Stoppen oder Löschen des Plugins ausgeführt.

4.2 Grundlegende Implementierungsdetails

Jedes Plugin, was in CT2 implementiert wird, stellt ein abgeschlossenes Programm dar. Die einzige Möglichkeit der Plugins zur Kommunikation untereinander sind ihre Konnektoren, also die Ein- und Ausgabeschnittstellen. Aus diesem Grund soll hier auf ein Klassendiagramm für jedes einzelne Plugin (bzw. jede Programmcode-Datei derselben) verzichtet und stattdessen auf Abbildung 3.2 zurückverwiesen werden. Die dort bereits beschriebenen Bausteine (orange) entsprechen genau den nun zu implementierenden Plugins, wobei die Pfeile die Verbindungen der jeweiligen Konnektoren darstellen, über die Informationen ausgetauscht werden. Es muss sich, wie bereits in Kapitel 3 angemerkt, vor einer Implementierung zunächst auf eine Datenstruktur für die Speicherung der Schlüssel, der Photonen sowie der Filterbasen geeinigt werden. In dieser Implementation wurden zur Lösung dieser Frage Strings benutzt, da diese eine Verkettung von einzelnen Zeichen (*Characters* oder *Chars*) darstellen, welche sich einfach den jeweiligen Elementen zuordnen lassen. Folgende Konvention wurde für die Chars festgelegt:

- Für Schlüssel
  - 0-Bit: ‘0’
  - 1-Bit: ‘1’
- Für Filterbasen
  - Plus-Basis: ‘+’
  - X-Basis: ‘x’
- Für Photonen
  - Vertikales Photon: ‘|’
  - Horizontales Photon: ‘-’

– Linksdiagonales Photon: ‘\’

– Rechtsdiagonales Photon: ‘/’

Im folgenden wird ein String, welcher ausschließlich aus den Chars für die Bits besteht, “Bit-String” genannt; für den “Photonen-String” und den ”Basis-String” gilt dasselbe. Eine weitere Notwendigkeit ist die Benutzung von Ressourcendateien für die bilinguale Umsetzung. Begriffe und Texte, die später bei der Programmausführung für den Benutzer sichtbar sind, werden im Programmcode nicht fest eingegeben. Stattdessen wird die Ressourcendatei benutzt, welche nach definierten Begriffen sucht, und sie in einen String auflöst, der für die derzeit eingestellte Sprache festgelegt wurde. Aus Gründen der Übersicht wird in allen hier beschriebenen Komponenten die Zeichenfolge “res_ ” vor eine so aufgelöste Ressource gestellt.
Des weiteren wurde besonderer Wert auf eine saubere Struktur aller Programme gelegt. Dies betrifft zum größten Teil die ordnungsgemäße Kapselung von Programmteilen mit gleichem Abstraktionslevel und die sinnvolle Benennung der erstellten Methoden. Jeder Name einer Methode sollte dem Betrachter des Codes idealerweise direkt deren Funktion offenbaren.

4.2.1 Graphische Darstellung

In den Präsentationen, die hier ebenfalls beschrieben werden, müssen bestimmte Grafiken für die Repräsentation der physikalischen Gegebenheiten dienen. Abbildung 4.2 zeigt alle verwendeten Grafiken.

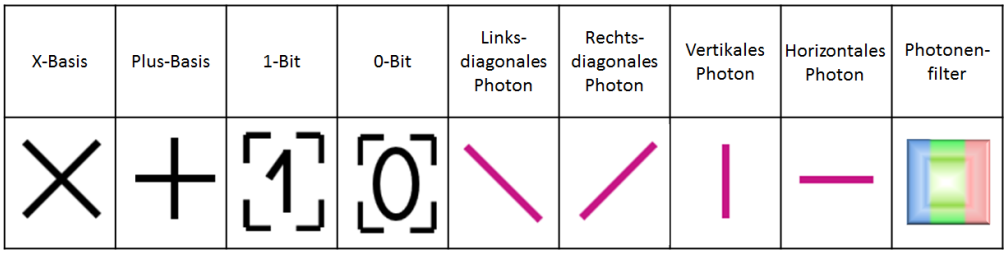

Abbildung 4.2: Verwendete Grafiken

Zusätzlich wird noch ein Hintergrundbild verwendet, damit sich die Elemente deutlicher von der Oberfläche abheben.

4.3 Implementierte Plugins

Theoretisch wäre es möglich, alle in Kapitel 3 beschriebenen Komponenten in CT2 in einem gemeinsamen Plugin zu vereinen, dass die vollständige Funktion des BB84-Protokolls beherrscht. In dieser Arbeit ist jedoch entschieden worden, die Komponenten jeweils als eigene Plugins auszulagern, damit der Benutzer zum Einen die einzelnen Schritte und Schnittstellen besser versteht, zum Anderen aber auch sehr viel mehr Freiraum in der Gestaltung hat. Er kann Änderungen am Aufbau vornehmen und somit das Protokoll verändern oder ein ganz neues schaffen. Daher müssen alle Komponenten als jeweils eigenes Projekt in die Visual-Studio-Arbeitsmappe von CT2 eingebunden werden. Aus Gründen der Auffindbarkeit soll der Begriff "BB84" vor jeden Namen hinzugefügt werden. Folgende Plugins wurden erstellt (deutscher und englischer Name):

- BB84-Photonenbasis-Generator (BB84 Photonbase Generator)
- BB84-Photonenkodierer (BB84 Photon Encoder)
- BB84-Mittelsmann (BB84 Man in the Middle)
- BB84-Photonendekodierer (BB84 Photon Decoder)
- BB84-Schlüsselgenerator (BB84 Key Generator)
- BB84-Fehlerdetektor (BB84 Error Detector)

Es folgt eine Beschreibung der wichtigsten Bestandteile dieser Plugins. Die *Execute()*-Methode nimmt dabei einen besonderen Stand ein, da sie ausschließlich die höchste Abstraktionsebene des Programms widerspiegelt, und wird deshalb stets mit aufgeführt. Da in vier der Plugins auch eine Präsentation eingebaut wurde, wird diese ebenfalls erläutert.

4.3.1 Plugin: BB84-Photonenbasis-Generator

Dieses Plugin stellt die Implementierung des Photonenbasis-Generators dar. Es erzeugt einen String aus zufällig gewählten Photonenbasen. Die Länge dieses Strings kann dabei vom Benutzer auf verschiedene Arten gewählt werden.

4.3.1.1 Funktionsweise

Textauszug 4.1: Execute()-Methode des BB84-Photonenbasis-Generator

```
1 public void Execute()
2 {
3   ProgressChanged(0, 100);
4   generateRandomBases();
5   notifyAllOutputs();
6   ProgressChanged(100, 100);
7 }
```

Die *ProgressChanged(...)*-Methoden werden in allen Plugins zur Darstellung des aktuellen Rechenfortschritts benutzt, *notifyAllOutputs()* ist eine selbst definierte Methode zur Weitergabe des Ergebnisses (in diesem Fall des Basis-Strings) an die Ausgabekonnektoren. Die eigentliche Erzeugung des Strings geschieht in der Methode *generateRandomBases()*. Dazu benötigt das Plugin eine Eingabe für die Länge des zu erzeugenden Basis-Strings (gespeichert in der Integer-Variable *keyLength*). Da diese Größe jedoch nicht universell für alle Plugins festgelegt ist, sondern von dem eingegebenen Schlüssel abhängt, ist es sinnvoll, neben der Einstellungsmöglichkeit in den Plugin-eigenen Settings, mehrere Möglichkeiten als Eingabetyp zuzulassen, um dem Benutzer eine Wahl zu lassen, wie er die Anzahl der zu bestimmenden Basen eingeben möchte. So kann diese Komponente eventuell auch in späteren Projekten auf eine andere Art genutzt werden.
Textauszug 4.2 zeigt, wie diese Möglichkeiten implementiert wurden.

Textauszug 4.2: Bestimmung der Länge des Basis-Strings

```
1 //...
2 if (inputKey is BigInteger)
3 {
4   BigInteger temp = (BigInteger)inputKey;
5   this.keyLength = Int32.Parse(temp.ToString());
6 }
7 else if (inputKey is string)
8 {
9   string temp = (string)inputKey;
```

```
10    temp = filterValidInput(temp);
11    this.keyLength = temp.Length;
12  }
13  else if (inputKey is int[]){
14    int[] temp = (int[])inputKey;
15    this.keyLength = temp.Length;
16  }
17  else if (inputKey is char[])
18  {
19    char[] temp = (char[])inputKey;
20    this.keyLength = temp.Length;
21  }
22  else
23  {
24    this.keyLength = (inputKey.ToString().Length);
25  }
26  //...
```

Mögliche Eingaben sind also Integer, Strings, Integer-Arrays und Char-Arrays. Bei einem Integer wird die eingegebene Zahl direkt als Wert für *keyLength* übernommen, bei einem String wird die Länge dieses String gewählt und bei Integer- oder Char-Arrays die Anzahl der in dem Array befindlichen Werte.
Zusätzlich müssen bei der Eingabe eines Strings alle Zeichen, welche sich nicht in der weiter oben genannten Konvention befinden, herausgefiltert werden (in Textauszug 4.2 durch die Methode *filterValidInput()* realisiert).
Dies stellt sicher, dass bei falschen Eingaben für diese nicht ebenfalls Basen erzeugt werden.

4.3.2 Plugin: BB84-Photonenkodierer

Dieses Plugin repräsentiert den Photonenkodierer. Es erzeugt mit Hilfe eines Basis-Strings und eines Bit-Strings über eine Kodierungstabelle einen Photonen-String und gibt diesen anschließend aus.

