Bestimmung sicherer Schlüsselraten in der Quantenkryptographie

Master-Arbeit
zur Erlangung des Akademischen Grades
Master of Science
(M.Sc.)

der Universität Siegen

Department Physik

vorgelegt von
Florian Köppen

29. März 2017

Inhaltsverzeichnis

Abbildungsverzeichnis

Symbole

$H()$	Shannon-Entropie
$S()$	von-Neumann-Entropie
$I()$	klassische Information
$\chi()$	Holevo-Größe (Quanteninformation)
$r()$	Schlüsselrate
ρ	Dichtematrix

Kapitel 1

Einleitung

1.1 Einleitung

Kryptografie oder die Verschlüsselung von Informationen ist in den letzten Jahrzehnten zu einem fundamentalen Bestandteil unserer modernen Gesellschaft geworden. Egal ob zu Hause im W-LAN, im Supermarkt an der Kasse beim Bezahlen mit der EC-Karte oder beim Online-Banking, überall werden verschlüsselt Daten übertragen. Dabei basieren all die verwendeten Verfahren zur Verschlüsselung der Daten auf einem von zwei Grundprinzipien. Das erste Prinzip sind die symmetrischen Kryptosysteme. Hier besitzen sowohl der Sender (Alice), als auch der Empfänger der Daten (Bob) einen gemeinsamen Schlüssel mit dem die Daten ver- bzw. entschlüsselt werden. Der Schlüssel ist im allgemeinen so lang wie die Nachricht, die gesendet werden soll. Dann ist die verschlüsselte Nachricht die Summe von Klartext und Schlüssel. Symmetrische Kryptosysteme sind nur so lange sicher, wie der Schlüssel nur für eine einzige Nachricht genutzt wird. Diese Methode der Verschlüsselung ist besonders schnell, allerdings

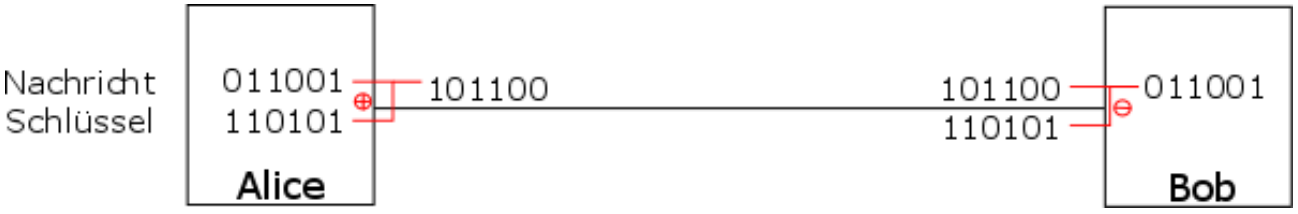

Abbildung 1.1: Symmetrische Kryptosysteme

muss der Schlüssel über einen sicheren Kanal übertragen werden. 1975 wurde der Data Encryption Standard (DES) als Behördlicher Verschlüsselungsstandard in den USA publiziert, der sich vor allem für den Hardwarebereich eignet. Es gibt bei diesem, wie auch bei anderen Standard

Verschlüsselungsalgorithmen nur praktische keine unbedingte Sicherheit.
Dies gilt auch für das zweite Prinzip von Kryptosystemen, den asymmetrischen Kryptosystemen. Grundlage dieser Kryptosysteme ist der Diffie-Hellman Schlüsselaustausch. Er ermöglicht es, den Schlüssel über einen öffentlichen Kanal zu verteilen. Dazu erzeugt Bob zunächst einen privaten Schlüssel, den er für sich behält. Aus diesem Privaten Schlüssel wird ein öffentlicher Schlüssel erzeugt und an Alice gesendet. Alice nutzt diesen Schlüssel um die Nachricht zu verschlüsseln. Dann wird die verschlüsselte Nachricht an Bob gesendet und dort mit dem privaten Schlüssel entschlüsselt. Auch die Sicherheit dieser Kryptosysteme beruht auf der Komplexität

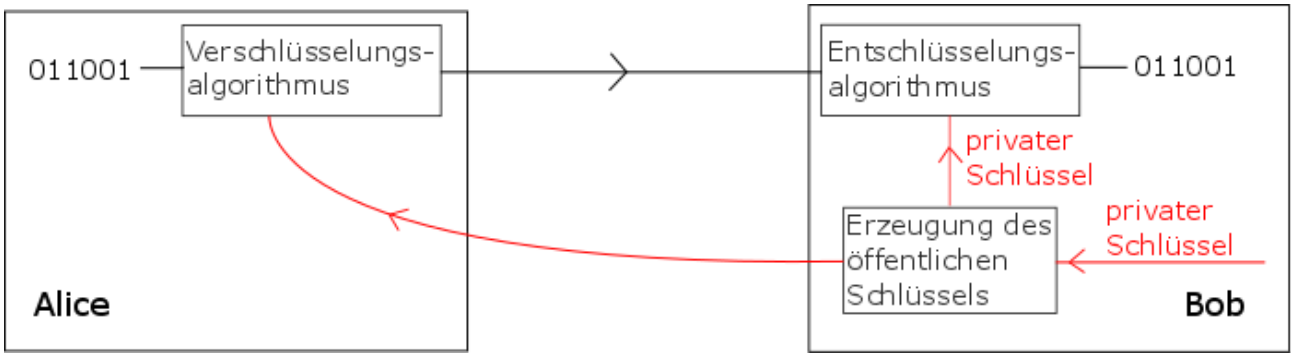

Abbildung 1.2: Asymmetrische Kryptosysteme

der Bestimmung des Schlüssels durch einen Abhörer.
Insgesamt ist zu bemerken, dass die Sicherheit aller Systeme mit öffentlichem Schlüssel nicht bewiesen ist. Viele beruhen z.B. auf der L̈angsamkeitd̈er Primfaktorzerlegung, die mathematisch allerdings unbewiesen ist.

1.2 Quantum Key Distribution und das BB84

Anfang der 1970er Jahre hatte Stephen Wiesner die Idee, die Quantenmechanik zur Kryptographie zu nutzen. Die Idee war ganz einfach. Ausgehend von der Annahme das jede Messung ein Quantensystem stört, kam man auf die Idee statt klassischer Bits, Qubits als Schlüssel über einen Quantenkanal zu versenden. Das bedeutet ein Lauscher (Eve) kann die Qubits nicht messen ohne diese zu stören und damit seine Präsens zu enthüllen. Mit dieser Idee im Kopf entwickelten Charles Bennett und Gilles Brassard 1984 das erste Protokoll zur Quantenschlüsselverteilung (BB84).
Das BB84 zeigt die Idee, die allen QKD-Protokollen zu Grunde liegt. Um einen Schlüssel zu erzeugen präpariert Alice statt klassischer Bits, zufällige Qubits in Signalzuständen. Dabei ist es wichtig, dass die Signalzustände nicht alle orthogonal zueinander sind. Im BB84 werden dazu

jeweils zwei Zustände aus zwei verschiedenen Basen gewählt, die den klassischen Bits 0 und 1 entsprechen. Betrachtet man Photonen mit ihren Polarisationszuständen, so präpariert Alice die Qubits in den Signalzuständen $|+\rangle$, $|-\rangle$, $|0\rangle$ und $|1\rangle$. Dabei entsprechen $|+\rangle$ und $|0\rangle$ dem klassischen Bit 0 und $|-\rangle$ und $|1\rangle$ dem klassischen Bit 1. Alice schickt diese über einen (hoffentlich) sicheren Kanal an Bob. Dieser führt nun an den Signalzuständen eine Messung durch. Dabei entscheidet er bei jedem Qubit, ob er in der x- oder in der z-Basis misst. Die Messbasis wird über einen öffentlichen Kanal an Alice übermittelt. Diese akzeptiert die Messung oder lehnt sie ab. Die abgelehnten Bits werden aus dem Schlüssel gestrichen. Bei einem rauschfreien Kanal muss der übrig gebliebene Schlüssel nun übereinstimmen oder Eve hat den Schlüssel mitgehört. Um das zu prüfen tauschen Alice und Bob einen Teil des Schlüssels aus.
Ist der Kanal nicht rauschfrei werden Fehlerkorrekturprotokolle und Privacy-Amplification Protokolle aus der klassischen Kryptographie angewendet um die Fehler zu entfernen und die Sicherheit zu maximieren.

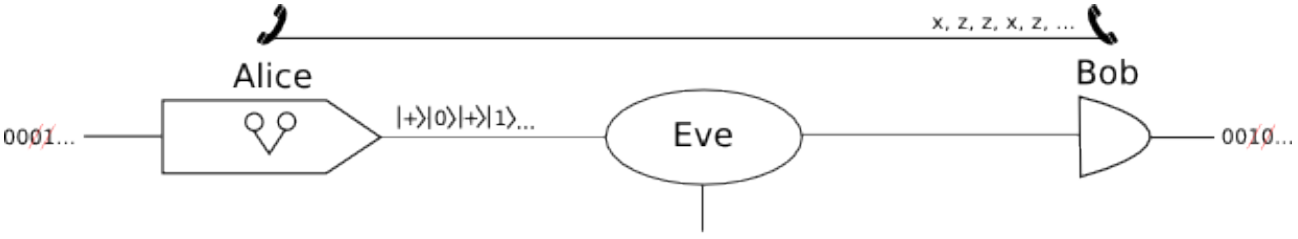

Abbildung 1.3: Das BB84

Die Quantenschlüsselverteilung hat eine große Menge verschiedener Protokolle hervorgebracht, die alle mehr oder weniger auf den Ideen des BB84 basieren. Dabei ist es bei diesen Protokollen von Interesse, wie groß der Fehler ist, den sie zulassen, bis sie unsicher werden und wie effizient ein sicherer Schlüssel mit diesem Protokoll erzeugt werden kann. Dazu gibt es bei vielen Protokollen, wie zum Beispiel beim 2-state Protokoll von 1992 (B92) analytische Abschätzungen der Sicherheitsschwelle. In dieser Arbeit soll nun versucht werden, durch die konvexe Optimierung der Schlüsselrate zu zeigen, wie genau diese Abschätzungen sind und ob diese noch verbessert werden kann.

Kapitel 2

Herleitung der Schlüsselrate

In der Quanten Schlüsselverteilung ist die Schlüsselrate der Dreh- und Angelpunkt. Dabei stehen zwei zentrale Fragen im Mittelpunkt. *Ist das verwendete Protokoll sicher?* Und die für diese Arbeit zentrale Frage, *wie hoch ist die generierbare sichere Schlüsselrate?* Diese Frage lässt sich mit Hilfe des Sicherheitsbeweises aus den Referenzen [DW05], [RGK05] und [FL12] beantworten.

2.1 Schema zur Ersetzung der Quelle

Die Standard-Protokoll in der Quanten Schlüsselverteilung sind zumeist Protokolle nach dem Prepare & Measure Prinzip. Dabei präpariert Alice mit einer gewissen Wahrscheinlichkeit $p(x)$ einen Signalzustand $|s_x\rangle_S$ aus einem Set $\mathcal{S}$ verschiedener Signalzustände und sendet diesen an Bob. Um später eine optimierbare Form der Schlüsselrate zu erhalten, ist es allerdings notwendig, ein anderes Schema der Zustandspräparation zu benutzen. Dazu wird das so genannte Source-Replacement Schema aus Ref. [FL12] genutzt.
Alice sendet hierbei nicht mehr direkt Signalzustände an Bob, sondern hat in ihrem Labor eine Quelle, die verschränkte zweiparteien Zustände erzeugt. Von diesem Zustand sendet Alice einen Teil durch den Quantenkanal zu Bob, an dem anderen Teil führt sie selbst projektive Messungen durch. Dadurch ergibt sich ein Eingangszustand

$$|\Phi\rangle_{AS} = \sum_x \sqrt{p(x)}\,|x\rangle_A \otimes |s_x\rangle_S \tag{2.1}$$

mit einem Set $\mathcal{X} = \{|x\rangle \mid x = 0, \ldots, |S| - 1\}$ aus orthonormalen Basis-Zuständen eines Hilbertraumes $\mathcal{H}_\mathcal{S}$ mit der Dimension $|S|$. Man sieht leicht, dass nach der projektiven Messung von

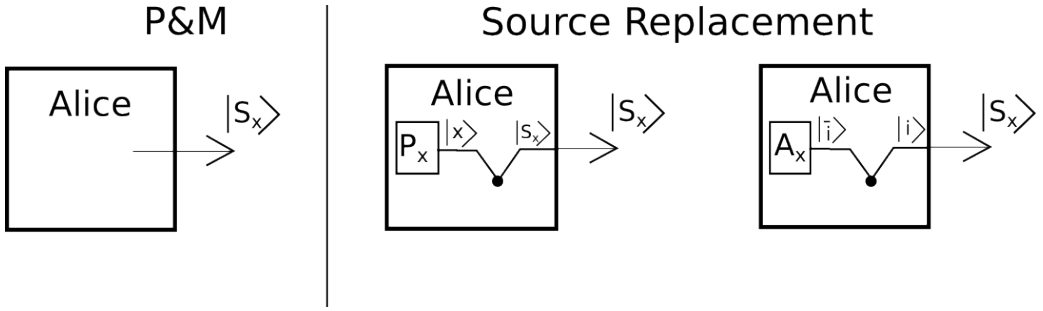

Abbildung 2.1: Der Vergleich zwischen Prepare & Measure Protokoll und Source Replacement Protokoll. Beim Prepare & Measure Protokoll wird der Signalzustand von Alice präpariert und an Bob gesendet. Beim Source Replacement Protokoll wird von einer von Alice kontrollierten Quelle ein verschränkter zweiparteien Zustand präpariert. Dann führt Alice eine projektive bzw. eine allgemeine Messung an ihrem Teil des Zustandes durch.

Alice die selbe Situation, wie beim Prepare & Measure Prinzip vorliegt.
Mit Hilfe der Schmidt-Zerlegung lassen sich die Eingangszustände, falls die Signalzustände $|s_x\rangle_S$ linear abhängig sind, in eine kompaktere Form

$$|\Phi\rangle_{AS} = \sum_{i=0}^{d-1} \sqrt{\alpha_i}\, |\bar{i}\rangle_A \otimes |i\rangle_S \ , \tag{2.2}$$

mit den Schmidtkoeffizienten $\sqrt{\alpha_i}$, bringen. Es gilt $d < |S|$ und die Vektoren $|i\rangle$ bilden nun eine orthonormale Basis. Die orthonormale Basis $|\bar{i}\rangle$ im System Alice lässt sich leicht aus den ursprünglichen Vektoren bestimmen

$$\begin{aligned} & \sum_{j=1}^{d-1} \sqrt{\alpha_j}\, |\bar{j}\rangle_A \otimes |j\rangle_S = && \sum_x \sqrt{p(x)}\, |x\rangle_A \otimes |s_x\rangle_S \\ \Leftrightarrow \quad & \sqrt{\alpha_i}\, |\bar{i}\rangle_A \langle i|i\rangle_S = && \sum_x \sqrt{p(x)}\, |x\rangle_A \langle i|s_x\rangle_S \\ \Leftrightarrow \quad & |\bar{i}\rangle_A = && \sum_x \sqrt{\frac{p(x)}{\alpha_i}}\, \langle i|s_x\rangle_S\, |x\rangle_A \, . \end{aligned} \tag{2.3}$$

Des weiteren gibt es eine Beziehung zwischen den projektiven Messungen von Alice im ursprünglichen Hilbertraum und einem POVM (Positive Operator Valued Probability Measure) $\mathcal{M}_A = \{A_x\}$ im neuen kleineren Hilbertraum. Mit Hilfe der Naimark Erweiterung [Nai40] kann gezeigt werden, dass für jedes POVM $\mathcal{M}$ in einem Hilbertraum $\mathcal{H}_A$ eine entsprechende projektive Messung $\{P_i\}$ in einem höher dimensionalen Hilbertraum $\mathcal{H}_A \otimes \mathcal{H}_B$ existiert.