4.3.2.1 Funktionsweise

Textauszug 4.3: Execute-Methode des BB84-Photonenkodierer

```
1 public void Execute()
2 {
3   ProgressChanged(0, 1);
4   encodeKeyIntoPhotons();
5   startPresentationIfVisible();
6   notifyOutput();
7   ProgressChanged(1, 1);
8 }
```

Die Methode *startPresentationIfVisible()* sorgt für das Starten der Präsentation, wenn diese vom Benutzer sichtbar gestellt wurde, in *encodeKeyIntoPhotons()* geschieht die eigentliche Veränderung des Strings, daher soll auf diese hier noch genauer eingegangen werden. Textauszug 4.4 zeigt einen Ausschnitt des Quellcodes.

Textauszug 4.4: encodeKeyIntoPhotons()-Methode des Photonenkodierers

```
1 private void encodeKeyIntoPhotons()
2 {
3   StringBuilder tempOutput = new StringBuilder();
4
5   for (int i = 0; i < inputKey.Length; i++)
6   {
7     if (inputBases.Length > i && inputKey.Length > i)
8     {
9       if (inputBases[i].Equals('+'))
10      {
11        if (inputKey[i].Equals('0'))
12        {
13          tempOutput.Append(getPlusBasePhoton(settings.
              PlusZeroEncoding));
14        }
15        else if (inputKey[i].Equals('1'))
16        {
```

```
17              tempOutput.Append(getPlusBasePhoton(settings.
                    PlusOneEncoding));
18            }
19          }
20          else if (inputBases[i].Equals('x'))
21          {
22            // [...]
23          }
24        }
25        ProgressChanged(i/(inputKey.Length-1), 1);
26      }
27      this.photonOutput = tempOutput.ToString();
28  }
```

Alle Zeichen des Eingabeschlüssels (*inputKey*) werden durchlaufen, es wird die jeweils dazugehörige Basis geprüft und je nachdem, ob es sich um eine Plus- oder X-Basis handelt, wird der dazugehörige Char, welcher die Photonen repräsentiert, ausgewählt. Die Kodierungstabelle wird dabei von den beiden Methoden *getPlusBasePhoton(...)* (vgl. Zeilen 13, 17) und *getXBasePhoton(...)* (vgl. Zeilen 24, 28) dargestellt. Wie zu sehen ist, wird diesen beiden Methoden jedoch kein fester Wert übergeben, sondern es wird aus den Einstellungen die jeweils festgelegte Kodierung abgerufen (*settings.PlusZeroEncoding, settings.PlusOneEncoding, settings.XZeroEncoding, settings.XOneEncoding*). Somit kann die Kodierungstabelle vom Benutzer jederzeit in den Einstellungen verändert werden.
Außerdem ist erwähnenswert, dass zur Erstellung des Ausgabestrings *tempOutput* ein StringBuilder verwendet wurde, was im Gegensatz zur Verkettung von Strings mittels "++-Operator" die Geschwindigkeit des Programmes deutlich erhöht.

4.3.2.2 Die Präsentation

Der BB84-Photonenkodierer ist das erste der vier Plugins, welche durch eine Präsentation erweitert werden. Abbildung 4.3 zeigt die Ansicht des Plugins im Präsentationsmodus.

Das graue, längliche Rechteck symbolisiert hier den Photonenkanal, der am rechten Rand endet und in den anderen Plugins weitergeführt wird. An der linken Seite befindet sich eine Liste der nächsten drei Bits, am unteren Rand eine der letzten

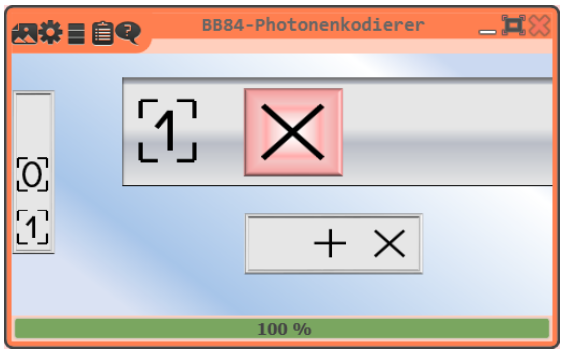

Abbildung 4.3: Präsentation des BB84-Photonenkodierer

drei Basen. Dies simuliert eine Warteschlange, die sich nach und nach abbaut. Die Basen gelangen dabei in die rote, quadratische Vorrichtung in der Mitte, welche den Lichtfilter repräsentiert. Die Bits bewegen sich von der linken Seite an diesen Filter und werden ausgeblendet. Anschließend wird die Basis ebenfalls ausgeblendet und stattdessen erscheint das Photon, das zu verschicken ist, und bewegt sich zum rechten Rand heraus. Dies wird für alle Bits, Basen und Photonen wiederholt.
Realisiert wurde diese Animation in der Hauptsache durch die Benutzung einer WPF-Klasse, dem *Storyboard*. Storyboards stellen eine Art Container dar, die Objekt- und Eigenschaftsinformationen für die ihnen untergeordneten Animationen bereitstellen. Hierzu muss zunächst das Storyboard in der XAML-Datei mit allen dazugehörigen Animationen definiert werden. Abbildung 4.5 zeigt ein Beispiel einer solchen Definition aus der *BB84PhotonEncoderPresentation.xaml*.

Textauszug 4.5: Beispiel für die Definition eines Storyboards

```
1 <Storyboard x:Key="movementRight" SpeedRatio="0.6" >
2   <DoubleAnimation From="0" To="160" AutoReverse="False
      " RepeatBehavior="1x" Storyboard.TargetProperty="X
      " Storyboard.TargetName="rightVertical" Completed=
      "completedMovementRight"></DoubleAnimation>
3   <!--[...]-->
4 </Storyboard>
```

Der auskommentierte Bereich erhält noch weitere Animationen, die alle unter demselben Storyboard eingeordnet sind. Um die Animation zu kontrollieren, müssen bestimmte Methoden aus der entsprechenden Hauptklasse in der C# -Datei (in

diesem Fall also *BB84PhotonEncoderPresentation.cs* aufgerufen werden. Die wichtigsten Steuermethoden, deren Namen selbsterklärend sind, lauten:

- Begin()
- Pause()
- Resume()
- Stop()

Die erstellte Animation lässt sich dabei in 3 Phasen aufgeteilt, welche folgendes beinhalten:

Phase I

- Bewegung der Basis-Grafik in den Filter
- Vergrößerung der Basis-Grafik
- Bewegung der Bit-Grafik zum Filter
- Vergrößerung der Bit-Grafik

Phase II

- Ausblendung der Basis-Grafik
- Ausblendung der Bit-Grafik
- Bewegung der Bit-Warteschlange
- Bewegung der Basis-Warteschlange

Phase III

- Bewegung der Photon-Grafik

Der Ablauf dieser Animationsphasen wird in Abbildung 4.4 beschrieben.

Die Animation kann auf zwei Wegen zum Ende kommen: per Abbruch durch den Benutzer oder weil alle Iterationen ausgeführt wurden. In beiden Fällen muss dafür gesorgt werden, dass die Elemente im Plugin ausgeblendet werden. Dies geschieht, indem das Attribut *Visibility* der Hauptzeichenfläche *mainCanvas* auf *Hidden* gesetzt wird. Bei erneutem Start muss es entsprechend wieder auf *Visible* geändert werden.

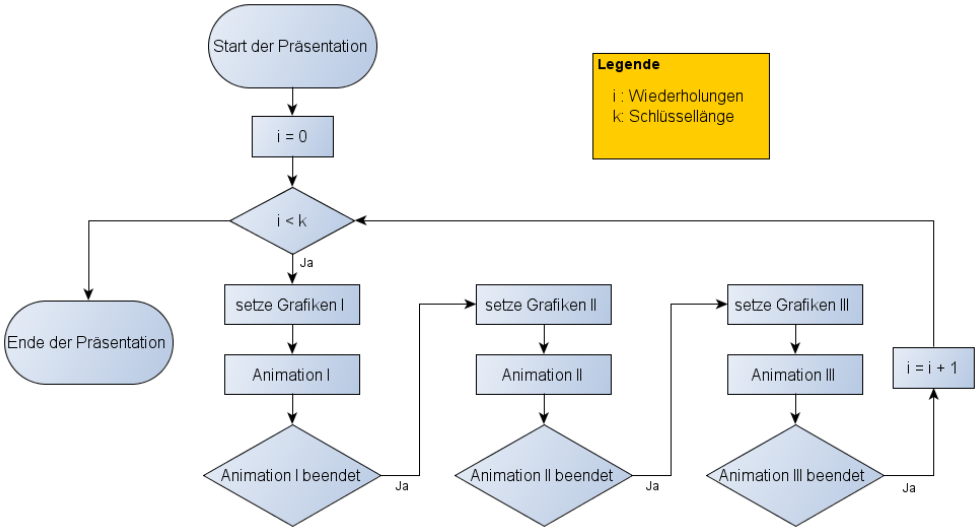

Abbildung 4.4: Aufbau der Präsentation des BB84-Photonenkodierer

4.3.3 Plugin: BB84-Mittelsmann

Der angreifende Mittelsmann wird mit diesem Plugin dargestellt. Anhand eines Photonen-Strings und eines Basis-Strings wird ein Bit-String erzeugt. Der Photonenstring wird dabei eventuell verändert und anschließend zusammen mit dem Bit-String ausgegeben.

4.3.3.1 Funktionsweise

Das Plugin besitzt eine Integer-Variable *IsListening*, welche beschreibt, ob es Aktiv oder Inaktiv geschaltet ist. Die Variable wird in der Einstellungsdatei gespeichert und kann vom Benutzer über die Konfigurationsoberfläche per Dropdown-Menü eingestellt werden. Ist das Plugin aktiv (Variablenwert: 1), dekodiert es die empfangenen Zeichen über einen Basis-String, den es von einem BB84-Photonenbasis-Generator-Plugin erhält. Den so abgefangenen Schlüssel gibt es schließlich über einen Ausgang aus, den veränderten Photonen-String über einen zweiten.
Ist der Modus des Plugins auf Inaktiv gestellt (Variablenwert: 0), wird der Photonen-String empfangen und lediglich über den zweiten Ausgang unverändert weitergegeben. Über den Ausgang für den empfangenen Schlüssel wird dann eine Meldung über die Inaktivität des Plugins ausgegeben.