Satz 2.1 (Naimark Erweiterung)**.** Für jedes POVM $\mathcal{M} = \{M_i\}_{i\in\Omega}$, $\sum_{i\in\Omega} M_i = \mathbb{1}$, $M_i \geq 0$ eines

endlichen Datensets Ω in einem Hilbertraum $\mathcal{H}_A$ existiert ein Hilbertraum $\mathcal{H}_B$, ein Zustand $\rho_B \in \mathcal{H}_B$ und ein Set projektiver Messungen $\mathcal{P} = \{P_i\}_{i\in\Omega}$ in $\mathcal{H}_A \otimes \mathcal{H}_B$, so dass gilt

$$\mathrm{tr}_A\left(\rho M_i\right) = \mathrm{tr}_{A,B}\left(\left[\rho \otimes \rho_B\right] P_i\right), \qquad \rho \in \mathcal{S}(\mathcal{H}_A),\ \forall i \in \Omega. \tag{2.4}$$

Beweis. Nach Davies [Dav78] genügt es, Rang eins Matrizen der Form $M_i = u_i u_i^\dagger$ mit $\mu = 1, \ldots, N$ zu betrachten. Dies sind Elemente eines optimalen POVM. Mit Hilfe der Vektoren v_s, die orthogonal zueinander und zu den Vektoren u_i werden $N - n$ zusätzlichen Dimensionen zu $\mathcal{H}_A$ hinzugefügt. Der Index s läuft dabei von $n+1$ bis N. Betrachtet man die Vektoren

$$w_i := u_i + \sum_{s=n+1}^{N} c_{is} v_s$$

mit den komplexen Koeffizienten c_{is}, so bilden diese eine komplette orthonormale Basis im größeren Hilbertraum $\mathcal{H}_B$, falls gilt

$$\langle w_j, w_i\rangle = \langle u_j, u_i\rangle + \sum_{s=n+1}^{N} c_{js}^* c_{is} = \delta_{ij}.$$

Es gibt mehr Gleichungen als unbekannte Koeffizienten c_{is}. Außerdem folgen die Vektoren u_i der Geschlossenheitsbedingung des POVM $\sum_{i=1}^N M_i = \sum_{i=1}^N u_i u_i^\dagger = \mathbb{1}$ und damit $\sum_{i=1}^N u_{i\mu}^* u_{i\nu} = \delta_{\mu\nu}$, wobei μ und ν von 1 bis n läuft.
Nach dem selben Prinzip lässt sich das Skalarprodukt explizit schreiben, als

$$\sum_{\mu=1}^{n} u_{j\mu}^* u_{i\mu} + \sum_{s=n+1}^{N} c_{js}^* c_{is} = \delta_{ij},\ (i,j = 1, \ldots, N).$$

Betrachtet man nun die quadratische Matrix

$$\begin{pmatrix} u_{11} & \ldots & u_{1n} & c_{1,n+1} & \ldots & c_{1N} \\ u_{21} & \ldots & u_{2n} & c_{2,n+1} & \ldots & c_{2N} \\ \vdots & & \vdots & \vdots & & \vdots \\ u_{N1} & \ldots & u_{Nn} & c_{N,n+1} & \ldots & c_{NN} \end{pmatrix},$$

so ist diese unitär. Die ersten n Spalten sind n orthonormale Vektoren in einem N-dimensionalen Hilbertraum. Es gibt dann unendlich viele Möglichkeiten die fehlenden $N - n$ orthonormalen Vektoren zu konstruieren. □

Die POVM Elemente A_x entsprechen der Projektion der Projektoren $P_x = |x\rangle\langle x|$ in den neuen Hilbertraum

$$\begin{aligned} A_x &= \sum_i |i\rangle\langle i|\, P_x \sum_j |j\rangle\langle j| \\ &= \sum_{i,j} \sqrt{\frac{p(x)}{a_i^*}} \sqrt{\frac{p(x)}{a_j}} \langle s_x|i\rangle \langle j|s_x\rangle |i\rangle\langle j| \\ &= p(x) \sum_{i,j} \sqrt{\frac{1}{a_i^* a_j}} \langle s_x|i\rangle \langle j|s_x\rangle |i\rangle\langle j| \,. \end{aligned} \tag{2.5}$$

Wendet Alice dieses POVM auf den von der Quelle präparierten Zustand an, so erhält Bob die Signalzustände, die sich durch die projektive Messung ergeben würden

$$\begin{aligned} \mathrm{tr}_A \left(A_x \otimes \mathbb{1}\, |\Psi\rangle\langle\Psi|\right) &= \mathrm{tr}_A \left(p(x) \sum_{i,j} \sqrt{\frac{1}{a_i^* a_j}} \langle s_x|i\rangle \langle j|s_x\rangle |i\rangle\langle j| \otimes \mathbb{1} \right. \\ &\qquad \left. \sum_m \sqrt{a_m}\, |m\rangle \otimes |m\rangle \sum_n \sqrt{a_n^*}\, \langle n| \otimes \langle n| \right) \\ &= \mathrm{tr}_A \left(p(x) \sum_{i,j,n} \sqrt{\frac{a_n^*}{a_i^*}} \langle s_x|i\rangle \langle j|s_x\rangle |i\rangle\langle n| \otimes |j\rangle\langle n| \right) \\ &= p(x) \sum_{i,j} \langle s_x|i\rangle \langle j|s_x\rangle |j\rangle\langle i| \\ &= p(x)\, |s_x\rangle\langle s_x| \,. \end{aligned} \tag{2.6}$$

Die projektive Messung $P_x = |x\rangle\langle x|$ im Hilbertraum $\mathcal{H}_x$ entspricht somit dem POVM Element A_x im kleineren Hilbertraum $\mathcal{H}_A$.

Da nun sowohl der zu präparierende Zustand als auch die allgemeine Messung die Alice an ihrem Qubit machen muss, um dem Prepare & Measure Protokoll zu entsprechen, bestimmt sind, kann dieses Schema durch ein passendes Source-Replacement Schema ersetzt werden. Im nächsten Schritt ist zu bestimmen, wie groß die sichere Schlüsselrate auf der Grundlage dieses Schemas ist.

2.2 Die Bestimmung der Schlüsselrate

Mit dem Source-Replacement Schema gibt es nun eine Quelle, die verschränkte Zustände präpariert. Solch verschränkungsbasierte Varianten sind in der Sicherheitsanalyse von Protokollen üblich, denn es genügt die Zustände ρ_{AB} zu beschreiben, die Alice und Bob haben, bevor sie ihre Messungen durchführen. Dabei ist zu beachten, dass ein Zuhörer (Eve) die Zustände durch Interaktion mit dem an Bob gesendeten Teil beeinflusst. Dagegen bleibt der Teil ρ_A, der in Alices Labor verbleibt, unbeeinflusst. Nach der Attacke von Eve haben Alice und Bob einen unbekannten gemischten Zustand, da sie die Strategie von Eve nicht kennen. Der erste Schritt zur Bestimmung der Schlüsselrate ist somit die so genannte Parameterabschätzung (parameter estimation). Dabei sollen die beim Senden der Qubits aufgetretenen Fehler bestimmt werden. Die übliche Vorgehensweise ist, dass Alice und Bob zufällig Qubits auswählen und ihre Werte miteinander Vergleichen. Damit kann der von Alice und Bob gehaltene Zustand nach der Interaktion teilweise bestimmt werden. Da im weiteren Verlauf die unbekannten Einträge der Dichtematrix und die damit kompatiblen Zustände von besonderem Interesse sind, ist die folgende Definition hilfreich.

Definition 2.2. Die Menge Γ von zweiparteien Zuständen enthält alle Zustände ρ_{AB}, die mit der bestimmten Wahrscheinlichkeitsverteilung nach der Parameterabschätzung kompatibel sind.

Alice und Bob kennen, wie zuvor erwähnt, Eves Abhörstrategie nicht. Zur Bestimmung der Schlüsselrate ist es allerdings notwendig, diese zumindest auf eine der drei üblichen Arten von Angriffen, die in der Quanten Schlüsselverteilung benutzt werden, zu beschränken. In dieser Arbeit verwendet Eve kollektive Attacken. Dabei besitzt Eve ein eigenes Quantensystem ρ_E. Dieses verbindet Eve mit dem an Bob gesendeten Teil von $\rho_{AS} = |\phi\rangle\langle\phi|_{AS}$ und interagiert durch eine unitäre Transformation U_{SE} mit diesen Signalzuständen. Äquivalent zu dieser unitären Transformation kann man den Zustand auch mit Hilfe einer Reinigung (purification) des von Alice und Bob gehaltenen Zustandes ρ_{AB} beschreiben.

Satz 2.3 (Reinigung)**.** Es sei ρ ein gemischter Zustand in einem endlichen Hilbertraum $\mathcal{H}_A$ und $\mathcal{H}_B$ Hilbertraum von der gleichen Dimension wie $\mathcal{H}_A$.
Dann existiert ein reiner Zustand $|\psi\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$, so dass gilt $\mathrm{tr}_B(|\psi\rangle\langle\psi|) = \rho$. $|\psi\rangle$ heißt dann Reinigung von ρ.

Beweis. Sei ρ ein Zustand in $\mathcal{H}_A$ mit der Dimension n. Die Spektralzerlegung dieses Zustands ist $\rho = \sum_{i=1}^{n} \lambda_i |i\rangle\langle i|$ mit der Basis $\{|i\rangle \mid i = 1, \ldots, n\}$. Sei $\mathcal{H}_B$ ein Hilbertraum mit Dimension n

und einer Basis $\{|i'\rangle \mid i' = 1, \ldots, n\}$. Mit einem Zustand $|\psi\rangle = \sum_{i=1}^{n} \sqrt{\lambda_i}\, |i\rangle_A \otimes |i'\rangle_B$ in $\mathcal{H}_A \otimes \mathcal{H}_B$ folgt dann

$$\begin{aligned} \mathrm{tr}_B\left(|\psi\rangle\langle\psi|\right) &= \mathrm{tr}_B\left(\sum_{i,j=1}^{n} \sqrt{\lambda_i}\sqrt{\lambda_j}\, |i\rangle\langle j| \otimes |i'\rangle\langle j'|\right) = \sum_{i,j=1}^{n} \delta_{ij}\sqrt{\lambda_i}\sqrt{\lambda_j}\, |i\rangle\langle j| \\ &= \sum_{i=1}^{n} \lambda_i\, |i\rangle\langle i| = \rho. \end{aligned}$$

□

Alice, Bob und Eve haben also gemeinsam einen reinen Zustand $|\phi\rangle_{ABE}$. Mit diesem Resultat ist es nun leicht möglich, die Ausgangssituation zu erhalten, die für den Sicherheitsbeweis aus Ref. [DW05] benötigt wird. In ihm wird eine untere Schranke für die erreichbare sichere Schlüsselrate bei Protokollen mit Ein-Weg-Kommunikation für den Fall bestimmt, dass der Abhörer (Eve) kollektive Attacken benutzt. Wir beginnen mit einem Zustand des Gesamtsystems, bei dem Alice und Bobs Zustände klassische Zustände sind und Eves Zustand ein Quantenzustand ist (*ccq-Zustand*. Der Eingangszustand der Systeme von Alice, Bob und Eve ρ_{ABE} hat die Form

$$\rho_{XYE} = \sum_{x\in\mathcal{X},\ y\in\mathcal{Y}} P(x,y)\, |x\rangle\langle x|_X \otimes |y\rangle\langle y|_Y\, \rho_E^{x,y}. \tag{2.7}$$

Hier sind die Zustände $|x\rangle$ und $|y\rangle$ Mengen orthonormaler Basis-Zustände. Die Zustände $\rho_E^{x,y}$ sind die resultierenden Zustände, falls Alice einen bestimmen Wert x misst. Dieser Eingangszustand ergibt sich aus dem Zustand $\rho_{ABE} = |\phi\rangle\langle\phi|_{ABE}$, indem Alice ihre allgemeine Messung $\mathcal{A}$ durchführt und Bob die erhaltenen Signalzustände mit einer Messung $\mathcal{B}$ misst

$$\sum_{x,y} \mathrm{tr}_{A,B}\left(A_x \otimes B_y \otimes \mathbb{1}\ \rho_{ABE}\right) |x\rangle\langle x|_X \otimes |y\rangle\langle y|_Y = \sum_{x,y} p(x,y)\, |x\rangle\langle x|_X \otimes |y\rangle\langle y|_Y \otimes \rho_E^{x,y}, \tag{2.8}$$

mit dem auf die Messergebnisse von Alice und Bob konditionierte Zustand $\rho_E^{x,y}$.
Die resultierende sichere Schlüsselrate ist nun in Ref. [DW05] gegeben

$$r(\rho_{XYE}) = I(X:Y) - \chi(X:E). \tag{2.9}$$

Dabei ist $I(X:Y)$ die gemeinsame Information von Alice und Bob und $\chi(X:E)$ die gemeinsame Information von Alice und Eve. Es ist zu beachten, dass die Information von Alice und Bob klassische Information ist, während die Information von Eve Holevo-Information ist. Die klassische gemeinsame Information ist mit Hilfe von $I(X:Y) = H(X) + H(Y) - H(X,Y)$ leicht zu bestimmen, wobei $H(X) = -\sum_x p(x)\log_2 p(x)$ die Shannon-Entropie ist. Die gemeinsame

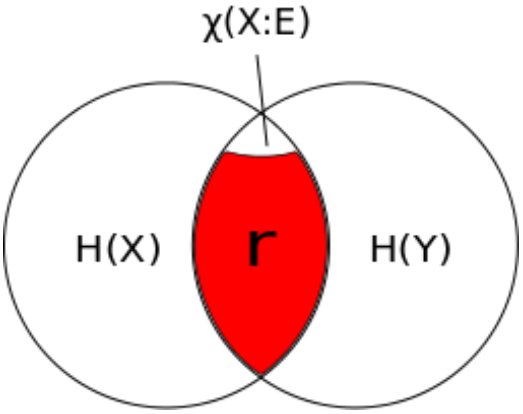

Abbildung 2.2: Der rote Bereich stellt die Schlüsselrate als gemeinsame Information von Alice und Bob und abzüglich der gemeinsamen Information mit Eve dar.