Textauszug 4.6: Execute()-Methode des BB84-Mittelsmann

```
1 public void Execute()
2 {
3   ProgressChanged(0, 1);
4   if (settings.IsListening == 0)
5   {
6     decodeIncomingPhotons();
7     forwardListenedPhotons();
8   }
9   else
10  {
11    forwardReceivedPhotons();
12    displaySleepMessage();
13  }
14  notifyOutputs();
15  startPresentationIfVisible();
16  ProgressChanged(1, 1);
17 }
```

Der eigentliche Dekodierungsvorgang ist hier derselbe wie im später noch beschriebenen BB84-Photonendekodierer. Da das Hauptaugenmerk bei diesem Plugin aber auf der unumgänglichen Veränderung an den dekodierten Photonen liegt, soll hier noch der Fall beschrieben werden, wenn ein Photon mit einer nicht passenden Basis gemessen werden soll. Hierzu wurde die Methode *getRandomBit()* definiert, die ein zufälliges Bit erzeugen soll.

Textauszug 4.7: getRandomBit()-Methode des BB84-Mittelsmann

```
1 private string getRandomBit()
2 {
3   sRandom = new RNGCryptoServiceProvider();
4   byte[] buffer = new byte[4];
5   sRandom.GetBytes(buffer);
6   int result = BitConverter.ToInt32(buffer, 0);
7   return ""+new Random(result).Next(2);
8 }
```

Zur Erzeugung der Zufallszahl wurde hier absichtlich nicht die häufig verwendete *System.Random*-Klasse, sondern der RNGCryptoServiceProvider verwendet, da hier ein quantenmechanisches Phänomen simuliert werden soll, und somit ein möglichst starker und unberechenbarer Zufallsgenerator wünschenswert ist.

4.3.3.2 Die Präsentation

Das BB84-Mittelsmann-Plugin besitzt ebenfalls eine Präsentation, die sich optional zwischen die Präsentationen vom BB84-Photonenkodierer und dem BB84-Photonendekodierer setzen lässt.

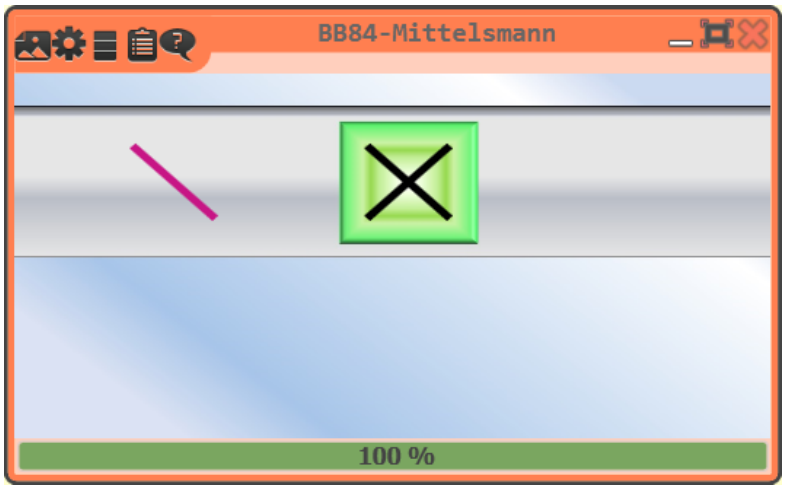

Abbildung 4.5: Präsentation des BB84-Mittelsmann

Man sieht, dass der in grau dargestellte Photonenkanal hier vom linken bis zum rechten Bildschirmrand läuft, sodass bei Benutzung aller drei Präsentationen der Anschein erweckt wird, ein Photon liefe durch einen verlängerten Kanal. Anders als in den anderen Präsentationen, ist der Verlauf dieser hier jedoch abhängig von der Einstellung des Plugins: ist der Mittelsmann auf *inaktiv* gestellt, so muss nur die Animation eines Photons dargestellt werden, das komplett durch den Kanal läuft, wozu ein einzelnes Storyboard ausreicht. Ist die Einstellung jedoch auf *aktiv* gesetzt, wird das Photon in die in der Mitte befindliche Basis bewegt und verdoppelt. Eine Kopie bewegt sich danach zum rechten, eine zum oberen Rand, was das Mithören des Mittelsmann symbolisiert. Triff durch eine falsche Basis ein Messfehler auf, wird auch das falsche Photon so kopiert und weiterbewegt. Ähnlich wie beim Photonenkodierer müssen insgesamt drei Phasen durchlaufen werden:

Phase I

- Bewegung der Photon-Grafik zum Filter

Phase II

- Ausblendung der Photon-Grafik
- Ausblendung der Basis-Grafik

Phase III

- Bewegung der Photon-Grafik nach oben
- Bewegung der Photon-Grafik nach rechts

Außerdem ist zu beachten, dass die Animation dieses Plugins nicht gleichzeitig mit der des Photonenkodierers starten darf, da sonst nicht die Illusion einer Weitergabe der Photonen gegeben ist. Die Animation muss also mindestens einen kompletten Zyklus abwarten, bevor sie beginnt. Da es aber auch Fälle geben kann, in denen mehrere Wartezyklen erforderlich sind, wie in etwas das in Reihe schalten mehrerer Mittelsmänner, kann der Benutzer in den Einstellungen die entsprechende Zahl selbst eingeben.

4.3.4 Plugin: BB84-Photonendekodierer

Dieses Plugin stellt den Photonendekodierer des Protokolls dar. Es bekommt einen Photonen-String und einen Basis-String und erzeugt anhand seiner Einstellungen einen entsprechenden Bit-String. Abhängig von der Variable *ErrorsEnabled*, welche durch die Benutzereinstellung verändert werden kann, wird eine normale Dekodierung durchgeführt oder eine, bei der Übertragungsfehler auftreten.

4.3.4.1 Funktionsweise

Textauszug 4.8: Execute()-Methode des BB84-Photonendekodierer

```
1 public void Execute()
2 {
3   ProgressChanged(0, 1);
4   if (settings.ErrorsEnabled == 0)
```

```
5   {
6     doNormalDecoding();
7   }
8   else
9   {
10    doDecodingWithErrors();
11  }
12  OnPropertyChanged("OutputKey");
13  startPresentationIfVisible();
14  ProgressChanged(1, 1);
15 }
```

Die fehlerfreie Dekodierung in *doNormalDecoding()* erfolgt ähnlich wie bei der Kodierung durch einen Schleifendurchlauf über die eingegangenen Photonen-Chars. Wenn zu einem dieser Chars ein passender Basis-Char vorliegt, wird in der Einstellungsklasse der entsprechende Bit-Char herausgesucht und im Ausgabestring abgelegt. Passt eine Basis nicht, so wird stattdessen ein zufälliges Bit gewählt, ganz wie zuvor im BB84-Mittelsmann beschrieben. In der fehlersimulierenden Methode *doDecodingWithErrors()* jedoch werden abhängig von einer durch die Einstellungen festgelegten Fehlerrate auch bei passenden Paaren von Photonen und Basen zufällig Fehler verursacht. Der Codeausschnitt in Textauszug 4.9 zeigt, wie im Durchschnitt jedes n-te Bit zufällig erzeugt wird, wobei n dem Wert der Variable *errorRatio* multipliziert mit der Anzahl der Photonen-Chars entspricht.

Textauszug 4.9: Zufällige Fehler im BB84-Photonendekodierer

```
1 // [...]
2 int fail = (int)Math.Round(inputPhotons.Length *
      errorRatio);
3 for (int i = 0; i < inputPhotons.Length; i++)
4 {
5   if (i % fail == 0)
6   {
7     tempOutput.Append(getRandomBinary());
8   }
9 // [...]
```

4.3.4.2 Die Präsentation

Die Präsentation des BB84-Photonendekodierer-Plugins beendet die Reihe der scheinbar ineinander übergehenden Animationen.

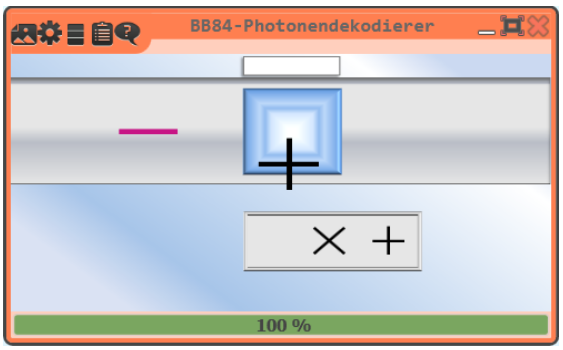

Abbildung 4.6: Präsentation des BB84-Photonendekodierer

Der graue Photonenkanal endet in diesem Fenster und die Animation geschieht in etwa gegenteilig zum BB84-Photonenkodierer: Photonen bewegen sich von der linken Seite in das den Filter repräsentierende Rechteck, gleichzeitig wird eine Basis aus der Warteschlange am unteren Rand in diesen hineingelegt. Beide Grafiken verblassen und es erscheint das Bit, das dekodiert wurde, und bewegt sich zum rechten Rand hin hinaus. Zusätzlich existiert ein weißes Feld über dem Filter, in welchem rote Blitzsymbole aufblinken, sobald ein Photon mit einer falschen Basis gemessen wird. Dies ist zwar für den Benutzer des Dekodierers eigentlich nicht ersichtlich, soll dem Benutzer des Plugins aber darauf hinweisen, dass das korrespondierende Bit später aus dem Schlüssel gestrichen werden muss (siehe Abbildung 4.7).