Holevo-Information lässt sich folgendermaßen beschreiben $\chi(X:E) = H(X)+S(E)-S(X,E)$, wobei $S(E)$ dabei die von-Neumann-Entropie $S(E) = -\mathrm{tr}\,(\rho_E \log_2 \rho_E)$ ist. Mit der allgemeinen Eigenschaft bedingter Entropien $S(E|X) = S(X,E) - H(X)$ folgt zusammen mit dem Ausgangszustand aus Gleichung (2.7) die Holevo-Information

$$\chi(X:E) = S(E) + S(E|X) = S(\rho_E) - \sum_x p(x) S(\rho_E^x), \tag{2.10}$$

wobei ρ_E^x der auf das von Alice gemessene Ergebnis x konditionierte Zustand von Eve ist. Insgesamt folgt für die Schlüsselrate damit

$$r(\rho_{XYE}) = H(X) + H(Y) - H(X,Y) - S(\rho_E) + \sum_x p(x) S(\rho_E^x). \tag{2.11}$$

Mit dieser Form der Schlüsselrate ist es nun möglich diese für den Fall zu berechnen, dass Eves Attacke und der Zustand von Eve bekannt ist. Da dies im allgemeinen nicht der Fall ist, soll zunächst der Zustand von Eve durch eine bestimmbare Größe ersetzt werden.

Satz 2.4. Sei $|\phi\rangle_{AB}$ ein reiner Zustand in $\mathcal{H}_A \otimes \mathcal{H}_B$. Dann gilt für die von-Neumann-Entropie der Teilsysteme $S(\rho_A) = S(\rho_B)$.

Beweis. Sei $|\phi\rangle_{AB}$ ein reiner Zustand in $\mathcal{H}_A \otimes \mathcal{H}_B$. Dieser lässt sich in der Schmidtform $|\phi\rangle_{AB} = \sum_i \alpha_i |u_i\rangle \otimes |v_i\rangle$ darstellen. Dann haben die Dichtematrizen der Teilsysteme $\rho_A = \alpha_i^2 |u_i\rangle\langle u_i|$ und $\rho_B = \alpha_i^2 |v_i\rangle\langle v_i|$ die selben Eigenwerte. Da die Entropie $S(\rho) = -\sum_i \lambda_i \log_2(\lambda_i)$ nur aus den Eigenwerten λ_i bestimmt wird, gilt für die Entropien der Teilsysteme $S(\rho_A) = S(\rho_B)$. □

Mit diesem Satz lässt sich die Schlüsselrate alleine in Termen mit Zuständen von Alice und Bob ausdrücken, denn sowohl der Zustand $|\psi\rangle_{ABE}$ als auch die Zustände $\rho^x_{BE} = \frac{1}{p(x)}\mathrm{tr}_A\left(A_x \otimes \mathbb{1} \otimes \mathbb{1}\,|\phi\rangle\langle\phi|_{ABE}\right)$ sind reine Zustände. Es gilt für die Schlüsselrate

$$r(\rho_{AB}) = H(X) + H(Y) - H(X,Y) - S(\rho_{AB}) + \sum_x p(x) S(\rho^x_B). \tag{2.12}$$

Bei diesem Ergebnis sind allerdings noch einige weitere Punkte zu beachten.
Um die Schlüsselrate exakt zu bestimmen, muss der Zustand von Alice und Bob ρ_{AB} vollständig bekannt sein. Da dies bei fast allen Protokollen nicht der Fall ist, kann mit den bekannten Parametern des Zustandes ρ_{AB} nur eine untere Grenze für die in diesem Fall optimale Attacke von Eve bestimmt werden.

Definition 2.5. Die Menge Γ ist die Menge aller Zweiparteienzustände, die den bekannten Parametern der Dichtematrix, das heißt der Wahrscheinlichkeitsverteilung $p(x,y)$ und dem reduzierten Zustand ρ_A, entsprechen.

Die minimale Schlüsselrate folgt dann mit

$$r_{\min} = \inf_{\rho_{AB}\in\Gamma} \left[H(X) + H(Y) - H(X,Y) - S(\rho_{AB}) + \sum_x p(x) S(\rho^x_B) \right]. \tag{2.13}$$

Damit kann nun, für den Fall, dass die Schlüsselrate optimierbar ist, eine untere Schranke für die Schlüsselrate alleine aus Kenntnis der Messungen von Alice und Bob und einiger Einträge des Zustandes ρ_{AB} bestimmt werden. Allerdings wurde hier ein weiterer wichtiger Punkt der Bestimmung der Schlüsselrate eines Protokolls noch nicht betrachtet und das ist die Nachauswahl (*Post Selection*) der Daten, die im ersten Kapitel beschrieben wurde. Wichtig ist hier vor allem das Aussieben der Basen (*Base Sifting*). In allen hier verwendeten Protokollen wird nur ein Teil der gesendeten Qubits verwendet um daraus einen Schlüssel zu generieren. Um eine Form der Schlüsselrate zu erhalten, in die dies mit einfließt, werden wir diese für jede im Protokoll gewählte Basis separat berechnen [FL12].

2.2.1 Die Schlüsselrate nach der Nachauswahl

Am Beispiel des BB84 Protokolls aus Kapitel 1.2 wird deutlich, wie das Base Sifting abläuft. Nach der Messung von A_x und B_y an jedem Qubit, bei der man die Daten x bzw. y erhält, wird die Messung einer der Basen zugeordnet. Dies kann mit $f(x) = v$ bzw. $f(y) = w$ beschrieben

werden, wobei im Fall des BB84

$$f(x) = \begin{cases} z & \text{, falls } x \in 0,1 \\ x & \text{, falls } x \in +,\text{-} \end{cases}, \tag{2.14}$$

ist. v und w sind die Ergebnisse, die über einen öffentlichen Kanal ausgetauscht werden und enthalten normalerweise keine weitere Information. Alle Ereignisse, die nicht in der gleichen Basis gemessen wurden und damit also nicht $v = w$ gilt, werden nun gestrichen. Für Ergebnisse, die in der gleichen Basis $v = w\hat{u}$ gemessen wurden, wird in jeder Basis eine Schlüsselrate ermittelt. Die POVMs M_A und M_B können in die Untermengen $m_A^v = \{A_x \ : \ f(x) = v\}$ und $m_B^w = \{B_y \ : \ f(y) = w\}$ eingeteilt werden, die der Messung in der jeweiligen Basis entsprechen. Um das öffentliche Austauschen der Information und das Aussortieren der Ereignisse quantenmechanisch zu beschreiben, werden Kraus-Operatoren $K_u \otimes L_u$ verwendet, die den Zustand des Systems ρ_{AB} auf den Zustand nach dem Aussortieren abbilden. Für die Kraus-Operatoren gilt $K_u \otimes L_u = \sqrt{\sum_{m_A^u} A_x} \otimes \sqrt{\sum_{m_B^u} B_y}$ mit $\sum_u K_u^\dagger K_u \otimes L_u^\dagger L_u \leq \mathbb{1}$. Die Wahrscheinlichkeit, dass der Zustand ρ_{AB} nach dem Aussortieren erhalten bleibt, ist p_{kept}.
Nach dem Aussortieren ist allen Parteien bekannt, in welcher Basis gemessen wurde, da diese Basis über einen öffentlichen Kanal kommuniziert wurde. Damit ist der Zustand, den Alice, Bob und Eve halten

$$\Psi = \sum_u p(u) \left|\Psi_u\right\rangle\!\left\langle\Psi_u\right|_{ABE} \otimes \left|u\right\rangle\!\left\langle u\right|_R, \tag{2.15}$$

wobei das System R ein klassisches Register darstellt, das die öffentliche Information enthält. $\left|\Psi_u\right\rangle = \frac{1}{\tilde{p}(u)} K_u \otimes L_u \otimes \mathbb{1}_E \left|\Psi\right\rangle$ ist der auf die Basis u konditionierte Zustand. Nachdem nun die Messbasis bekannt gegeben wurde, folgt das neue POVM aus der Normalisierung der POVM Elemente, die der Messung in der jeweiligen Basis entsprechen

$$M_A^u = \{A_x^u\} = \{K_u^{-1} A_x K_u^{-1\dagger} \ : \ A_x \in m_A^u\}, \tag{2.16}$$

$$M_B^u = \{B_y^u\} = \{L_u^{-1} B_y L_u^{-1\dagger} \ : \ B_y \in m_B^u\}. \tag{2.17}$$

Dabei ist zu beachten, dass $\sum_{m_A^u} K_u^{-1} A_x K_u^{-1\dagger} = \sum_{m_A^u} K_u^{-1} K_u K_u^\dagger K_u^{-1\dagger} = \mathbb{1}$ und entsprechend $\sum_{m_B^u} L_u^{-1} B_y L_u^{-1\dagger} = \sum_{m_B^u} L_u^{-1} L_u L_u^\dagger L_u^{-1\dagger}$ gilt. Es kann leicht gezeigt werden, dass die neue Messung nach der Nachauswahl der alten Messung entspricht

$$\begin{aligned}
&\mathrm{tr}_{AB}\left(A_x^u \otimes B_y^u \otimes \mathbb{1}_E \left|\Psi_u\right\rangle\!\left\langle\Psi_u\right|\right) \\
=&\mathrm{tr}_{AB}\left(K_u^{-1} A_x K_u^{-1\dagger} \otimes L_u^{-1} B_y L_u^{-1\dagger} \otimes \mathbb{1}_E \left(\frac{1}{\tilde{p}(u)} K_u \otimes L_u \otimes \mathbb{1}_E \left|\Psi\right\rangle\!\left\langle\Psi\right| K_u^\dagger \otimes L_u^\dagger \otimes \mathbb{1}_E\right)\right) \\
=&\frac{1}{\tilde{p}(u)}\ \mathrm{tr}_{AB}\left(A_x \otimes B_y \otimes \mathbb{1}_E \left|\Psi\right\rangle\!\left\langle\Psi\right|\right). \qquad (2.18)
\end{aligned}$$

Wie in Gleichung (2.7) transformiert die Messung den Zustand in einen ccq-Zustand

$$\left|\Psi_u\right\rangle\!\left\langle\Psi_u\right| \to \rho_{XYE}^u = \sum_{M^u} p_u(x,y) \left|x,y\right\rangle\!\left\langle x,y\right| \otimes \rho_E^{xy} \qquad (2.19)$$

mit $p_u(x,y) = \mathrm{tr}\left(A_x^u \otimes B_y^u \otimes \mathbb{1}_E \left|\Psi_u\right\rangle\!\left\langle\Psi_u\right|\right) = \frac{1}{\tilde{p}(u)}\mathrm{tr}\left(A_x \otimes B_y \otimes \mathbb{1}_E \left|\Psi\right\rangle\!\left\langle\Psi\right|\right) = \frac{p(x,y)}{\tilde{p}(u)}$ und den konditionierten Zuständen $\rho_E^{xy} = \frac{1}{p(x,y)}\mathrm{tr}_{AB}\left(A_x \otimes B_y \otimes \mathbb{1} \left|\Psi\right\rangle\!\left\langle\Psi\right|\right)$. Damit kann nun die Schlüsselrate für jede Messbasis nach dem Herausfiltern der in gleicher Basis gemessenen Ergebnisse unabhängig bestimmt und daraus die gesamte Schlüsselrate errechnet werden. Für die untere Grenze der Schlüsselrate ergibt sich

$$r_{\min} = \inf_{\rho_{AB}\in\Gamma} \left[\bar{r}(\rho_{AB})\right] = \inf_{\rho_{AB}\in\Gamma} \left[\sum_u p(u) r(\rho_{XYE}^u)\right]. \qquad (2.20)$$

Bei einer optimalen Attacke von Eve wird diese untere Grenze erreicht. Da der Zustand ρ_{AB} nach der Attacke von Eve nicht vollständig bekannt ist, muss, wie schon beschrieben, über die möglichen Zustände optimiert werden, um die untere Grenze der Schlüsselrate zu bestimmen. Hier soll mit Hilfe konvexer Optimierung diese untere Grenze gefunden werden. Dazu muss die Schlüsselrate eine konvexe Funktion der Zustände ρ_{AB} sein. Diese Eigenschaft wird im nächsten Kapitel gezeigt.

Kapitel 3

Konvexität der Schlüsselrate

Nachdem eine optimierbare untere Grenze der Schlüsselrate gefunden ist, muss nun gezeigt werden, dass diese konvex ist und durch Verfahren aus der Konvexen Optimierung bestimmt werden kann. Dazu wird zunächst, wie in Ref. [FL12], nur der klassische Teil der Schlüsselrate betrachtet. Damit die in Gleichung (2.12) angegebene Schlüsselrate konvex ist, muss der klassische Teil dieser ebenfalls konvex sein. Wichtig ist dabei die Bedingung, dass der Teil des Zustands, der in Alice Labor verbleibt, sich nicht verändert. Daraus folgt als Voraussetzung, dass die Wahrscheinlichkeitsverteilung $p(x) = \mathrm{tr}\,(A_x \rho_A)$ konstant bleibt. Mit dieser Voraussetzung ist nun leicht zu zeigen, dass dieser Teil der Schlüsselrate konvex ist.

Theorem 3.1 (Konvexität der klassischen Information)**.** *Gegeben seien die Zustände ρ_{AB}, σ_{AB} und die konvexe Summe $\bar{\rho}_{AB} = \lambda\rho_{AB} + (1-\lambda)\sigma_{AB}$ für $\lambda \in [0,1]$ mit der Wahrscheinlichkeitsverteilung $p(x,y) = tr\,(A_x \otimes B_y\ \rho_{AB})$, $q(x,y) = tr\,(A_x \otimes B_y\ \sigma_{AB})$ und $\bar{p}(x,y) = \lambda p(x,y) + (1-\lambda)q(x,y)$. Falls die Wahrscheinlichkeitsverteilung $p(x) = q(x)$ für alle x erfüllt, dann ist die klassische Information konvex*

$$I(X:Y)_{\bar{p}} \leq \lambda I(X:Y)_p + (1-\lambda) I(X:Y)_q. \tag{3.1}$$

Beweis. Für die klassische gemeinsame Information gilt, wie schon zuvor erwähnt, $I(X:Y) = H(X) + H(Y) - H(X,Y) = H(X) - H(X|Y)$. Beispielhaft für die Wahrscheinlichkeitsverteilung $p(x,y)$ ist die Shannon-Entropie $H(X)_p = H(p(x)) = -\sum_x p(x)\log p(x)$ und die konditionierte Entropie $H(X|Y)_p = \sum_{x,y} p(x,y)\log\left(\frac{p(x,y)}{p(y)}\right)$. Zunächst folgt aus $p(x) = q(x)$ der Zusammenhang $H(p(x)) = H(q(x)) = H(\bar{p}(x))$. Es muss also nur noch gezeigt werden, dass $H(X|Y)_{\bar{p}} = \lambda H(X|Y)_p + (1-\lambda)H(X|Y)_q$ gilt. Dies folgt aus der Jensenschen Ungleichung bzw.

der Log-Sum Ungleichung [CT91]

$$\sum_i a_i \left(\frac{a_i}{b_i}\right) \geq \sum_i a_i \log \frac{\sum_j a_j}{\sum_k b_k}, \tag{3.2}$$

mit der dann für die konditionierte Entropie gilt

$$\begin{aligned} H(X|Y)_{\bar{p}} &= [\lambda p(x,y) + (1-\lambda)q(x,y)] \ \log \left[\frac{\lambda p(x,y) + (1-\lambda)q(x,y)}{\lambda p(y) + (1-\lambda)q(y)}\right] \\ &\leq \lambda p(x,y) \log \frac{\lambda p(x,y)}{\lambda p(y)} + (1-\lambda)q(x,y) \log \frac{(1-\lambda)q(x,y)}{(1-\lambda)q(y)} \\ &= \lambda p(x,y) \log \frac{p(x,y)}{p(y)} + (1-\lambda)q(x,y) \log \frac{q(x,y)}{q(y)} \\ &= \lambda H(X|Y)_p + (1-\lambda) H(X|Y)_q. \end{aligned} \tag{3.3}$$

Damit ist die Konvexität für den klassischen Teil der Schlüsselrate gezeigt. □

Damit nun die Schlüsselrate aus Gleichung (2.12) im Gesamten konvex ist, muss nun noch gezeigt werden, dass der quantenmechanische Teil, die Holevo Größe $\chi(X:E)$ aus Gleichung (2.10), konkav ist.

Theorem 3.2 (Konkavität der Holevo-Information)**.** *Gegeben seien die Zustände ρ_{AB}, σ_{AB} und die konvexe Summe $\bar{\rho}_{AB} = \lambda\rho_{AB} + (1-\lambda)\sigma_{AB}$ für $\lambda \in [0,1]$ mit der Wahrscheinlichkeitsverteilung $p(x,y) = tr\left(A_x \otimes B_y \ \rho_{AB}\right)$, $q(x,y) = tr\left(A_x \otimes B_y \ \sigma_{AB}\right)$ und $\bar{p}(x,y) = \lambda p(x,y) + (1-\lambda)q(x,y)$. Dann ist die Holevo Größe $\chi(X:E)$ konkav*

$$\chi(X:E)_{\bar{p}} \geq \lambda\chi(X:E)_p + (1-\lambda)\chi(X:E)_q. \tag{3.4}$$

Beweis. Die gegebenen Zustände ρ_{AB} und σ_{AB} haben eine Reinigung $|\Psi\rangle_{ABE}$ und $|\Sigma\rangle_{ABE}$. Nach der Messung $\mathcal{M}_A$ von Alice befinden sich diese Zustände in dem cq-Zustands

$$|\Psi\rangle \to \rho_{XE} = \sum_x p(x) \ \ |x\rangle\langle x| \otimes \rho_E^x, \tag{3.5}$$

$$|\Sigma\rangle \to \sigma_{XE} = \sum_x q(x) \ \ |x\rangle\langle x| \otimes \sigma_E^x, \tag{3.6}$$

$$\tag{3.7}$$

mit den konditionierten Zuständen $\rho_E^x = \frac{1}{p(x)} \mathrm{tr}_{AB}\left(A_x \otimes \mathbb{1}_{BE} \ \ |\Psi\rangle\langle\Psi|\right)$ und $\sigma_E^x = \frac{1}{q(x)} \mathrm{tr}_{AB}\left(A_x \otimes \mathbb{1}_{BE} \ \ |\Sigma\rangle\langle\Sigma|\right)$. Die konvexe Summe der Zustände ist ein gemischter Zustand

$\rho_{ABE} = \lambda \left|\Psi\right\rangle \left\langle\Psi\right| + (1-\lambda) \left|\Sigma\right\rangle \left\langle\Sigma\right|$. Dieser Zustand kann wiederum durch Anhängen eines Hilfssystems F gereinigt werden

$$\left|\bar{\Psi}\right\rangle_{ABEF} = \sqrt{\lambda} \left|\Psi\right\rangle \left|0\right\rangle_F + \sqrt{1-\lambda} \left|\sigma\right\rangle \left|1\right\rangle_F . \tag{3.8}$$

Nach der Messung $\mathcal{M}_A$ von Alice ergibt sich

$$\left|\bar{\Psi}\right\rangle \to \bar{\rho}_{XEF} = \sum_x \bar{p}(x) \left|x\right\rangle \left\langle x\right| \otimes \bar{\rho}^x_{EF} \tag{3.9}$$

mit dem konditionierten Zustand $\bar{\rho}^x_{EF} = \frac{1}{\bar{p}(x)} \mathrm{tr}_{AB} \left(A_x \otimes \mathbb{1}_{BEF} \ \left|\bar{\Psi}\right\rangle \left\langle\bar{\Psi}\right|\right)$.
Es sei $\mathcal{M}$ eine spurerhaltende Operation $\mathcal{M} : F \to F'$ und in diesem Fall eine Messung in der Standardbasis $\{\left|0\right\rangle, \left|1\right\rangle\}$. Damit werden alle Offdiagonalterme in der Dichtematrix $\bar{\rho}_{XEF'}$ in F' vernichtet

$$\begin{aligned}
\bar{\rho}_{XEF} \mapsto \Bigg(& \sum_x \bar{p}(x) \left|x\right\rangle \left\langle x\right| \\
& \otimes \frac{1}{\bar{p}(x)} \mathrm{tr}_{AB} \left(A_x \otimes \mathbb{1}_{BEF} \ \left[\lambda \left|\Psi\right\rangle \left\langle\Psi\right| \otimes \left|0\right\rangle \left\langle 0\right| + (1-\lambda) \left|\Sigma\right\rangle \left\langle\Sigma\right| \otimes \left|1\right\rangle \left\langle 1\right|\right]\right) \Bigg) \\
& \sum_x \bar{p}(x) \left|x\right\rangle \left\langle x\right| \otimes \frac{1}{\bar{p}(x)} \left[\lambda p(x) \rho^x_E \otimes \left|0\right\rangle \left\langle 0\right| + (1-\lambda) q(x) \sigma^x_E \otimes \left|1\right\rangle \left\langle 1\right|\right] .
\end{aligned} \tag{3.10}$$

Mit $q(x) = \frac{\bar{p}(x) - \lambda p(x)}{1-\lambda}$ und der Definition $\lambda_x = \lambda \frac{p(x)}{\bar{p}(x)}$ folgt daraus

$$\bar{\rho}_{XEF} \mapsto \sum_x \bar{p}(x) \left|x\right\rangle \left\langle x\right| \otimes \left[\lambda_x \rho^x_E \otimes \left|0\right\rangle \left\langle 0\right| + (1-\lambda_x) \sigma^x_E \otimes \left|1\right\rangle \left\langle 1\right|\right] . \tag{3.11}$$

Nach dem gemeinsamen Entropie Theorem aus Ref. [NC00] folgt für die Holevo-Information

$$\begin{aligned}
\chi(X : EF') &= \sum_x \bar{p}(x) S(\bar{\rho}^x_{EF'}) \\
&= \sum_x \bar{p}(x) \ S(\lambda_x \rho^x_E \otimes \left|0\right\rangle \left\langle 0\right| + (1-\lambda_x) \sigma^x_E \otimes \left|1\right\rangle \left\langle 1\right|) \\
&= \sum_x \left[\lambda p(x) \ S(\rho^x_E) + (1-\lambda) q(x) \ S(\sigma^x_E) + \bar{p}(x) h(\lambda_x)\right] \\
&= \lambda \chi(X : E)_\rho + (1-\lambda) \chi(X : E)_\sigma + \sum_x \bar{p}(x) h(\lambda_x) \\
&= \lambda \chi(X : E)_\rho + (1-\lambda) \chi(X : E)_\sigma + h(\lambda) - \sum_x \bar{p}(x) h(\lambda_x) \\
&\geq \lambda \chi(X : E)_\rho + (1-\lambda) \chi(X : E)_\sigma .
\end{aligned} \tag{3.12}$$

Da eine generelle Messung wie in Gleichung (3.10) die Entropie nicht erhöhen kann [NC00], ergibt sich hieraus die Konkavität der Holevo-Information

$$\chi(X : EF) \geq \chi(X : EF') \geq \lambda\chi(X : E)_\rho + (1 - \lambda)\chi(X : E)_\sigma. \tag{3.13}$$

□

Die klassische Information ist nach Theorem 3.1 konvex und die Holevo-Information nach Theorem 3.2 konkav.
Zusammen resultiert hieraus die Konvexität der Schlüsselrate.

Kapitel 4

Konvexe Optimierung

Nachdem nun festgestellt wurde, dass die Schlüsselrate eine konvexe Funktion des Zustandes zwischen Alice und Bob ist, ist der nächste Schritt die Konvexe Optimierung. Das Ziel der Konvexen Optimierung ist es, eine konvexe Funktion $f(x)$ für einen bestimmten Funktionswert $x \in X$ unter Nebenbedingungen zu minimieren bzw. zu maximieren. Diese Arbeit orientiert sich dabei an Ref. [BV04] und Ref. [GK02].

Definition 4.1 (Konvexes Problem)**.** Sei $X \in \mathbb{R}^n$ nichtleer, $f: X \to \mathbb{R}$ und $g_i: \mathbb{R}^n \to \mathbb{R}$ stetig differenzierbare konvexe Funktionen und $h_j(x): \mathbb{R}^n \to \mathbb{R}$ linear. Dann ist

$$\begin{array}{ll} \min & f(x) \\ \text{u.d.N.} & h_j(x) = 0, \quad j = 1, \ldots, p \\ & g_i(x) \leq 0, \quad i = 1, \ldots, m \end{array}$$

ein konvexes Optimierungsproblem.

Es ist natürlich entscheidend zu wissen, wann die optimale Lösung des Optimierungsproblems erreicht ist. Dabei spielen die *Karush-Kenn-Tucker-Bedingungen* eine wichtige Rolle.

Definition 4.2. Die Lagrange-Funktion des Optimierungsproblems ist

$$L(x, \lambda, \mu) := f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{p} \mu_j h_j. \tag{4.1}$$

Damit können nun die Karush-Kenn-Tucker-Bedingungen eingeführt werden.

Definition 4.3 (KKT-Bedingungen)**.** Betrachte ein allgemeines Optimierungsproblem

$$\begin{aligned} \min \quad & f(x) \\ \text{u.d.N.} \quad & h_j(x) = 0, \quad j = 1, \dots, p \\ & g_i(x) \leq 0, \quad i = 1, \dots, m \end{aligned}$$

mit differenzierbarem f, g und h. Dann heißen die folgenden Bedingungen *Karush-Kenn-Tucker-Bedingungen* (KKT-Bedingungen)

$$\begin{aligned} \nabla_x^2 L(x, \lambda, \mu) &= 0, \\ h(x) &= 0, \\ \lambda \geq 0,\ g(x) \geq 0,\ \lambda^T g(x) &= 0. \end{aligned} \tag{4.2}$$

Der Gradient der Lagrange-Funktion ist

$$\nabla_x^2 L(x, \lambda, \mu) = \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 g_i(x) + \sum_{j=1}^{p} \mu_j \nabla^2 h_j(x). \tag{4.3}$$

Jeder Vektor aus den Lagrange-Multiplikatoren $(x^*, \lambda^*, \mu^*) \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p$, der den KKT-Bedingungen genügt, heißt KKT-Punkt des Optimierungsproblems.

Es kann gezeigt werden, dass für jedes lokale Minimum x^* ein KKT-Punkt existiert. Bei einem konvexen Problem können die KKT-Bedingungen angepasst werden.

Satz 4.4 (KKT-Bedingungen eines konvexen Problems)**.** Betrachte ein konvexes Optimierungsproblem

$$\begin{aligned} \min \quad & f(x) \\ \text{u.d.N.} \quad & b_j^T x = \beta_j, \quad j = 1, \dots, p \\ & g_i(x) \leq 0, \quad i = 1, \dots, m \end{aligned}$$

mit differenzierbarem und konvexem f und g, gegebenen Vektoren $b_j \in \mathbb{R}^n$ und gegebenen Skalaren $\beta_j \in \mathbb{R}$. Dann gilt als KKT-Bedingungen

$$\begin{aligned} \nabla^2 f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla^2 g_i(x^*) + \sum_{j=1}^{p} \mu_j^* b_j = 0, \\ b_j^T x^* = \beta_j \ \forall j = 1, \ldots, p, \\ g_i(x^*) \leq 0 \ \forall i = 1, \ldots, m \\ \lambda_i^* g_i(x^*) = 0 \ \forall i = 1, \ldots, m \\ \lambda_i^* \geq 0 \ \forall i = 1, \ldots, m. \end{aligned} \tag{4.4}$$

Beweis. Siehe [GK02]. □

Ein großer Vorteil der Konvexen Optimierung ist, dass jedes lokale Minimum auch ein globales Minimum ist. Außerdem ist jeder KKT-Punkt eines konvexen Problems auch ein lokales, d.h. auch ein globales Minimum. Es muss also ein Weg gefunden werden, diese Punkte zu bestimmen. Bei der Konvexen Optimierung der Schlüsselrate liegt der Schwerpunkt auf den konvexen Problemen mit Ungleichheitsrestringtionen, also Probleme der Art

$$\begin{aligned} \min \quad & f(x) \\ \text{u.d.N.} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \ . \end{aligned}$$

Zur Lösung dieses Problems wird häufig eines der Innere-Punkt Verfahren verwendet, von denen einige im weiteren Verlauf kurz erläutert werden sollen.