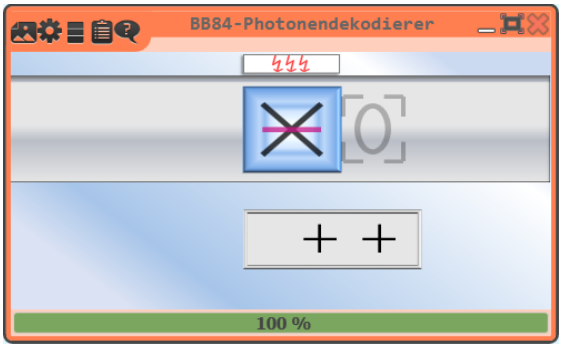

Abbildung 4.7: Fehlerdarstellung bei falscher Basis

Somit ergeben sich folgende drei Animationsphasen für dieses Plugin:

Phase I

- Bewegung der Basis-Grafik in den Filter
- Vergrößerung der Basis-Grafik
- Bewegung der Photon-Grafik zum Filter

Phase II

- Ausblendung der Basis-Grafik
- Ausblendung der Photon-Grafik
- Bewegung der Basis-Warteschlange
- Einblendung der Bit-Grafik
- Optional: Einblendung und Blink-Animation der Blitze

Phase III

- Bewegung der Bit-Grafik

Auch in diesem Plugin muss sich die Anzahl der Wartezyklen einstellen lassen, da sonst keine Synchronität mit den anderen Präsentationen garantiert ist.
Benutzt man alle Präsentationen zugleich, so entsteht der Eindruck einer großen, kontinuierlichen Animation, wie Abbildung 4.8 zeigt.

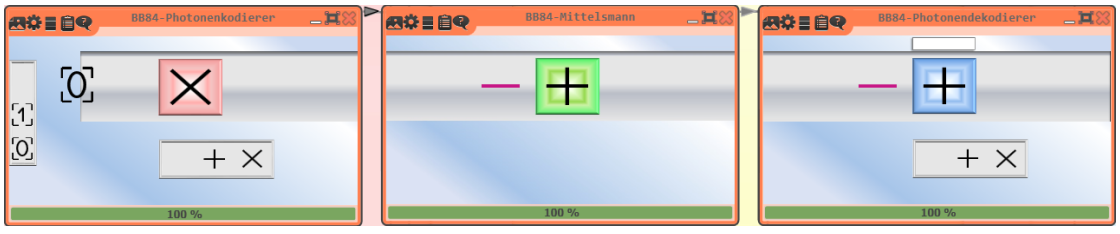

Abbildung 4.8: Vollständige Animation des Photonenflusses

Der vollständige Vorgang von Kodierung, Angriff und Dekodierung ist nun simuliert.

4.3.5 Plugin: BB84-Schlüsselgenerator

Dieses Plugin, das den Schlüsselgenerator repräsentiert, erhält als Eingabe einen Bit-String und zwei Basis-Strings. Hiermit erzeugt es einen neuen Bit-String, welcher lediglich die Stellen des alten beinhaltet, deren entsprechenden Stellen in beiden Basis-Strings übereinstimmen.

4.3.5.1 Funktionsweise

Textauszug 4.10: Execute()-Methode des BB84-Schlüsselgenerator

```
1 public void Execute()
2 {
3   ProgressChanged(0, 1);
4   generateCommonKey();
5   showPresentationIfVisible();
6   notifyOutput();
7   ProgressChanged(1, 1);
8 }
```

Da die Funktionsweise der Methode *generateCommonKey()* bereits in Textauszug 3.1 durch Pseudocode beschrieben wurde, soll sie hier nicht im Detail wiederholt werden, sondern stattdessen genauer auf seine Präsentation eingegangen werden.

4.3.5.2 Die Präsentation

Anders als die bisherigen Präsentationen ist die des Schlüsselgenerators nicht Teil einer kontinuierlichen Bewegung, sondern stellt eine Darstellung für die Schlüsselerzeugung dar, die unabhängig von den anderen Animationen eingeschaltet werden kann.

Beim Start der Präsentation wird zuerst die eingegebene Bitfolge angezeigt, anschließend die beiden Folgen von Basen tabellarisch darunter. Danach werden die Spalten, in denen die gleiche Basis existiert, grün eingefärbt und darunter entsteht eine Ergebniszeile mit den entsprechenden Stellen der Bitfolge, die nun den gemeinsamen Schlüssel darstellen. Die Darstellung der Strings erfolgt mit Hilfe von TextBlock-Elementen, die von WPF zu Verfügung gestellt werden. Für die Animation ergeben sich folgende Phasen:

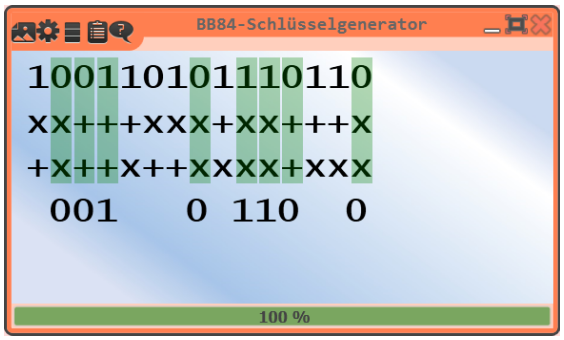

Abbildung 4.9: Präsentation des BB84-Schlüsselgenerator

Phase I

- Einblendung der Bitfolgen-Grafik

Phase II

- Einblendung der ersten Basisfolgen-Grafik

Phase III

- Einblendung der zweiten Basisfolgen-Grafik

Phase IV

- Einblendung der Spaltenmarkierungen

Phase V

- Einblendung der Schlüsselfolge **oder**
- Einblendung der Fehlermeldung

In den Einstellungen lässt sich die Animationsgeschwindigkeit anhand eines Slider-Elementes verändern, weitere Einstellungsmöglichkeiten sind nicht erforderlich. Sollte der (zumindest bei großer Bitzahl) unwahrscheinliche Fall auftreten, dass keine gemeinsamen Basen gefunden wurden, so wird dies durch eine Meldung im Plugin ausgegeben (sieh Abbildung 4.10. Der ausgegebene Schlüssel ist dann leer.

Abbildung 4.10: Meldung bei keinen gemeinsamen Basen

4.3.6 Plugin: BB84-Fehlerdetektor

Das letzte Plugin beschreibt den Fehlerdetektor des Protokolls. Es erhält zwei Bit-Strings als Eingabe und bestimmt eine Sequenz der beiden, welche durch Anfangs- und Endwert in den Einstellungen festgelegt ist. Dann berechnet es den Grad der Übereinstimmung zwischen diesen beiden Sequenzen, prüft, ob dies die eingestellte Fehlergrenze überschreitet und gibt sein Ergebnis als String aus.

4.3.6.1 Funktionsweise

Textauszug 4.11: Execute()-Methode des BB84-Fehlerdetektor

```
1 public void Execute()
2 {
3   ProgressChanged(0, 1);
4   setSequenceBounds();
5   calculateErrorRatio();
6   generateDetectionMessage();
7   notifyOutput();
8   ProgressChanged(1, 1);
9 }
```

Die Methode *setSequenceBounds()* ist zur Fehlervermeidung wichtig. Sie überprüft, ob die Grenzen für die Prüfungssequenz außerhalb des prüfbaren Bereiches liegen (also zum Beispiel eine Sequenz von Zeichen 5 bis 30 bei einem Bit-String von 20 Zeichen). Der selbe Fall tritt auch auf, wenn einer der Bit-Strings kürzer ist, als der andere (obwohl bei korrekter Benutzung aller Plugins dieser Fall nicht möglich

ist). Der fehlerhafte Grenzwert wird dann automatisch beim Start des Plugins auf die Länge des kürzesten Strings gesetzt.
Die wichtigste Methode des Plugins ist jedoch *calculateErrorRatio()*, da in ihr die eigentliche Bestimmung der Fehlerrate stattfindet. Textauszug 4.12 zeigt ihren Aufbau.

Textauszug 4.12: Berechnung der Fehlerrate

```
1  private void calculateErrorRatio()
2 {
3   double count = 0;
4   double errors = 0;
5   for (int i = mySettings.StartIndex; i <= mySettings.
       EndIndex; i++)
6   {
7     if (firstKey.Length > i && secondKey.Length > i)
8     {
9       if (!firstKey[i].Equals(secondKey[i]))
10      {
11        errors++;
12      }
13    }
14    count++;
15  }
16  errorRatio = errors/count;
17 }
```

Über beiden Strings wird in den zuvor gesetzten Grenzen iteriert, die Iterationen sowie die ungleichen Stellen werden in Variablen festgehalten und zum Schluss wird die Fehlerrate anhand dieser Variablen berechnet.
In der letzten der drei Hauptmethoden *generateDetectionMessage()* muss nun nur noch überprüft werden, ob die Fehlerrate den eingestellten Grenzwert überschritten hat. Abhängig davon wird eine Warnung oder Entwarnung als String ausgegeben.
Die Implementierung einer Präsentation hat sich für den BB84-Fehlerdetektor nicht angeboten, da der bloße Vergleich zweier Schlüssel und die Berechnung ihrer Abweichung ein intuitiv verständlicher Vorgang ist, der keiner Visualisierung

bedarf. Außerdem muss beachtet werden, dass nicht zu viele Animationen vorhanden sind, um den Benutzer nicht zu überfordern sondern das Augenmerk auf die wichtigsten Vorgehensweisen des Protokolls zu lenken.

4.4 Die Vorlagen

Alle notwendigen Plugins wurden nun implementiert und können vom Benutzer in der gewünschten Reihenfolge aneinandergereiht werden. Damit jedoch auch für Laien die Funktionsweise des BB84-Protokolls offen gelegt wird, sollte es vorgefertigte Vorlagen geben, die den vollständigen Ablauf in bereits zusammengesetzter Form beinhalten. Da der Angriff eines Mittelsmann nicht explizit Inhalt des eigentlichen Protokolls ist, soll eine Vorlage mit dem reinen Schlüsseltausch und eine mit zusätzlichem Angriff und folgender Sicherheitsprüfung erstellt werden.