4.1 Penalty-Methode

Ein klassisches Verfahren zur Lösung konvexer Probleme mit Gleichheitsrestringtionen

$$\begin{aligned} \min \quad & f(x) \\ \text{u.d.N.} \quad & h_j(x) = 0, \quad j = 1, \ldots, p \end{aligned}$$

ist die Penalty-Methode. Die Idee hinter diesem Verfahren ist es, dass Problem auf ein unrestringiertes Problem zu vereinfachen und dabei die Zielfunktion bei Verlassen des durch die Nebenbedingungen festgelegten zulässigen Bereiches zu bestrafen. Eine einfache Penaltyfunktion lautet

$$P(x; t) := f(x) + \frac{t}{2} \|h(x)\|^2 \tag{4.5}$$

mit dem Penalty-Parameter $t > 0$. Um die Penaltyfunktion genauer darzustellen, wird das

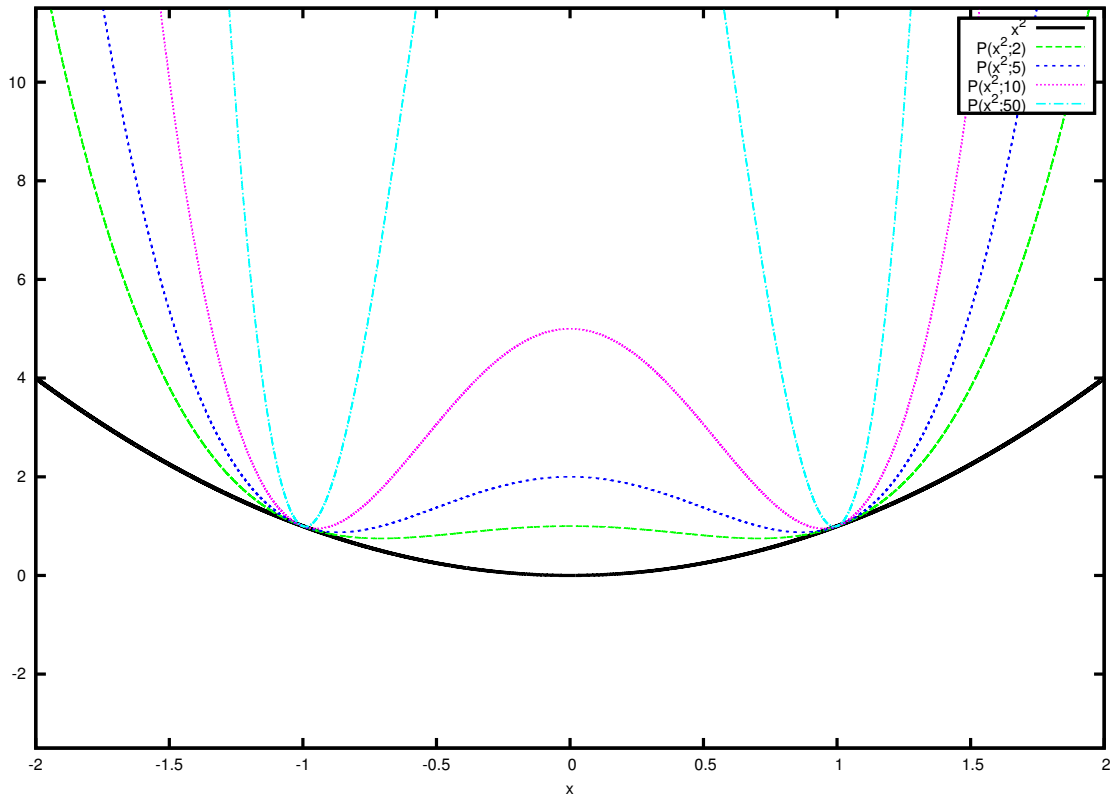

ABBILDUNG 4.1: Dargestellt ist die Penaltyfunktion $P(x^2;t) := x^2 + \frac{t}{2}\|x^2 - 1\|^2$ mit variierendem Penaltyparameter t. Deutlich nähern sich die lokalen Minima der Penaltyfunktion bei größer werdendem Penaltyparameter den zulässigen Punkten des Optimierungsproblems $x_1^* = -1$ und $x_2^* = +1$ an.

Beispiel

$$\begin{aligned} \min \quad & x^2 \\ \text{u.d.N.} \quad & x^2 - 1 = 0 \end{aligned}$$

in Abb. 4.1 dargestellt. Es wird deutlich, dass die Penaltyfunktion an den zulässigen Punkten mit der Zielfunktion $f(x)$ übereinstimmt und die lokalen Minima der Penaltyfunktion im Vergleich zum Minimum der Zielfunktion seitlich verschoben sind. Diese Minima nähern sich bei größer werdendem Penaltyparameter t zunehmend den zulässigen Punkten $x_1^* = -1$ und $x_2^* = 1$ an. Darauf stützt sich die Idee der Penalty-Methode. Es wird eine streng monotone Folge von Penaltyparametern t_k betrachtet und die dazugehörigen Minima der Penaltyfunktion berechnet. Diese sollen sich zunehmend den zulässigen Punkten der Zielfunktion annähern. Der Algorithmus der Penalty-Methode sieht also wie folgt aus

1. Wähle $t_0 > 0$, $\epsilon > 0$ und setze $k := 0$.

2. Bestimme $x^k \in \mathbb{R}^n$ als Lösung des unrestringierten Optimierungsproblems $\min P(x; t_k)$ mit $x \in \mathbb{R}^n$.

3. Ist $|h(x^k)| \leq \epsilon$, wird der Algorithmus gestoppt.

4. Bestimme $t_{k+1} > t_k$, setze $k = k + 1$ und beginne wieder bei Schritt 1.

Für diesen Algorithmus kann die Konvergenz der Penalty-Methode gezeigt werden.

Satz 4.5. Seien f und h stetig, $\{t_k\}$ streng monoton wachsend mit $t_k \to \infty$, die zulässige Menge $X := \{x \in \mathbb{R}^n \mid h(x) = 0\}$ eine durch den Algorithmus 5.5 erzeugte Folge. Dann gelten die folgenden Aussagen:

a) Die Folge der Zielfunktionswerte der Penaltyfunktion $\{P(x^k; t_k)\}$ ist monoton wachsend.
b) Die Folge der Abweichung der Nebenbedingung $\{\|h(x^k)\|\}$ ist monoton fallend.
c) Die Folge Zielfunktionswerte $\{f(x^k)\}$ ist monoton wachsend.
d) Es gilt $\lim_{k\to\infty} h(x^k) = 0$.
e) Jeder Häufungspunkt der Folge $\{x^k\}$ ist eine Lösung des Optimierungsproblems.

Beweis. Siehe Ref. [GK02] S.208.
(a) Wegen $t_k < t_{k+1}$ und der Definition von x^k gilt

$$P(x^k; t_k) \leq P(x^{k+1}; a_k) \leq P(x^{k+1}; t_{k+1}).$$

(b) Da $P(x^k; t_k) \leq P(x^{k+1}; t_k)$ und $P(x^{k+1}; t_{k+1}) \leq P(x^k; t_{k+1})$, ergibt sich

$$P(x^k; t_k) + P(x^{k+1}; t_{k+1}) \leq P(x^{k+1}; t_k) + P(x^k; t_{k+1}).$$

Mit der Definition von P folgt

$$t_k\|h(x^k)\|^2 + t_{k+1}\|h(x^{k+1})\|^2 \leq t_k\|h(x^{k+1})\|^2 + t_{k+1}\|h(x^k)\|^2,$$

und damit

$$(t_k - t_{k+1})\left(\|h(x^k)\|^2 - \|h(x^{k+1})\|^2\right) \leq 0.$$

Wegen $t_k < t_{k+1}$ ergibt sich somit $\|h(x^k)\| \geq \|h(x^{k+1})\|$ für alle $k \in \mathbb{N}$.
(c) Aus $P(x^k; t_k) \leq P(x^{k+1}; t_k)$ sowie (b) folgt

$$f(x^k) \leq f(x^{k+1}.$$

(d)Da X nichtleer ist, ergibt sich

$$f(x^k) \leq P(x^k; t_k) \leq \inf_{x \in X} P(x; t_k) = \inf_{x \in X} f(x) =: \tilde{f} < \infty.$$

Wegen $t_k \to \infty$ und $f(x^k) \geq f(x^0)$ nach Aussage (c) folgt $\lim_{k\to\infty} \|h(x^k)\| = 0$. □

Da nur die Stetigkeit von f und h benötigt wird, ist die Penalty-Methode auch auf Probleme mit Ungleichheitsrestringtionen $g_i(x) \leq 0,\ i = 1, \ldots m$ anwendbar. Diese Ungleichheitsrestringtionen können als Gleichheitsrestringtionen $\max\{0, g_i(x)\} = 0,\ i = 1, \ldots m$ geschrieben werden. Damit wird die Penaltyfunktion eines Optimierungsproblems

$$\begin{array}{ll} \min & f(x) \\ \text{u.d.N.} & h_j(x) = 0, \quad j = 1, \ldots, p \\ & g_i(x) \leq 0, \quad i = 1, \ldots, m \end{array}$$

zu

$$P(x; t) := f(x) + \frac{t}{2} \sum_{j=1}^{p} \|h_j(x)\|^2 + \frac{t}{2} \sum_{i=1}^{m} \left(\max\{0, g_i(x)\}\right)^2. \tag{4.6}$$

Mit dem Penalty-Verfahren kann außerdem, wenn $f(x)$ und $h(x)$ stetig differenzierbar sind, eine Folge von Lagrange-Multiplikatoren μ_k konstruiert werden, die zusammen mit der Folge $\{x^k\}$ gegen den KKT-Punkt (x^*, μ^*) konvergiert.

Satz 4.6. Seien $f : \mathbb{R}^n \to \mathbb{R}$ und $h : \mathbb{R}^n \to \mathbb{R}^p$ stetig differenzierbar, $\{x^k\}$ eine durch das Penalty-Verfahren erzeugte Folge mit $\lim_{k\to\infty} x^k = x^*$, die Gradienten $\nabla h_1(x), \ldots, \nabla h_p(x)$ linear unabhängig und μ^k die Folge $\mu_j^k := t_k h_j(x^k)$, dann gelten:

a) Die Folge $\{\mu^k\}$ konvergiert gegen einen Vektor $\mu^* \in \mathbb{R}^p$.
b) Das Paar (x^*, μ^*) mit μ^* aus a) ist ein KKT-Punkt des Optimierungsproblems, d.h. μ^* ist der eindeutig bestimmte Lagrange-Multiplikator zur Lösung x^* des Optimierungsproblems.

Beweis. Siehe Ref. [GK02] S.211.
(a) Seien $J_k := \frac{\partial h(x^k)}{\partial x} \in \mathbb{R}^{p\times n}$ und $J_* := \frac{\partial h(x^*)}{\partial x} \in \mathbb{R}^{p\times x}$ die Jakobimatrizen von h in x^k bzw. x^*. Aus der Stetigkeit der Jakobimatrizen gilt $J_k \to J_*$. Da die Gradienten $\nabla h_1(x^*), \ldots, \nabla h_p(x^*)$ ist die Matrix $J_* J_*^T \in \mathbb{R}^{p\times p}$ regulär. Für hinreichend große k ist somit auch $J_k J_k^T$ regulär und es

$$(J_k J_k^T)^{-1} \to (J_* J_*^T)^{-1}.$$

Da x^k ein Minimum von $P(x;t_k)$ ist, gilt

$$\begin{aligned}
0 &= \nabla_x P(x^k;t_k) \\
&= \nabla f(x^k) + t_k \sum_{j=1}^{p} h_j(x^k)\nabla h_j(x^k) \\
&= \nabla f(x^k) + \sum_{j=1}^{p} \mu_j^k \nabla h_j(x^k) \\
&= \nabla f(x^k) + J_k^T \mu^k.
\end{aligned}$$

Multiplikation von links mit J_k und Auflösen nach μ^k liefert dann $J_k J_k^T \mu^k = -J_k \nabla f(x^k)$ und damit konvergiert die Folge $\{\mu^k\}$ gegen ein μ^*

$$\mu_k = -(J_k J_k^T)^{-1} J_k \nabla f(x^k) \to -(J_* J_*^T)^{-1} J_* \nabla f(x^*) = \mu^*.$$

(b) Aus dem globalen Minimum x^k von $P(\cdot;t_k)$ aus dem folgt

$$0 = \nabla_x P(x^k;t_k) = \nabla f(x^k) + t_k \sum_{j=1}^{p} h_j(x^k)\nabla h_j(x^k).$$

Damit und mit $\mu_j^k = t_k h_j(x^k) \to \mu^*$ folgt die Aussage. □

Die Penalty-Methode hat den Nachteil, dass die iterierenden x^k auch außerhalb des zulässigen Bereichs liegen können. Für Funktionen, bei denen dieser Bereich nicht definiert ist, stellt dies ein Problem dar. Für solche Optimierungsprobleme sind Innere-Punkt Verfahren sinnvoll.

4.2 Innere-Punkt Verfahren

Die Innere-Punkt Verfahren sind eine Klasse von Standardverfahren zur Lösung konvexer Problemen mit Ungleichheitsrestringtionen. Dabei basieren sie auf Näherungs-
lösungen, die sich strikt im inneren des zulässigen Bereichs befinden. Es werden in diesem Kapitel zwei zentrale Konzepte der Innere-Punkt Verfahren, der Zentrale Pfad und die Barriere-Methode, verdeutlicht. Dabei beschränkt sich der Zentrale Pfad auf lineare Probleme, während die Barriere-Methode auch auf nicht-lineare Probleme angewendet werden kann.

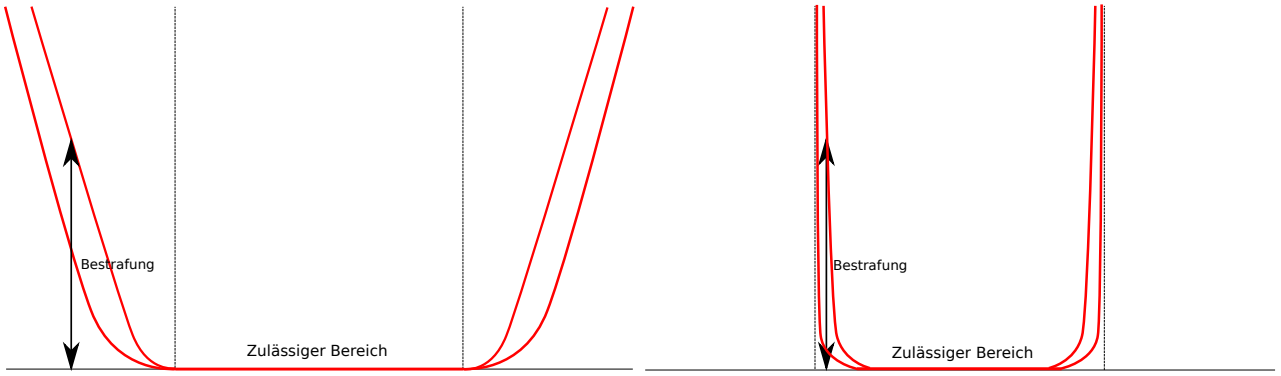

Abbildung 4.2

4.2.1 Barriere-Methode

Eine der bekannte Möglichkeit, ein konvexes Problemen mit Ungleichheitsrestringtionen in ein Problem mit Gleichheitsrestringtionen zu transformieren, ist das Finden einer Barrierefunktion. Der erste Schritt ist es, die Ungleichheitsrestringtionen in die Zielfunktion einzubauen. Dabei wird die sogenannte *Barrierefunktion* so gewählt, dass sich die im Optimierungsalgorithmus ermittelten Punkte x immer im zulässigen Bereich bewegen. Betrachtet man das Problem

$$\begin{aligned} \min \quad & f(x) \\ \text{u.d.N.} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \ , \end{aligned}$$

so kann als Barrierefunktion zum Beispiel die logarithmische Barrierefunktion

$$B(x,t) := f(x) - \alpha \sum_{i=1}^{m} \ln(-g_i(x)) \tag{4.7}$$

oder die inverse Barrierefunktion

$$B(x,t) := f(x) - \alpha \sum_{i=1}^{m} \frac{1}{g_i(x)}. \tag{4.8}$$

Beide Funktionen sind differenzierbar und damit kann das Newton Verfahren auf sie angewendet werden. Für den Zusatzterm der logarithmische Barrierefunktion kann außerdem die erste und

zweite Ableitung angegeben werden [BV04]

$$\nabla \left[\sum_{i=1}^{m} \ln(-g_i(x)) \right] = \sum_{i=1}^{m} \frac{1}{-g_i(x)} \nabla g_i(x), \tag{4.9}$$

$$\nabla^2 \left[\sum_{i=1}^{m} \ln(-g_i(x)) \right] = \sum_{i=1}^{m} \frac{1}{g_i(x)^2} \nabla g_i(x) \nabla g_i(x)^T + \sum_{i=1}^{m} \frac{1}{-g_i(x)} \nabla^2 g_i(x). \tag{4.10}$$

Ein einfaches Beispiel ist das Optimierungsproblem

$$\begin{aligned} \min \quad & f(x) = x^2 \\ \text{u.d.N.} \quad & x + 1 \leq 0, \quad i = 1, \ldots, m \ . \end{aligned}$$

Die dazugehörige logarithmische Barrierefunktion ist $B(x) = x^2 - \alpha \ln(-x - 1)$ und ist in Abb. 4.2 dargestellt. Der Algorithmus zur Lösung entspricht dem der Penalty-Methode. Sowohl die Penalty-Methode, wie auch die Barriere-Methode eignen sich zur Optimierung nichtlinearer Probleme. Die Barriere-Methode hat den Vorteil, dass sich die iterierenden Punkte x_k strikt im zulässigen Bereich bewegen.
Mit Hilfe der logarithmischen Barriere-Methode kann eine weitere Innere-Punkt Methode, das Problem des zentralen Pfades, genauer untersucht werden.