4.4.1 Vorlage: "BB84-Schlüsselaustausch"

Diese Vorlage beschreibt den reinen Schlüsselaustausch zwischen den zwei Parteien Alice und Bob. Nach dem Start von CT2 kann der Benutzer es aus dem Katalog der Vorlagen sofort abrufen. Abbildung 4.11 zeigt einen Screenshot bei seiner Ausführung.

Alle für diese Vorlage benötigten Plugins wurden in der richtigen Reihenfolge verbunden und, sofern vorhanden, in den Präsentationsmodus gestellt. Anschließend wurden farbige Hintergrundbilder eingefügt, um deutlich zu machen, welchen Parteien welche Plugins zugeordnet wurden. Auch eine Legende ist vorhanden, welche die verschiedenen Symbole erklärt, die in den Präsentationen zu sehen sind.

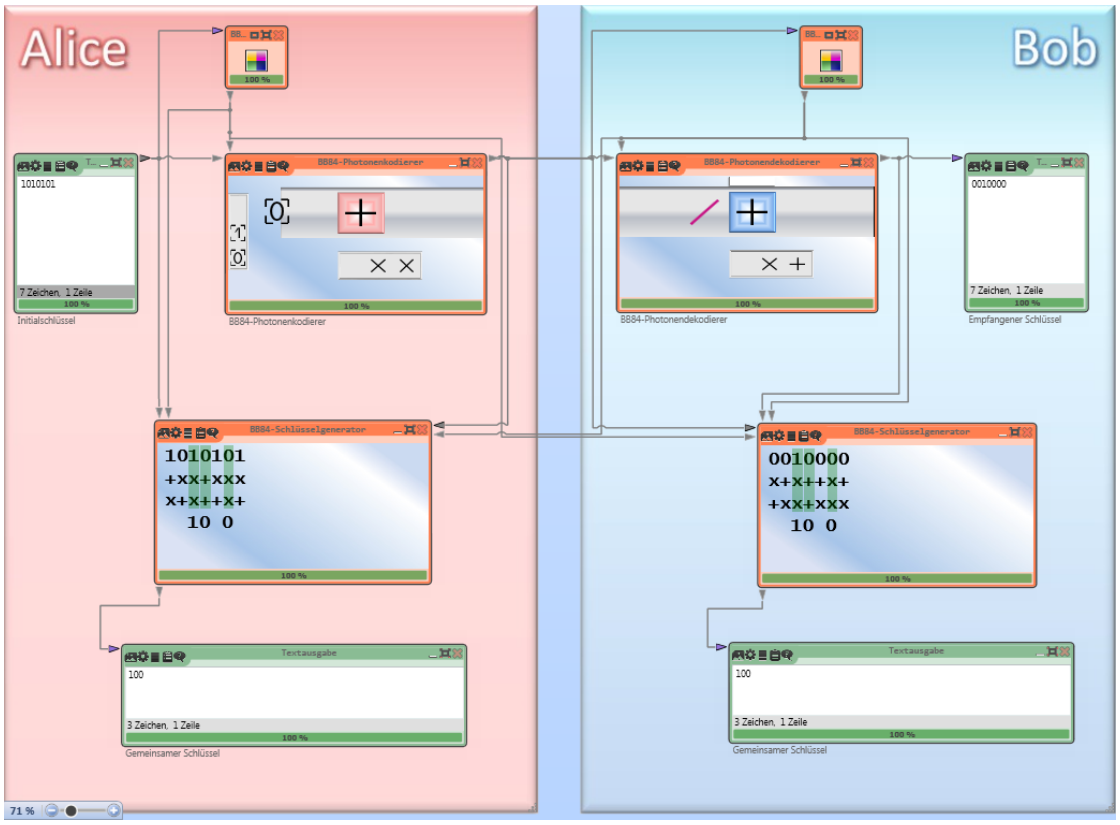

Abbildung 4.11: BB84-Schlüsselaustausch

4.4.2 Vorlage: "BB84-Schlüsselaustausch mit Lauschangriff"

Diese Vorlage beschreibt einen Schlüsseltausch von Alice und Bob, welcher von einem Mittelsmann (Eve) abgehört wird. Anschließend wird die Sicherheit des Schlüssels überprüft.

Auch hier sind alle Plugins bereits aufgebaut und verbunden. Der BB84-Mittelsmann steht standardmäßig auf "aktiv" und der BB84-Fehlerdetektor ist auf die Überprüfung des Teilstrings von den Indizes 0 bis 100 eingestellt. Die Legende wurde ebenfalls wieder in die Vorlage übernommen, ein Hilfetext am unteren Rand beschreibt das gesamte Protokoll in seiner Funktionsweise.

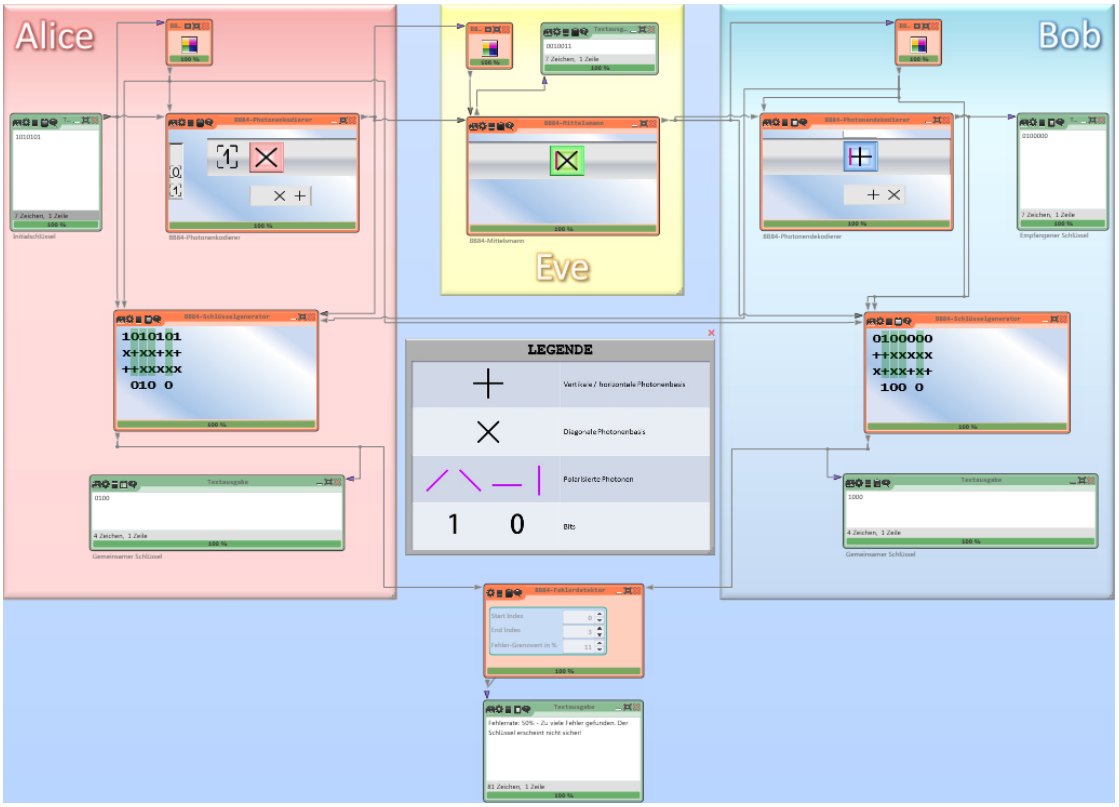

Abbildung 4.12: BB84-Schlüsselaustausch mit Lauschangriff

5 Sicherheit und Schwachstellen

Da die Implementierung nun abgeschlossen ist, soll im folgenden Kapitel noch auf verschiedene Aspekte der Sicherheit eingegangen werden. Es werden Stärken und Schwächen des BB84-Protokolls aufgeführt, weitere Angriffsmöglichkeiten genannt und die Unterschiede von einer echten Durchführung zur Computersimulation aufgezeigt.

5.1 Stärken des BB84-Protokolls

Um die Sicherheit des Schlüsselaustausches zu beschreiben, muss unbedingt zwischen theoretischer und praktischer Anwendung des Protokolls unterschieden werden.

5.1.1 Theoretische Anwendung

Bei rein theoretischer Betrachtung ist die Sicherheit des Schlüssels im Fall eines Lauschangriffes durch eine einzige Variable definiert: der Wahrscheinlichkeit eines verursachten Fehlers durch den Mittelsmann. Wie in Kapitel 3 beschrieben, wird statistisch gesehen bei 25% aller Photonen, die vom Mittelsmann abgefangen werden, ein Fehler im gemeinsamen Schlüssel erzeugt. Da schon ein einziger Fehler ausreicht, um den Angreifer zu enttarnen, berechnet sich die Erfolgswahrscheinlichkeit p_s (s für "success") abhängig von der Länge des gemeinsamen Schlüssels l wie folgt:

$$p_s(l) = 0,75^l \tag{5.1}$$

So ist zu sehen, dass zwar bei kleiner Schlüssellänge durchaus Chancen bestehen, dass der Angriff unbemerkt bleibt, bei Schlüsseln mit angemessen großen Längen,

wie etwa 128 Bit für heutige symmetrische Verfahren, sinkt die Wahrscheinlichkeit jedoch rapide:

$$p_s(128) = 0,75^{128} \approx 1,02 \cdot 10^{-16} \tag{5.2}$$

Allerdings besteht außerdem die Möglichkeit, dass ein Fehler zwar aufgetreten ist, von Alice und Bob jedoch nicht bemerkt wurde, da der zur Fehlerüberprüfung ausgetauschte Teilschlüssel keinen Fehler enthält. Die genannte Formel muss also auf die Länge des ausgetauschten Teilschlüssels angewandt werden, um die endgültige Erfolgswahrscheinlichkeit eines Angriffs zu bestimmen. Daher ist es dringend zu empfehlen, den eigentlichen Schlüsselaustausch mit sehr viel größerem Schlüssel zu beginnen, um dann im Teilschlüssel immer noch eine ausreichend große Menge an Bits zur Verfügung zu haben. In diesem Zusammenhang muss ebenfalls noch einmal erwähnt werden, dass der von Alice gesendete Initialschlüssel ohnehin im Durchschnitt die Hälfte seiner Größe verliert, sobald der gemeinsame Schlüssel errechnet wird. Es ergeben sich also folgende Größen:

K_i : Initialschluessel
K_c : erster gemeinsamer Schluessel
K_p : ausgetauschter Teilschluessel
K_f : finaler gemeinsamer Schluessel
wobei gilt
$K_c = 1/2 \cdot K_i$ und
$K_c - K_p = K_f$

Die optimale Länge des Initialschlüssels ist also von zwei anderen Längen abhängig: der Länge des gewünschten finalen gemeinsamen Schlüssels und der des auszutauschenden Teilschlüssels für die Fehlererkennung. Welche Werte für diese gesetzt werden, ist von der Nutzung des Schlüssels abhängig.
Wird als ideale Länge für K_f 128 Bit angenommen (was einen realistischen Wert in Blick auf symmetrische Verschlüsselungsverfahren darstellt) und aus Sicherheitsgründen die selbe Größe für K_p, da die extrem geringe Erfolgswahrscheinlichkeit eines Angreifers mit dieser Länge bereits festgestellt wurde, so ergibt sich die optimale Länge des Initialschlüssels $[K_i]$ wie folgt:

$$\begin{aligned} [K_i] &= ([K_f] + [K_p]) \cdot 2 \\ [K_i] &= (128 + 128) \cdot 2 = 512 \end{aligned} \tag{5.3}$$

Eine Initialschlüssellänge von 512 Bit wäre für die genannten Werte also empfehlenswert. Zu diskutieren wäre nun noch, welche Teilschlüssellänge für die Sicherheit ausreicht. Bei 16 Bit und einer resultierenden Erfolgswahrscheinlichkeit von etwa 0,01% für den Angreifer genügt ein Initialschlüssel von 288 Bit.

5.1.2 Praktische Anwendung

Die theoretischen Annahmen, die zur Sicherheit des Protokolls führen, gehen alle auf die Tatsache zurück, dass ein Fehler im Schlüssel sofort als Angriff zu deuten ist. In der Praxis ist die Übertragung von Photonen jedoch nicht fehlerfrei: es können Polarisationszustände durch die Übertragung in Glasfaserkabeln verloren gehen, ebenso bei der Übertragung durch die Luft. Außerdem müssten die Photonenemitter und Detektoren absolut fehlerfrei arbeiten, was in der Praxis nicht der Fall ist. Einzig die Methode, einen Schwellwert zu bestimmten, welcher Übertragungsfehler in Störungen und Angriff unterteilt, erscheint als Kontrolle sinnvoll. Dieser Wert ist nicht pauschal anzugeben. Viel mehr hängt er von der Qualität und Fehlerfreiheit der Übertragung ab und muss von Fall zu Fall neu bestimmt werden. In der Regel wird er jedoch in der Nähe der 25% angesetzt.

5.2 Weitere Angriffsmöglichkeiten

Neben dem normalen Lauschangriff, welcher in dieser Ausarbeitung beschrieben wurde, existieren einige weitere Angriffsmöglichkeiten, die versuchen, die Sicherheitsprüfung in irgendeiner Form zu umgehen. Beispielsweise wird die Eigenschaft von heutigen Photonendetektoren ausgenutzt, dass ihre Effizienz stark beeinträchtigt ist, wenn ein Signal zu einem unerwarteten Zeitpunkt eintrifft. Es können Zeitfenster entstehen, in denen der Detektor für beispielsweise das 0-Bit aktiv ist, der Detektor für das 1-Bit jedoch komplett inaktiv und ebenso andersherum. Diese Schwäche wurde zuerst 2005 von V. Makarov, A. Anisimov und J. Skaar aufgezeigt [MAS06]. In ihrem Paper wird die Möglichkeit beschrieben, wie Eve eine Messung am öffentlichen Kanal vornehmen und das veränderte Signal zeitversetzt so weiterleiten kann, dass der Fehler nicht auffällt. auch Y. Zhao, C.-H. F. Fung, B. Qi, C. Chen und H.-K. Lo nutzten 2008 diese Schwäche in ihrem "Time-shift attack" aus [ZFQ+08]. Sie zeigten, dass es als Angreifer gar nicht nötig sein muss, das gesendete Signal zu messen. Stattdessen kann Eve durch Zeitversetzung das

Signal einfach in ein von ihr gewünschtes umformen.
Ein anderer Angriff wurde von L. Lydersen, et al. 2010 dargelegt: Durch bestimmt angepasste helle Lichtsignale können Detektoren in zumindest zwei kommerziell verwendeten Quantenkryptographie-Systemen komplett ferngesteuert werden [LWW+10]. Dies macht es möglich, den gesamten Schlüssel zu erhalten, ohne ein Spur zu hinterlassen. Man sieht also, dass die theoretisch sichere Idee heute durch die technische Umsetzung viele Angriffsmöglichkeiten bietet.

5.3 Stärken und Schwächen der Computersimulation

Da es sich beim Gegenstand dieser Arbeit um eine Computersimulation handelt, sind viele Eigenschaften des "echten" BB84-Protokolls nicht gegeben. Hier sollen die wichtigsten Unterschiede und ihre Konsequenzen aufgezeigt werden.

5.3.1 Stärken

Abstraktion

Der größte Vorteil der Visualisierung ist die Anschaulichkeit, die durch einfache benutzte Symbole und simple Animationen gestärkt wird. Da die Kernidee der Arbeit die Verwendung als Lehrmittel ist, wurden komplexe Vorgänge wie die Veränderung quantenmechanischer Zustände hier stark vereinfacht. Für den Betrachter soll das Prinzip hinter dem Schlüsselaustausch verständlich werden und die Grundlage für detaillierte Erklärungen bereitstellen. In Zusammenhang mit einem Lehrbuch oder einer lehrenden Person entfaltet die implementierte Arbeit dann ihr volles Potential.

Individualisierung

Um einen Sachverhalt besser zu verstehen, ist es immer hilfreich, wenn der Betrachter eigene Veränderungen am System vornehmen und das Ergebnis anschließend selbst interpretieren kann. In der Simulation zeigt sich dies zum einen durch

die Einstellungsmöglichkeiten der einzelnen Komponenten, die direkt vom Benutzer verändert werden können, zum anderen aber auch durch die verschiedenen Möglichkeiten der Zusammenstellung aller Plugins. Ob ein oder gar mehrere Mittelsmänner dazugeschaltet werden oder im Anschluss eine Fehlersuche durchgeführt wird, obliegt alleine dem Benutzer. Das Experimentieren trägt gerade bei Autodidakten viel zum Lernerfolg bei.

5.3.1.1 Erweiterbarkeit

Letztlich ist zu beachten, dass die Simulation keine Verschlüsselung beinhaltet, sondern lediglich den Schlüsselaustausch simuliert. Es ist dem Benutzer freigestellt, ob er den ausgetauschten Schlüssel im Anschluss eventuell durch weitere Komponenten, die nicht Teil dieser Arbeit sind, weiterverwenden möchte. Möglichkeiten hierzu wären beispielsweise die Kodierung der Bits in einen Zeichensatz oder die Verwendung einer symmetrischen Chiffre auf beiden Seiten. Das BB84-Protokoll kann also Teil eines übergeordneten Projekts werden.

5.3.2 Schwächen

Der Zufall

Wie bereits in Kapitel 2 durch das Zitat von Zeilinger beschrieben, ist der Zufall, der auftritt, wenn ein polarisiertes Photon auf ein um 45 Grad zu ihm gedrehten Filter trifft, eine absolute Sondererscheinung zu allen bisher gekannten Zufällen. In der Computersimulation wird eine möglichst unberechenbare Zufallszahl erzeugt, die im besten Falle von niemandem vorhergesagt werden kann, da es bei weitem zu viele Einflüsse gibt, die auf diesen Zufall einwirken. Vergleichbar ist dies mit einem Würfelwurf, bei dem man zur Vorhersage des Ergebnisses alle Variablen, die auf diesen Würfel einwirken, kennen müsste, was aus Komplexitätsgründen schlichtweg nicht möglich ist. Der Quantenzufall ist jedoch nicht nur zu komplex für uns, er ist bis zum Zeitpunkt des Ergebnisses nicht definiert! Es ist der perfekte Zufall und dieser wird niemals simulierbar sein.
Auch wenn dies wie ein großer Makel erscheint, ist jedoch zuzugeben, dass es für das Ergebnis des Schlüsselaustausches keine Rolle spielt, da weder dessen Sicherheit noch die Fehlerrate auf diesem Zufall aufbaut.

Der Photonenkanal

Dass in jeder technischen Umsetzung der Kanal zum senden der Photonen Mängel aufweisen wird, wurde bereits erwähnt. In der Simulation lässt sich dieses Phänomen durch eine einstellbare Fehlerrate ausdrücken. Fakt ist jedoch, dass bei bekannter Fehlerrate die Fehler durch einen Angriff sehr leicht von den normal auftretenden zu unterscheiden sind. Zur verbesserten Simulation könnte man die Fehlerrate schwanken lassen, wobei hier ebenfalls die Grenzen einstellbar sein müssten, was im Endeffekt zum gleichen Resultat führt: wenn alle Parameter selbst bestimmt werden, können keine empirischen Schlüsse am Experiment gezogen werden. Da es aber in der Simulation hauptsächlich um den didaktischen Nutzen geht, fällt diese Schwäche nicht allzu sehr ins Gewicht.

Die Detektoren

Da, wie weiter oben beschrieben, viele Angriffe auf den Eigenschaften der Lichtdetektoren aufbauen, sind diese allesamt zumindest in der hier implementierten Simulation nicht durchführbar. Dies ist bei weitem die größte Schwäche, da sie eine Vielzahl von in der Realität existierenden Möglichkeiten zur Manipulation des Schlüsselaustausches ausschließt.