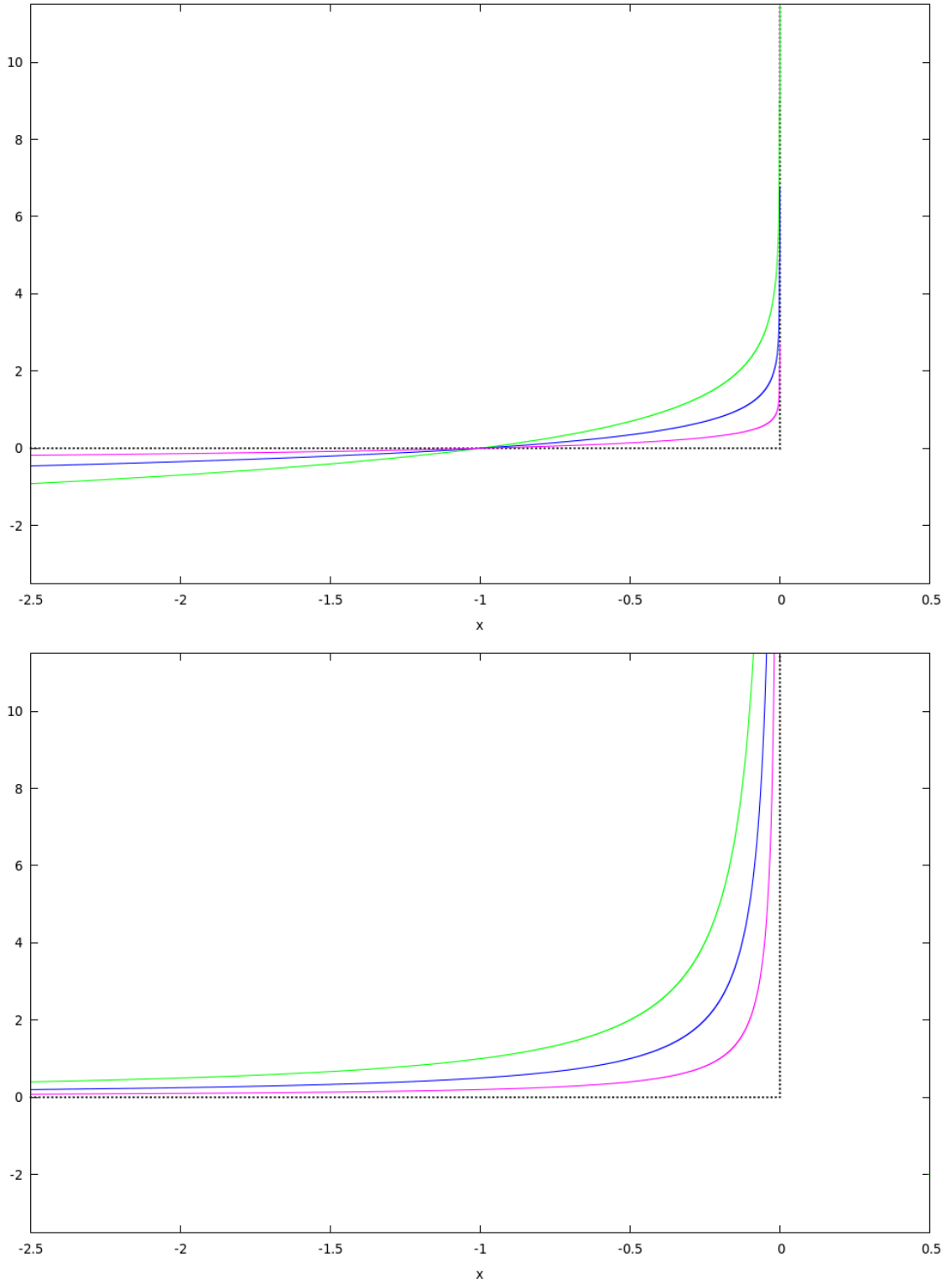

Abbildung 4.3: Die durchgehenden Linien stellen im oberen Bild den logarithmische Barriereterm mit dem Parameter $\alpha = 1.0$, 0.5 und 0.25 dar, im unteren den inversen Barriereterm. Die gestrichelte Linie ist ein idealer Barriereterm, der die Optimierung exakt auf einen gewünschten Bereich beschränkt, ohne den Funktionswert zu verändern. Bei kleinerem α nähert sich in beiden Fällen der Barriereterm immer weiter an den idealen Barriereterm an.

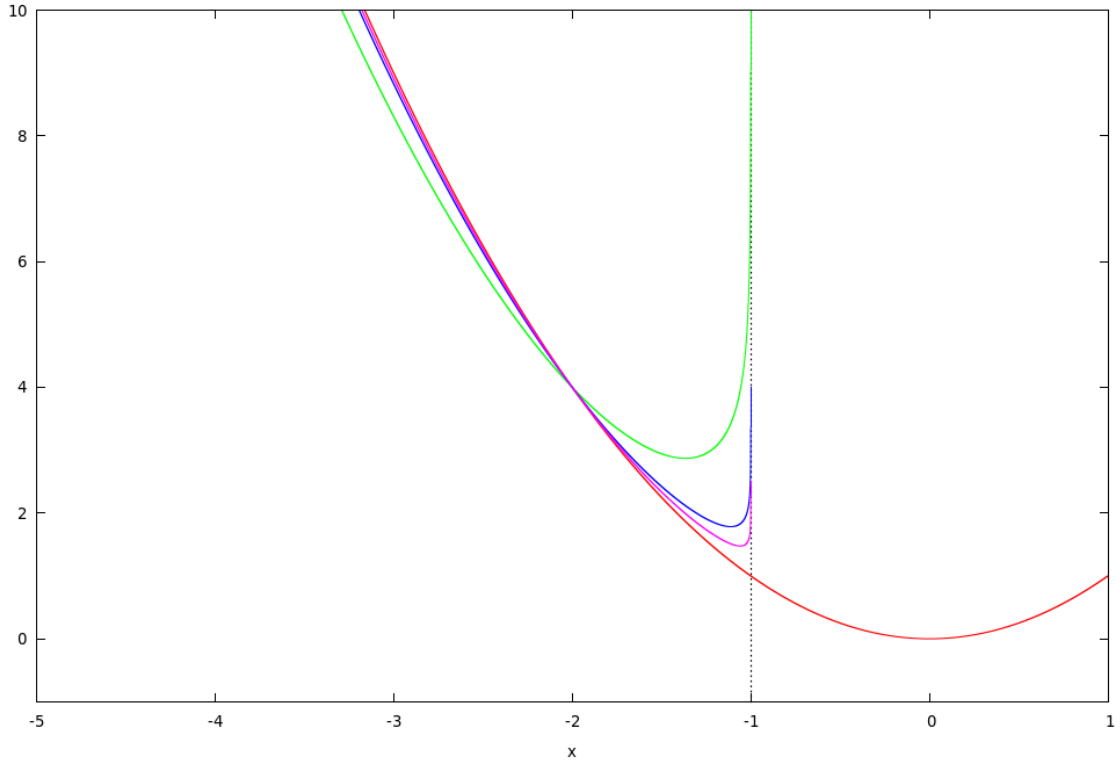

ABBILDUNG 4.4: Hier sind die Barrierefunktionen der Funktion $f(x) = x^2$ für die Parameter $\alpha = 1.0$, 0.25 und 0.125 dargestellt. Die gestrichelte Linie stellt die Nebenbedingung $x + 1 \leq 0$ dar.

4.2.2 Der Zentrale Pfad

Wir betrachten zu Beginn das lineare Programm

$$\begin{aligned} \min \quad & c^T x \\ \text{u.d.N.} \quad & Ax = b,\ x \geq 0 \end{aligned} \tag{4.11}$$

mit den Daten $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ und $b \in \mathbb{R}^m$. Mit der Lagrangefunktion (4.1)

$$L(x, \lambda, \mu) = c^T x + \lambda^T (-x) + \mu^T (b - Ax) \tag{4.12}$$

ergeben sich die KKT-Bedingungen 4.4

$$\begin{aligned} A^T \mu + \lambda &= c, \\ Ax &= b, \\ x &\geq 0, \\ \lambda &\geq 0, \\ \lambda_i x_i &\geq 0,\ i = 1, \dots n. \end{aligned} \tag{4.13}$$

Die Bedingung $x \geq 0$ aus dem Problem (4.11) lässt sich mit Hilfe der logarithmischen Barriere-Methode (4.7) zu dem Problem

$$\begin{aligned} \min \quad & c^T x - \alpha \textstyle\sum_{i=1}^{n} \ln(x_i) \\ \text{u.d.N.} \quad & Ax = b \end{aligned} \tag{4.14}$$

vereinfachen. Ziel ist es eine Folge von Barriereparametern α zu finden, die gegen das gesuchte Optimum konvergiert. Da das Problem (4.14) nicht länger linear ist, müssen die KKT-Bedingungen angepasst werden. Mit $\lambda_i := \frac{\alpha}{x_i}$ und $\lambda := (\lambda_1^T, \dots, \lambda_n^T)$ folgt

$$\begin{aligned} A^T \mu + \lambda &= c \\ Ax &= b \\ \lambda_i x_i &= \alpha,\ i = 1, \dots, n. \end{aligned} \tag{4.15}$$

Dies kann man, wenn $x > 0$ und $\lambda > 0$ als Störung der KKT-Bedingungen (4.13) ansehen. Besitzt das nichtlineare Gleichungssystem (4.15) für jedes $\alpha > 0$ eine Lösung, so nennt man diese Menge $\{(x(\alpha), \mu(\alpha), \lambda(\alpha)) | \alpha > 0\}$ **Zentraler Pfad**.
Die Frage ist, ob eine solche Lösung für die Zentralen Pfad-Bedingungen existiert. Dazu benötigt

man die *strikt zulässige Menge*

$$\mathcal{F}^O := \{(x, \lambda, \mu) | Ax = b,\ A^T\mu + \lambda = c,\ x > 0,\ \lambda > 0\}. \tag{4.16}$$

Ist diese Menge leer, haben die zentralen Pfadbedingungen (4.13) und damit auch das Barriere-Problem keine Lösung. Das bedeutet das $\mathcal{F}^O \neq \emptyset$ eine notwendige Bedingung dafür ist, dass das Barriere-Problem 4.14 ein Minimum besitzt. Man kann außerdem zeigen, dass diese Bedingung auch hinreichend ist.

Satz 4.7. Die strikt zulässige Menge $\mathcal{F}^O$ sei nichtleer. Dann besitzt das Barriere-Problem (4.14) für jedes $\alpha > 0$ eine Lösung $x(\alpha)$.

Beweis. Siehe Ref. [GK02] S.133.
Es sei $\alpha > 0$ fest gewählt und $(\hat{x}, \hat{\mu}, \hat{\lambda}) \in \mathcal{F}^O$ gegeben. Es gilt $A^T\hat{\mu} + \hat{\lambda} = c$, $A\hat{x} = b$, $\hat{x}, \hat{\lambda} > 0$. Es wird gezeigt, dass die Menge

$$\mathcal{L}_\alpha := \{x \in \mathbb{R}^n | Ax = b,\ x \geq 0,\ B(x; \alpha) \leq B(\hat{x}; \alpha)\}$$

mit der Barrierefunktion

$$B(x; \alpha) := c^T x - \alpha \sum_{i=1}^{n} \log(x_i)$$

kompakt ist. Da sie abgeschlossen ist, muss nur noch die Beschränktheit gezeigt werden. Es sei $\alpha \in L_\alpha$. Dann gilt

$$\begin{aligned} B(x; \alpha) &= c^T x - \alpha \sum_{i=1}^{n} \log(x_i) \\ &= c^T x - \hat{\mu}^T(Ax - b) - \alpha \sum_{i=1}^{n} \log(x_i) \\ &= c^T x - x^T A^T \hat{\mu} + b^T \hat{\mu} - \alpha \sum_{i=1}^{n} \log(x_i) \\ &= c^T x - x^T(c - \hat{\lambda}) + b^T \hat{\mu} - \alpha \sum_{i=1}^{n} \log(x_i) \\ &= x^T \hat{\lambda} + b^T \hat{\mu} - \alpha \sum_{i=1}^{n} \log(x_i). \end{aligned}$$

Damit ist $B(x;\alpha) \leq B(\hat{x};\alpha)$ äquivalent zu $x^T\hat{\lambda} + b^T\hat{\mu} - \alpha\sum_{i=1}^{n}\log(x_i) \leq B(\hat{x};\alpha)$ oder auch

$$\sum_{i=1}^{n}\Big(\hat{\lambda}_i x_i - \alpha\log(x_i)\Big) \leq B(\hat{x};\alpha) - b^T\hat{\mu} =: \kappa,$$

wobei κ eine Konstante ist.
Eine Funktion $x_i \mapsto p(x_i) := \hat{\lambda}_i x_i - \alpha\log(x_i)$ hat die Eigenschaft $p(x_i) \to \infty$ für $x_i \to 0$ oder $x_i \to 0$. Das bedeutet die Menge $\{x \in \mathbb{R}^n | B(x;\alpha) \leq B(\hat{x};\alpha)\}$ ist beschränkt und es folgt $x > 0$. Damit ist auch $\mathcal{L}_\alpha$ beschränkt und damit kompakt. Damit nimmt die stetige Funktion $B(x;\alpha)$ auf der kompakten Menge L_α ihr Minimum an. □

Mit dem nachfolgenden Satz folgt bereits, dass das Zentrale Pfad Problem eine Lösung besitzt.

Satz 4.8. Sei $\alpha > 0$. Dann besitzt das Barriere-Problem (4.14) genau dann eine Lösung $x > 0$, wenn die Zentralen Pfadbedingungen eine Lösung (x, μ, λ) mit $x > 0, \lambda > 0$ besitzen.

Beweis. Die KKT-Bedingungen sind für das konvexe Barriere-Problem sowohl notwendig wie auch hinreichend. Außerdem stimmen sie mit den Bedingungen des Zentralen Pfades (4.15) überein. □

Für die numerische Lösung der Bedingungen des Zentralen Pfades (4.15) wird das Newton-Verfahren verwendet, welches hier nicht näher beschrieben werden soll.