Sichtbare Polarisation

Für die Sicherheit des Protokolls ist der physikalische Fakt essentiell, dass kein Photon in seiner Ausrichtung gesehen werden kann, ohne es durch einen Detektor zu messen und damit unter Umständen zu beeinflussen. Für den Beobachter der implementierten Animation erscheint es jedoch so, als wäre die Polarisation bereits im Photonenkanal deutlich sichtbar. Vor der Vorführung muss unbedingt auf diese Tatsache hingewiesen werden, da sich sonst unweigerlich die Frage stellt, warum Eve denn nicht mit der passenden Basis misst, sondern eine zufällige auswählt. Die Sichtbarkeit für den Betrachter der Simulation ist jedoch keine Schwäche, sondern geschieht mit voller Absicht: nur so wird deutlich, wie die Fehler entstehen, die anschließend zur Aufdeckung von Angriffen führen. Würde man die sichtbare

Polarisation ausblenden, so könnte der Betrachter genau so wenig wie Eve oder Bob feststellen, welche Photonen verändert wurden und wo Fehler aufgetreten sind.

Zeitliche Abfolge

Für die Sicherheit des Protokolls ist es wichtig, dass der Austausch der Basen zwischen Alice und Bob erst nach der Übertragung der polarisierten Photonen geschieht, da Eve sonst alle Informationen hätte, um unbemerkt zu lauschen. In der Simulation geschehen beide Vorgänge jedoch parallel, was auf die technischen Gegebenheiten von CT2 zurückzuführen ist: ein zeitlicher Abstand beim Ausführen von Plugins lässt sich dort nicht realisieren. Wenn auf diese Schwäche in der Simulation hingewiesen wird, spielt sie jedoch für die Anschaulichkeit nur eine untergeordnete Rolle.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung der schriftlichen Ausarbeitung

In dieser Ausarbeitung wurden zunächst alle Grundlagen bereitgestellt, die zum Verständnis der Arbeit erforderlich sind. Außerdem wurde die Entwicklungsumgebung vorgestellt, sowie die angestrebten Ziele verdeutlicht.
Im Anschluss wurde das zu implementierende Protokoll zunächst im Allgemeinen erklärt, bevor ein detaillierter Entwurf der Komponenten, ihrer Funktion und ihres Zusammenspiels untereinander aufgeführt wurde. Schließlich wurde die konkrete Implementierung in großer Ausführlichkeit dargelegt, wobei sowohl auf die einzelnen implementierten Plugins als auch auf deren Zusammenspiel innerhalb zweier erstellter Vorlagen eingegangen wurde. Die Beschreibungen wurden durch Code-Ausschnitte und Grafiken ergänzt, um verschiedene Sachverhalte besser erklären zu können. Zuletzt wurden sowohl das theoretische BB84-Protokoll und die reale technische Umsetzung auf ihre Stärken und Schwächen untersucht, indem Sicherheitsaspekte und Angriffsmöglichkeiten aufgeführt wurden. Die Vor- und Nachteile der hier implementierten Computersimulation schloss dieses Kapitel ab.

6.2 Umsetzung der aufgestellten Ziele

In der Einleitung dieser Arbeit wurden fünf Anforderungen gestellt, dessen Umsetzung hier kurz zusammengefasst werden soll.

R1: Der Schlüsselaustausch ist in vollem Umfang zu simulieren

Durch vier der sechs erstellten Plugins für CT2 wird der eigentliche Schlüsselaustausch simuliert. Diese sind der Photonenbasis-Generator, der Photonenkodierer, der Photonendekodierer sowie der Schlüsselgenerator. Um die Erfüllung dieser Anforderung alleine zu betrachten kann die erstellte Vorlage "BB84-Schlüsselaustausch" geöffnet werden. Die Einstellungsmöglichkeiten der Plugins vervollständigen die Lösung.

R2: Ein simulierter Lauschangriff sowie Übertragungsfehler können optional aktiviert werden

Zur Erfüllung von R2 wurden zum einen das Plugin des Mittelsmanns implementiert, zum anderen lassen sich die Übertragungsfehler in den Einstellungen des Photonendekodierers aktivieren.

R3: Eine Sicherheitsprüfung des übertragenen Schlüssels muss durchführbar sein

Das letzte implementierte Plugin, der Fehlerdetektor, dient zur Erfüllung von R3, wobei der zu prüfende Teil des ausgetauschten Schlüssels vom Benutzer selbst ausgewählt werden kann. Ob die Sicherheit des Schlüssels gegeben ist, wird dem Benutzer durch diese Komponente mitgeteilt.

R4: Visuelle Präsentationen zum besseren Verständnis der Funktionsweise können abgespielt

Der Weg der Photonen vom Kodierer über den Mittelsmann zum Dekodierer wurde durch Animationen in jedem dieser Plugins sichtbar gemacht. Diese können je nach Bedarf ein- und ausgeschaltet werden. Auch der Schlüsselgenerator verfügt über eine Präsentation, welche die Berechnung des gemeinsamen Schlüssels verdeutlicht. Alle Animationsgeschwindigkeiten können vom Benutzer angepasst werden.

R5: Die Beschriftung und Dokumentation der einzelnen Komponenten erfolgt bilingual in Deutsch und Englisch

Alle Titel, Ein- und Ausgänge und Einstellungsmöglichkeiten wurden in beiden Sprachen beschriftet. Außerdem wurde für jede implementierte Komponente eine Hilfedatei angelegt, welche jeweils auf Deutsch und Englisch eine kurze Einführung, die Gebrauchsanweisung und, wenn vorhanden, eine Beschreibung der Präsentation enthält.

Somit wurden die Anforderungen an diese Arbeit allesamt erfüllt.

6.3 Fazit

In dieser Bachelorarbeit wurde das BB84-Protokoll in CrypTool 2.0 integriert. Durch die Anschaulichkeit der Visualisierung hilft es Studenten, Schülern und anderen Interessierten, einen Einstieg in dieses Thema zu erhalten. Die Arbeit hat einen Grundstein dafür gelegt, an diesem oder einem ähnlichen Thema weiter zu arbeiten und somit CrypTool 2.0 auch bei Entwicklung neuer quantenmechanischen Verfahren aktuell zu halten. Es konnte außerdem veranschaulicht werden, dass die Schwächen der Computersimulation für Lehrzwecke nicht ins Gewicht fallen, da dessen Funktionalität trotzdem in vollem Umfang realisiert wurde.

6.4 Ausblick

Im letzten Abschnitt dieser Ausarbeitung sollen zum Einen Anstöße für die Zukunft gegeben werden, wie die hier geleistete Arbeit sinnvoll ergänzt werden kann. Außerdem werden einige weitere Phänomene der Quantenmechanik, welche die Kryptographie betreffen, aufgeführt.

6.4.1 Ergänzungsmöglichkeiten in zukünftigen Arbeiten

Der Einschub in Kapitel 3 beschreibt bereits das Problem eines defekten und damit unbrauchbaren Schlüssels bei fehlerhafter Übertragung. Der entworfene Fehlerde-

tektor kann diese Fehler zwar berechnen und damit einen Angreifer erkennen, ist jedoch nicht zur Korrektur fähig. Aus diesem Grund wäre die Implementierung einer weiteren Komponente in CrypTool 2.0 sinnvoll, welche ein fehlerkorrigierendes Protokoll ausführt, das zusammen mit dem BB84-Protokoll eine gemeinsame Vorlage bilden könnte. Dieses Plugin müssten beide Parteien erhalten und es sollte eine Verbindung zwischen ihnen existieren, die zum Austausch der Schlüsselblöcke in beide Richtungen benutzt wird. Der Algorithmus müsste dann entweder bidirektional implementiert werden, sodass er auf beiden Seiten gleich funktioniert, oder es müsste eine entsprechende Einstellungsmöglichkeit für jeweils den Sender und den Empfänger vorhanden sein. Außerdem könnte der Mittelsmann erweitert werden, sodass er auch beim Austausch dieser Daten versucht, Informationen abzufangen.
Eine weitere, aber komplexere, Möglichkeit der Erweiterung wäre eine tatsächliche Implementierung einer Schnittstelle zur benötigten Hardware für einen echten quantenmechanischen Schlüsselaustausch. Ein Laser sowie ein Photonendetektor könnten mit der entsprechenden Software direkt über CT2 gesteuert werden, um das Protokoll nicht nur zu simulieren, sondern tatsächlich durchzuführen. Die in der Simulation verwendeten Algorithmen ließen sich dabei mit wenig Mühe anpassen, einzig die teure Hardware und eventuell umfangreiche Treiberintegration wäre eine größere Herausforderung.
Letztlich sei noch die Möglichkeit aufgegriffen, die implementierten Vorlagen durch Verschlüsselungsverfahren zu erweitern oder in andere Projekte zu integrieren, um umfangreichere Simulationen zu ermöglichen. Die Rolle des Mittelsmann oder der auftretenden Übertragungsfehler wird anhand andere Komponenten eventuell noch deutlicher.

6.4.2 Weitere Quantenmechanik in der Kryptographie

Da es sich beim implementierten Protokoll um ein quantenmechanisches Verfahren handelt, sollen noch einige Beschreibungen anderer Bereiche der Kryptographie, in denen ebenfalls die Gesetze der Quantenphysik benutzt werden, gegeben werden.

6.4.2.1 Das Ekert-Protokoll

Ein ganz ähnliches Verfahren wie das des BB84-Protokolls stellt das Ekert-Protokoll zum Quantenschlüsselaustausch dar [Eke91]. Kern dieses Verfahren ist der physi-

kalische Zustand der Verschränkung von Teilchen auf subatomaren Ebenen, wie beispielsweise Photonen. In diesem Zustand sind zwei Teilchen auf nichtlokale Art miteinander verbunden. Vor einer Messung sind ihre Eigenschaften, bei Photonen zum Beispiel die Polarisation, nicht festgelegt. Wird jedoch ein Teilchen gemessen und damit definiert, ist im gleichen Moment die Ausrichtung des anderen Teilchens ebenfalls festgelegt. Dieser für die klassische Physik schwer vorstellbare Prozess, der von Einstein aufgrund seiner Unheimlichkeit als "spukhafte Fernwirkung" [Bor69] bezeichnet wurde, ist für den Schlüsselaustausch von großem Wert. Nach der Verschränkung kann das eine Teilchen nämlich zu Alice, das andere zu Bob geschickt und anschließend auf beiden Seiten gemessen werden. Da durch die eigene Messung auch das Messergebnis des jeweils anderen eindeutig wird, können beide einen gemeinsamen Schlüssel erzeugen, ohne weiter miteinander zu kommunizieren. Auch eine Sicherheitsprüfung kann durchgeführt werden, da bestimmte physikalische Gegebenheiten wie die *Bellsche Ungleichung* [B+64] nur erfüllt sind, wenn beide Teilchen beim Messen nicht verschränkt wurden, was auf einen Mittelsmann deuten würde, welche die Verschränkung beim Messen aufgehoben hat. Dieses Verfahren wurde bereits 1999 auf kommerzieller Ebene von Anton Zeilinger in Zusammenarbeit mit dem *Austrian Institute of Technology* demonstriert [JSW+00].
Eine Simulation dieses Protokoll ließe sich ebenfalls in CrypTool 2.0 implementieren.

6.4.2.2 Quantenmechanischer Zufall

Der bereits mehrfach genannte besondere und absolute Zufall bei quantenmechanischen Phänomenen könnte in vielen sicherheitstechnischen Verfahren von großem Wert sein. Bei der Erzeugung von Schlüsseln beispielsweise für das One-Time-Pad hängt die gesamte Sicherheit davon ab, dass der Schlüssel unvorhersehbar zufällig generiert wird. Neben den bislang erfolgreichsten Verfahren zur Erzeugung einer solchen Zufälligkeit wie das Messen der Strahlung einer radioaktiven Quelle oder das Abhören von Atmosphärenrauschen bieten sich verhältnismäßig einfache quantenmechanische Verfahren an. Einzelne polarisierte Photonen könnten auf einen Spalt mit einer zu 45 Grad gedrehten Ausrichtung geschickt werden. Wie schon in Kapitel 2 beschrieben wird die Wahrscheinlichkeit, durch den Spalt zu gelangen für jedes Photon bei 50% liegen, und diese Wahrscheinlichkeit ist völlig unabhängig von anderen Variablen. Somit ließen sich durch einfache Kodie-

rungsverfahren (Photon gelangt durch den Spalt: '1', Photon wird absorbiert: '0') beliebig lange binäre Zufallszahlen erzeugen.

6.4.2.3 Post-Quanten-Kryptographie

In der Einleitung dieser Arbeit wurde erwähnt, dass es zukünftigen Computermodellen auf Quantenbasis (sogenannten *Quantencomputern*) eventuell möglich sein könnte, aktuelle kryptographische Verfahren zu brechen. Aus diesem Grund beschäftigt sich das Teilgebiet der *Post-Quanten-Kryptographie* (Begriff eingeführt von Daniel J. Bernstein [BGD+04]) mit Verschlüsselungs- und Authentifizierungsmethoden, die selbst bei Verwendung von leistungsstarken Quantencomputern nicht unerlaubt zu entschlüsseln sind. Hauptaugenmerk dieser Fachrichtung sind asymmetrische Verschlüsselungen, da diese oft auf der Problematik der Primfaktorzerlegung oder der Berechnung des diskreten Logarithmus beruhen, was durch leistungsstarke Quantencomputer zumindest in der Theorie möglich ist. Symmetrische Verfahren hingegen sind meist durch Vergrößerung des verwendeten Schlüssels einfach zu stärken und deswegen nicht die primären Ziele dieses Bereichs.
Der in dieser Arbeit implementierte BB84-Schlüsselaustausch gehört ebenfalls zu den Protokollen, die in der Post-Quanten-Kryptographie als sichere Schlüsselaustauschmethode genutzt werden.

Literaturverzeichnis

[B+64] Bell, John S et al.: *On the einstein-podolsky-rosen paradox*. Physics, 1(3):195–200, 1964.

[BB+84] Bennett, Charles H, Gilles Brassard et al.: *Quantum cryptography: Public key distribution and coin tossing.* In: *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*, Band 175. New York, 1984.

[BGD+04] Buchmann, Johannes, Luis Carlos Coronado García, Martin Döring, Daniela Engelbert, Christoph Ludwig, Raphael Overbeck, Arthur Schmidt, Ulrich Vollmer und Ralf-Philipp Weinmann: *Post-Quantum Signatures.* IACR Cryptology ePrint Archive, 2004:297, 2004.

[Bor69] Born, Max: *Albert Einstein, Hedwig und Max Born. Briefwechsel 1916-1955*. München, 1969.

[BS94] Brassard, Gilles und Louis Salvail: *Secret-key reconciliation by public discussion.* In: *advances in Cryptology—EUROCRYPT'93*, Seiten 410–423. Springer, 1994.

[ct2a] *Cryptool 2.0*. `http://www.cryptool.org/de/cryptool2`.

[ct2b] *CrypTool 2.0 Tutorial*. `http://www.cryptool.org/de/ct2-dokumentation-de`.

[DR00] Daemen, Joan und Vincent Rijmen: *The block cipher Rijndael.* In: *Smart Card Research and Applications*, Seiten 277–284. Springer, 2000.

[Eke91] Ekert, Artur K: *Quantum cryptography based on Bell's theorem.* Physical review letters, 67(6):661–663, 1991.

[HDM80] Hellman, Martin E, Bailey W Diffie und Ralph C Merkle: *Cryptographic apparatus and method*, April 29 1980. US Patent 4,200,770.

[Hub11] Huber, Thomas Claudius: *Windows Presentation Foundation*. Galileo Press, 2011.

[JSW+00] Jennewein, Thomas, Christoph Simon, Gregor Weihs, Harald Weinfurter und Anton Zeilinger: *Quantum cryptography with entangled photons*. Physical Review Letters, 84(20):4729, 2000.

[KB06] Köbler, Johannes und Olaf Beyersdorff: *Von der Turingmaschine zum Quantencomputer—ein Gang durch die Geschichte der Komplexitätstheorie*. In: *Informatik*, Seiten 165–195. Springer, 2006.

[Lib09] Liberty, Jesse: *Programming C#: Building. NET Applications with C#*. O'reilly, 2009.

[Lis] Lischke, Andrea: *Vernam-Chiffre und das One-Time-Pad*.

[LM04] Lenz, Gunther und Thomas Moeller: *Dot NET: A Complete Development Cycle*. Addison-Wesley Professional, 2004.

[LWW+10] Lydersen, Lars, Carlos Wiechers, Christoffer Wittmann, Dominique Elser, Johannes Skaar und Vadim Makarov: *Hacking commercial quantum cryptography systems by tailored bright illumination*. Nature photonics, 4(10):686–689, 2010.

[MAS06] Makarov, Vadim, Andrey Anisimov und Johannes Skaar: *Effects of detector efficiency mismatch on security of quantum cryptosystems*. Physical Review A, 74(2):022313, 2006.

[RCH13] Rennert, P., A. Chassé und W. Hergert: *Einführung in die Quantenphysik: Experimentelle und theoretische Grundlagen mit Aufgaben, Lösungen und Mathematica-Notebooks*. Gabler, Betriebswirt.-Vlg, 2013.

[Riv95] Rivest, Ronald L: *The RC5 encryption algorithm*. In: *Fast Software Encryption*, Seiten 86–96. Springer, 1995.

[RSA78] Rivest, Ronald L, Adi Shamir und Len Adleman: *A method for obtaining digital signatures and public-key cryptosystems.* Communications of the ACM, 21(2):120–126, 1978.

[Sho94] Shor, Peter W: *Algorithms for quantum computation: discrete logarithms and factoring.* In: *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, Seiten 124–134. IEEE, 1994.

[Weg05] Wegener, Ingo: *Complexity theory - exploring the limits of efficient algorithms.* Springer, 2005.

[WZ82] Wootters, William K und Wojciech H Zurek: *A single quantum cannot be cloned.* Nature, 299(5886):802–803, 1982.

[xam] *xaml.* `http://msdn.microsoft.com/en-us/library/ms752059.aspx`.

[xml] *xml.* `http://www.w3.org/standards/xml`.

[Zei03] Zeilinger, Anton: *Einsteins Schleier: die neue Welt der Quantenphysik.* CH Beck, 2003.

[ZFQ+08] Zhao, Yi, Chi-Hang Fred Fung, Bing Qi, Christine Chen und Hoi-Kwong Lo: *Quantum hacking: Experimental demonstration of time-shift attack against practical quantum-key-distribution systems.* Physical Review A, 78(4):042333, 2008.

Eidesstattliche Erklärung

Ich, Benedict Beuscher, Matrikelnummer 2245950, wohnhaft in 47057 Duisburg, versichere an Eides Statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen übernommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe.

Ich versichere an Eides Statt, dass ich die vorgenannten Angaben nach bestem Wissen und Gewissen gemacht habe und dass die Angaben der Wahrheit entsprechen und ich nichts verschwiegen habe.

Die Strafbarkeit einer falschen eidesstattlichen Versicherung ist mir bekannt, namentlich die Strafandrohung gemäß §156 StGB bis zu drei Jahren Freiheitsstrafe oder Geldstrafe bei vorsätzlicher Begehung der Tat bzw. gemäß 163 StGB bis zu einem Jahr Freiheitsstrafe oder Geldstrafe bei fahrlässiger Begehung.

Duisburg, 13. Oktober 2013 _____

Unterschrift