Kapitel 5

Die Ableitungen der Schlüsselrate

5.1 Parametrisierung der Schlüsselrate

Die Parametrisierung der Dichtematrix ist ein wichtiger Punkt zur Optimierung der Schlüsselrate. Wie schon im vorherigen Kapitel beschrieben, wird zur Optimierung der Schlüsselrate ihre erste und zweite Ableitung nach den zu optimierenden Parametern benötigt. Es muss also eine Darstellung gefunden werden, die den festgelegten Anteil der Dichtematrix, aus dem die von Alice und Bob ermittelten Parameter hervorgehen, vom variablen Teil, der Alice und Bob nicht zugänglich ist und über den optimiert wird, trennt. Die Parametrisierung der Dichtematrix ρ_{AB} sollte außerdem so gewählt sein, dass sich daraus leicht die Ableitungen der von-Neumann-Entropie bestimmen lassen.
Eine Möglichkeit einer passenden Parametrisierung ist es, die Bloch-Darstellung auf höher Dimensionale Systeme zu erweitern [EBP]. Das bedeutet für die Parametrisierung einer Dichtematrix

$$\rho = \frac{1}{n}\mathbb{1}_n + \frac{1}{2}\sum_{j=1}^{n^2-1} \lambda_j \hat{\lambda}_j. \tag{5.1}$$

Dabei ist n die Dimension des Hilbertraumes. $\hat{\lambda}_j$ sind die Generatoren der speziellen unitären Gruppe SU(n) mit den Eigenschaften

$$\hat{\lambda}_j^* = \hat{\lambda}_j,\ \mathrm{tr}\left(\hat{\lambda}_j\right) = 0,\ \mathrm{tr}\left(\hat{\lambda}_i\hat{\lambda}_j\right) = 2\delta_{ij},\ i,j = 1,\ldots,n^2-1 \tag{5.2}$$

und den folgenden Kommutator- und Antikommutatorrelationen

$$\left[\hat{\lambda}_i, \hat{\lambda}_j\right] = 2i \sum_{k=1}^{n^2-1} f_{ijk}\hat{\lambda}_k, \ \left\{\hat{\lambda}_i, \hat{\lambda}_j\right\} = \frac{4}{n}\delta_{ij}\mathbb{1}_n + 2\sum_{k=1}^{n^2-1} g_{ijk}\hat{\lambda}_k. \tag{5.3}$$

g_{ijk} und f_{ijk} sind die Strukturkonstanten der SU(n).
Im Fall des BB84 Protokolls präpariert die Quelle ein System aus zwei Qubits. Daher ist es sinnvoll für die Generatoren statt einer Basis aus allgemeinen 4x4 Gell-Mann Matrizen, die Basis als Basis der SU(2)⊗SU(2) zu wählen. Damit die Eigenschaften erfüllt sind, muss zum Tensorprodukt der Pauli-Matrizen jeweils noch ein Faktor $\frac{1}{\sqrt{2}}$ ergänzt werden. Für die Basis der SU(2)⊗SU(2) folgt dann

$$\begin{aligned}
\hat{\lambda}_i &= \frac{1}{\sqrt{2}}\sigma_i \otimes \mathbb{1}_2, \ i = 1,2,3 \\
\hat{\lambda}_i &= \frac{1}{\sqrt{2}}\mathbb{1}_2 \otimes \sigma_{i-3}, \ i = 4,5,6 \\
\hat{\lambda}_i &= \frac{1}{\sqrt{2}}\sigma_1 \otimes \sigma_{i-6}, \ i = 7,8,9 \\
\hat{\lambda}_i &= \frac{1}{\sqrt{2}}\sigma_2 \otimes \sigma_{i-9}, \ i = 10,11,12 \\
\hat{\lambda}_i &= \frac{1}{\sqrt{2}}\sigma_3 \otimes \sigma_{i-6}, \ i = 13,14,15.
\end{aligned} \tag{5.4}$$

Damit kann eine zwei Qubit Zustand folgendermaßen ausgedrückt werden

$$\rho = \frac{1}{2\sqrt{2}}\left(\frac{1}{\sqrt{2}}\mathbb{1}_4 + \sum_{i=1}^{3}\lambda_i\sigma_i \otimes \mathbb{1}_2 + \sum_{i=1}^{3}\lambda_{i+3}\mathbb{1}_2 \otimes \sigma_i + \sum_{i=1}^{3}\sum_{j=1}^{3}\lambda_{j+3i+3}\sigma_i \otimes \sigma_j\right), \tag{5.5}$$

wobei die λ_i so gewählt werden müssen, dass ρ positiv ist. Mit $\sigma_0 := \mathbb{1}_2$,
$F_{ij} := \frac{1}{2}\sigma_i \otimes \sigma_j$ und einer Umdefinition der Koeffizienten λ_i in r_{ij} folgt

$$\rho = \frac{1}{4}\mathbb{1}_4 + \left(\sum_{i=1}^{3} r_{i0}\ F_{i0} + r_{0i}\ F_{0i}\right) + \sum_{i=1}^{3}\sum_{i=j}^{3} r_{ij}\ F_{ij}. \tag{5.6}$$

Bei den Protokollen ist ein Teil der Koeffizienten aus den Messungen bekannt, über die anderen muss optimiert werden. Daraus ergibt sich eine sinnvolle Parametrisierung der Dichtematrix durch die folgende Darstellung

$$\rho_{AB} = \rho_{\text{fix}} + \sum_{i,j} r_{ij}\ F_{ij}, \tag{5.7}$$

mit den Kombinationen i und j für die unbekannten Parameter r_{ij}. Mit dieser Parametrisierung ist es nun möglich die Ableitung der Schlüsselrate zu berechnen.

5.2 Matrix Analysis

Um die Ableitung der von-Neumann-Entropie mit der in Gleichung (5.7) gegebenen Parametrisierung zu bestimmen, müssen zunächst einige mathematische Grundlagen mit Hilfe von Ref. [HJ91] geschaffen werden.

Definition 5.1 (Primäre Matrix Funktion)**.** Es sei $\varphi : \mathbb{R} \to \mathbb{R}$ eine gegebene Funktion und $A \in \mathbb{S}^m$ eine gegebene symmetrische Matrix. Des weiteren sei $A = S\Lambda S^T$ eine Eigenwertzerlegung von A, $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)^T$. Dann ist die Primäre Matrix Funktion Φ, die zu φ korrespondiert

$$\begin{aligned}\Phi : \mathbb{S}^m &\to \mathbb{S}^m \\ A &\mapsto S \begin{pmatrix} \varphi(\lambda_1) & 0 & \ldots & 0 \\ 0 & \varphi(\lambda_2) & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \ldots & 0 & \varphi(\lambda_m) \end{pmatrix} S^T.\end{aligned}$$

Um die Notation zu verkürzen ist die folgende Definition hilfreich:

Definition 5.2. Sei (a, b) ein gegebenes reelles Intervall, $t_1, \ldots, t_m$ reelle Werte und $\varphi : \mathbb{R} \to \mathbb{R}$ eine zweifach stetig differenzierbare Funktion. Dann kann man folgendes definieren:

$$\Delta\varphi(t_i, t_j) = \begin{cases} \frac{\varphi(t_i)-\varphi(t_j)}{t_i-t_j} & \text{, für } i \neq j, \\ \varphi'(t_i) & \text{, für } i = j, \end{cases}$$

$$\Delta^2\varphi(t_i, t_j, t_k) = \begin{cases} \frac{\Delta\varphi(t_i,t_k)-\Delta\varphi(t_j,t_k)}{t_i-t_j} & \text{, für } i \neq j, \\ \frac{\Delta\varphi(t_i,t_j)-\Delta\varphi(t_k,t_j)}{t_i-t_j} & \text{, für } i = j \neq k, \\ \varphi''(t_i) & \text{, für } i = j = k. \end{cases}$$

Außerdem wird die Frobenius Kovarianz Matrix genutzt.

Definition 5.3 (Frobenius Kovarianz Matrix)**.** Sei $\mathcal{A} : D \subset \mathbb{R}^n \to \mathbb{S}^m$ eine gegebene Abbildung. Seien $\lambda_1(x), \ldots, \lambda_{\mu(x)}(x)$ die in aufsteigender Reihenfolge geordneten Eigenwerte von $\mathcal{A}(x)$. Des weiteren sei $\mathcal{A}(x) = S(x)\Lambda(x)S(x)^T$ eine Eigenwertzerlegung von $\mathcal{A}(x)$ mit $\Lambda(x) =$

$\text{diag}(\lambda_1(x), \ldots, \lambda_1(x), \ldots, \lambda_{\mu(x)}(x), \ldots, \lambda_{\mu(x)}(x))$, wobei jeder Eigenwert in seiner gegebenen Vielfachheit auftaucht. Dann ist die Frobenius Kovarianz Matrix definiert durch

$$P_{\mathcal{A},i} = S(x)\text{diag}(0, \ldots, 0, 1, \ldots, 1, 0, \ldots, 0)S(x)^T \quad , \; i = 1, \ldots, \mu(x).$$

Die Frobenius Kovarianz Matrix hat wichtige Eigenschaften, die in dem folgenden Lemma beschrieben werden.

Lemma 5.4. *Sei $\mathcal{A} \in \mathbb{S}^m$ gegeben. Dann gilt:*

1. $P_{\mathcal{A},i}(x) + \cdots + P_{\mathcal{A},\mu(x)}(x) = \mathbb{1}_m$;
2. $P_{\mathcal{A},i}(x)P_{\mathcal{A},j}(x) = 0$ *für alle* $i \neq j$, $i, j = 1, \ldots, \mu(x)$;
3. $P_{\mathcal{A},i}(x)^k$ *für alle* $k \geq 1$, $i = 1, \ldots, \mu(x)$.

Es gilt für die partielle Ableitung einer Matrixfunktion folgender wichtiger Satz.

Satz 5.5. Es sei $(a, b) \subseteq \mathbb{R}$, $m \in \mathbb{N}$ und $A : D \subset \mathbb{R}^n \to \mathbb{S}^m$ eine zweifach stetig differenzierbare Abbildung. $\lambda_1(x), \ldots, \lambda_{\mu(x)}(x)$ bezeichnet die $\mu(x)$ in aufsteigender Reihenfolge geordneten Eigenwerte von $A(x)$ und es gilt $\lambda_1(x) \geq a$ und $\lambda_2(x) \leq b$ für alle $x \in D$. Des weiteren sei $\varphi : (a, b) \to \mathbb{R}$ eine zweifach stetig differenzierbare Funktion und Φ die korrespondierende Matrixfunktion. Dann ist $\Phi \circ A$ zweifach differenzierbar für alle $x \in D$ und folgende Formeln gelten:

$$\frac{\partial}{\partial x_i}\Phi(A(x)) = \sum_{k,l=1}^{\mu(x)} \Delta\varphi(\lambda_k(x), \lambda_l(x))P_k(x)A_i'(x)P_l(x), \tag{5.8}$$

$$\begin{aligned}\frac{\partial^2}{\partial x_i \partial x_j}\Phi(A(x)) = &\sum_{k,l,s=1}^{\mu(x)} \Delta^2\varphi(\lambda_k(x), \lambda_l(x), \lambda_s(x)) \cdot (M + M^T), \\ &+ \sum_{k,l=1}^{\mu(x)} \Delta\varphi(\lambda_k(x), \lambda_l(x))P_k(x)A_{i,j}''(x)P_l(x)\end{aligned} \tag{5.9}$$

mit der Matrix M als $M = P_k(x)\mathcal{A}_i'P_l(x)\mathcal{A}_j'(x)P_s(x)$.

Beweis. Siehe [HJ91]. □

Auf dieser Grundlage ist es nun möglich, die erste und zweite Ableitung der Schlüsselrate zu bestimmen.

5.3 Erste und zweite Ableitung

Um die Ableitung der Schlüsselrate zu berechnen, muss die Ableitung der von-Neumann-Entropie bestimmt werden. Die klassische Information, die sich Alice und Bob teilen, muss auch bei Veränderung des Zustands $\rho_{AB} \in \Gamma$ konstant bleiben. Es wird also bei der Optimierung nur über die r_{ij} abgeleitet, die Alice und Bob nicht zugänglich sind. Dann ergibt sich für die Ableitung aus Gleichung (2.12)

$$\frac{\partial}{\partial r_{ij}}\, r = \frac{\partial}{\partial r_{ij}}\chi(X:E) = \frac{\partial}{\partial r_{ij}}S(\rho_{AB}(\vec{r})) - \sum_x p(x)\frac{\partial}{\partial r_{ij}}S(\rho_{AB}(\vec{r})), \tag{5.10}$$

Somit genügt es die erste und zweite Ableitung der von-Neumann-Entropie $\frac{\partial}{\partial r_{ij}}S(\rho(\vec{r}))$ zu bestimmen. Es gilt

$$\begin{aligned}\frac{\partial}{\partial r_{ij}}\; - \operatorname{tr}\left(\rho(\vec{r})\log_2\left[\rho(\vec{r})\right]\right) =&\; \operatorname{tr}\left(-\frac{\partial\rho(\vec{r})}{\partial r_{ij}}\log_2\left[\rho(\vec{r})\right] - \rho(\vec{r})\frac{\partial}{\partial r_{ij}}\log_2\left[\rho(\vec{r})\right]\right)\\ =&\; -\operatorname{tr}\left(\frac{\partial\rho(\vec{r})}{\partial r_{ij}}\log_2\left[\rho(\vec{r})\right]\right) - \operatorname{tr}\left(\rho(\vec{r})\frac{\partial}{\partial r_{ij}}\log_2\left[\rho(\vec{r})\right]\right).\end{aligned} \tag{5.11}$$

Untersucht wird zunächst der rechte Term dieser Gleichung. Die Primäre Matrix Funktion ist in diesem Fall $\varphi(x) = \log_2(x)$. Es gilt

$$\begin{aligned}\operatorname{tr}\left(\rho(\vec{r})\frac{\partial}{\partial r_{ij}}\log_2\left[\rho(\vec{r})\right]\right) &= \operatorname{tr}\left(\rho(\vec{r})\sum_{k,l}\varphi(\lambda_k,\lambda_l)P_kF_{ij}P_l\right)\\ &= \sum_{k,l}\Delta\varphi(\lambda_k,\lambda_l)\,\operatorname{tr}\left(\rho(\vec{r})P_kF_{ij}P_l\right)\\ &= \sum_{k,l}\Delta\varphi(\lambda_k,\lambda_l)\operatorname{tr}\left(\sum_m\lambda_m\left|\Psi_m\right\rangle\left\langle\Psi_m|\Psi_k\right\rangle\left\langle\Psi_k\right|F_{ij}\left|\Psi_l\right\rangle\!\left\langle\Psi_l\right|\right)\\ &= \sum_m\Delta\varphi(\lambda_m,\lambda_m)\lambda_m\left\langle\Psi_m\right|F_{ij}\left|\Psi_m\right\rangle\\ &= \sum_m\frac{1}{\ln 2}\frac{1}{\lambda_m}\lambda_m\left\langle\Psi_m\right|F_{ij}\left|\Psi_m\right\rangle\\ &= \frac{1}{\ln 2}\underbrace{\operatorname{tr}\left(F_{ij}\right)}_{=0}.\end{aligned} \tag{5.12}$$

Die Spur der Matrizen, die in der angegebenen Parametrisierung den Zustand generieren ist Null, da es sich um Pauli-Matrizen, Gell-Mann-Matrizen oder ihre Tensorprodukte handelt. Der

linke Term der Ableitung ergibt

$$\begin{aligned}
\mathrm{tr}\left(\frac{\partial\rho(\vec{r})}{\partial r_{ij}}\log_2\left[\rho(\vec{r})\right]\right) &= \mathrm{tr}\left(F_{ij}\log_2\left[\rho(\vec{x})\right]\right) \\
&= \sum_k \langle\Psi_k| F_{ij}\log_2[\rho(\vec{r})]\,|\Psi_k\rangle \\
&= \sum_k \langle\Psi_k| F_{ij}\sum_l \log_2[\lambda_l]\,|\Psi_l\rangle\langle\Psi_l|\Psi_k\rangle \\
&= \sum_k \log_2[\lambda_k]\langle\Psi_k| F_{ij}\,|\Psi_k\rangle \\
&= \sum_k \frac{\ln[\lambda_k]}{\ln[2]}\langle\Psi_k| F_{ij}\,|\Psi_k\rangle\,. \qquad (5.13)
\end{aligned}$$

Damit ergibt sich für die erste Ableitung

$$\frac{\partial}{\partial r_{ij}}\ \left[-\mathrm{tr}\left(\rho(\vec{r})\log_2\left[\rho(\vec{r})\right]\right)\right] = -\sum_k \log_2[\lambda_k]\langle\Psi_k| F_{ij}\,|\Psi_k\rangle\,. \qquad (5.14)$$

Nach dieser einfachen ersten Ableitung folgt die zweite Ableitung durch

$$\begin{aligned}
\frac{1}{\partial r_{kl}}\left(\frac{\partial}{\partial r_{ij}}\ \left[-\mathrm{tr}\left(\rho(\vec{r})\log_2\left[\rho(\vec{r})\right]\right)\right]\right) &= -\frac{1}{\partial r_{kl}}\mathrm{tr}\left(F_{ij}\log_2\left[\rho(\vec{x})\right]\right) \\
&= -\mathrm{tr}\left(F_{ij}\frac{1}{\partial r_{kl}}\log_2\left[\rho(\vec{x})\right]\right) \\
&= -\mathrm{tr}\left(F_{ij}\sum_{n,o}\Delta\varphi(\lambda_n,\lambda_o)P_nF_{kl}P_o\right) \\
&= -\sum_{n,o}\Delta\varphi(\lambda_n,\lambda_o)\mathrm{tr}\left(F_{ij}P_nF_{kl}P_o\right) \\
&= -\sum_{n,o}\Delta\varphi(\lambda_n,\lambda_o)\langle\Psi_o| F_{ij}\,|\Psi_n\rangle\langle\Psi_n| P_nF_{kl}P_o\,|\Psi_o\rangle\,. \qquad (5.15)
\end{aligned}$$

Mit diesen beiden Ableitungen kann nun sowohl der Gradient, wie auch die Hesse-Matrix der Schlüsselrate für die konvexe Optimierung bestimmt werden. Damit sind alle theoretischen Grundlagen gelegt und es kann nun damit begonnen werden, die minimale Schlüsselrate verschiedener Protokolle zu ermitteln.

Kapitel 6

Protokolle

In diesem Kapitel soll die minimale Schlüsselrate verschiedener Protokolle bestimmt werden. Zunächst werden einige bekannte Ergebnisse mit dem in den vorherigen Kapiteln beschriebenen Methoden reproduziert. Danach werden mit dem BB84 mit asymmetrischen Fehlerraten und dem 2-state mit und ohne Verlust einige Protokolle untersucht, für die noch keine analytischen Lösungen existieren und bei denen eine Verbesserung der unteren Schranke der erreichbaren Schlüsselrate zu erwarten ist.

6.1 6-state Protokoll

Das Ziel des Kapitels ist es, die Schlüsselrate des 6-state Protokolls zu bestimmen. Dazu wird zunächst das Prepare&Measure-Schema des 6-state Protokolls in ein verschränkungsbasiertes Schema gebracht. Es folgt die Bestimmung des Zustandes von Alice und Bob, sowie die Bestimmung der Schlüsselrate mit Hilfe des Devetak-Winter Security Bound. Das 6-state Protokoll ist eine Erweiterung des in Kapitel 1.1 beschriebenen BB84-Protokolls. Hier präpariert Alice die Signalzustände nicht nur in der z- und x-Basis, sondern auch in der y-Basis. Das Besondere ist hierbei, dass damit die Dichtematrix ρ_{AB} zwischen Alice und Bob komplett bestimmt ist. Es ist somit nicht notwendig, die Schlüsselrate zu optimieren.

6.1.1 Verschränkungsbasiertes 6-state Protokoll

Um eine verschränkungsbasierte Version des 6-State Protokolls zu erhalten, kann man folgendermaßen vorgehen. Alice präpariert durch eine Messung in der Standartbasis

$\{|0\rangle, |1\rangle, |2\rangle, |3\rangle, |4\rangle, |5\rangle\}$ den Zustand, den Bob erhalten soll.
Bob erhält die Zustände $\{|0\rangle, |1\rangle, |+\rangle, |-\rangle, |+i\rangle, |-i\rangle\}$. Dann ist der Zustand der Quelle

$$|\Psi_S\rangle = \frac{1}{\sqrt{6}} \left(|0\rangle|0\rangle + |1\rangle|1\rangle + |2\rangle|+\rangle + |3\rangle|-\rangle + |4\rangle|+i\rangle + |5\rangle|-i\rangle\right). \tag{6.1}$$

Im nächsten Schritt gilt es, die Schmidt-Zerlegung zu finden

$$\begin{aligned} |\Psi_S\rangle &= \frac{1}{\sqrt{6}} \left(\left(|0\rangle + \frac{|2\rangle + |3\rangle + |4\rangle + |5\rangle}{\sqrt{2}}\right)|0\rangle + \left(|1\rangle + \frac{|2\rangle - |3\rangle + i\,|4\rangle - i\,|5\rangle}{\sqrt{2}}\right)|1\rangle\right) \\ &= \frac{1}{\sqrt{2}} \left(|\tilde{0}\rangle|0\rangle + |\tilde{1}\rangle|1\rangle\right) = |\Phi^+\rangle, \end{aligned} \tag{6.2}$$

mit $|\tilde{0}\rangle := \frac{1}{\sqrt{3}}\left(|0\rangle + \frac{|2\rangle+|3\rangle+|4\rangle+|5\rangle}{\sqrt{2}}\right), |\tilde{1}\rangle := \frac{1}{\sqrt{3}}\left(|1\rangle + \frac{|2\rangle-|3\rangle+i|4\rangle-i|5\rangle}{\sqrt{2}}\right)$. Die Zustände $|\tilde{0}\rangle$, $|\tilde{1}\rangle$ sind orthogonal. Damit hat man die sechs Dimensionen von Alice auf zwei Dimensionen reduziert. Aus Alices projektiver Messung P_x am alten P&M- Zustand wird nun, wie im zweiten Kapitel beschrieben, ein POVM $\mathcal{A}$, für das gilt $\mathcal{A} = \{\frac{1}{3}|\tilde{0}\rangle\langle\tilde{0}|, \frac{1}{3}|\tilde{1}\rangle\langle\tilde{1}|, \frac{1}{3}|\tilde{+}\rangle\langle\tilde{+}|, \frac{1}{3}|\tilde{-}\rangle\langle\tilde{-}|, \}$. Mit Hilfe dieses Schemas kann man die Schlüsselrate des Protokolls bestimmen. Dazu muss allerdings zunächst der Zustand, den Alice und Bob nach Eves Attacke teilen, ermittelt werden.

6.1.2 Bestimmung von ρ_{AB}

Um den Schlüssel zu generieren, wollen Alice und Bob viele Kopien des verschränkten Zustands $|\Phi^+\rangle$ teilen. Allerdings wechselwirkt Eve mit dem an Bob gesandten Zustand und dadurch verändert sich der Zustand ρ_{AB}. Ein übliches Model für ein Rauschen, das heißt für Fehler, die im Kanal erzeugt werden, ist der depolarisierende Kanal

$$\rho_{AB} \mapsto p\rho_{AB} + \frac{1-p}{d}\rho_A \otimes \mathbb{1}_B \tag{6.3}$$

mit der Wahrscheinlichkeit p und der Dimension d. Messen Alice und Bob in den selben Basen und erhalten trotzdem verschiedene Ergebnisse, so bezeichnet man dies als so genannten *Qubitfehler* (QBER). Im 6-state Protokoll ist der Zusammenhang zur Wahrscheinlichkeit p des depolarisierenden Kanals durch $p = (1 - 2Q)$ gegeben. Damit folgt für den Zustand von Alice

und Bob nach Durchlaufen des Kanals

$$\rho_{AB} = \begin{pmatrix} \frac{1-Q}{2} & 0 & 0 & \frac{1-2Q}{2} \\ 0 & \frac{Q}{2} & 0 & 0 \\ 0 & 0 & \frac{Q}{2} & 0 \\ \frac{1-2Q}{2} & 0 & 0 & \frac{1-Q}{2} \end{pmatrix}. \tag{6.4}$$

Man kann diesen Ausdruck in die Bell-Basis bringen. Die Transformationsmatrix ist die Matrix der Bellzustandsvektoren in der Standartbasis

$$T_{\text{Bell}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & -1 \end{pmatrix}. \tag{6.5}$$

Mit der Transformation $\tilde{\rho}_{AB} = T_{\text{Bell}}^{-1} \rho_{AB} T_{\text{Bell}}$ folgt dann in der neuen Basis

$$\tilde{\rho}_{AB} = \begin{pmatrix} 1 - \frac{3Q}{2} & 0 & 0 & 0 \\ 0 & \frac{Q}{2} & 0 & 0 \\ 0 & 0 & \frac{Q}{2} & 0 \\ 0 & 0 & 0 & \frac{Q}{2} \end{pmatrix}. \tag{6.6}$$

6.1.3 Die Schlüsselrate

Mit diesem Zustand ist es nun leicht die Schlüsselrate des 6-state Protokolls zu berechnen. Es gilt, wie in Gleichung (2.12) angegeben,

$$r(\rho_{AB}) = H(X) + H(Y) - H(X,Y) - S(\rho_{AB}) + \sum_x p(x) S(\rho_B^x). \tag{6.7}$$

Im Fall des 6-State Protokolls lassen sich all diese Größen in Abhängigkeit von Q bestimmen. Für die klassische gemeinsame Information von Alice und Bob gilt

$$I(X:Y) = 2 + \frac{Q}{2} \log_2 \left[\frac{Q}{2}\right] + \frac{1-Q}{2} \log_2 \left[\frac{1-Q}{2}\right] = 1 - h(Q), \tag{6.8}$$

wobei $h(x) = -(1-x)\log_2(1-x) - x\log(x)$ die binäre Shannonentropie ist. Für die Quanteninformation folgt, da die Eigenwerte der auf den Ausgang von Alice konditionierten Dichtematrix

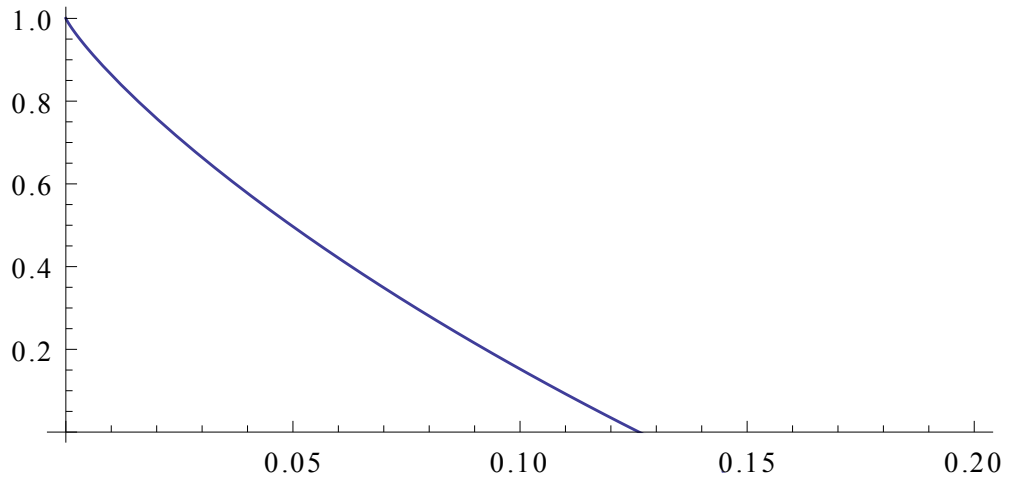

Abbildung 6.1: Schlüsselrate des 6-state Protokolls abhängig von der gemessenen QBER.

in allen Messbasen gleich sind, mit

$$\rho_B^0 = \frac{1}{p(0)} \mathrm{tr}_A \left(|0\rangle\langle 0| \otimes \mathbb{1} \rho_{AB} \right) = \begin{pmatrix} 1-Q & 0 \\ 0 & Q \end{pmatrix}$$

und analogem Ausdruck bei gemessenem Bitwert 1 bei Alice

$$\chi(X:E) = S(\rho_{AB}) - h(Q) = -(1-\frac{3}{2}Q)\log_2\left[1-\frac{3}{2}Q\right] - 3\frac{Q}{2}\log_2\left[\frac{Q}{2}\right] - h(Q). \qquad (6.9)$$

Das Ergebnis lässt sich in Abhängigkeit der QBER Q darstellen und ist in Abb. 1 zu sehen. Die Fehlerrate, ab der kein sicherer Schlüssel mehr generiert werden kann, liegt bei einer QBER von $Q = 0.126$.

6.2 Die Schlüsselrate des BB84 Protokolls

Nach dem einfachen, optimierungsfreien Fall des 6-state Protokolls soll nun die Schlüsselrate des BB84 Protokolls bestimmt werden. Im Gegensatz zum 6-state Protokoll ist in diesem Fall der Zustand ρ_{AB} durch die Messergebnisse nicht eindeutig bestimmt. Daher muss für die freien Parameter eine Optimierung durchgeführt werden, um die untere Grenze der Schlüsselrate zu bestimmen. Das BB84 Protokoll ist in Kapitel 1.1 schon ausführlich besprochen worden. Daher werden hier nur die entscheidenden Schritte zur Optimierung der Schlüsselrate betrachtet.

6.2.1 Verschränkungsbasiertes BB84 Protokoll

Wie im Fall des 6-state Protokolls soll auch hier das Prepare & Measure Schema des Protokolls in ein verschränkungsbasiertes Schema umgewandelt werden. Dazu wird analog zum 6-state Protokoll vorgegangen. Im P&M Schema des BB84 Protokolls präpariert Alice jeweils zwei Zustände in der x- und in der z-Basis mit identischer Wahrscheinlichkeit. Damit ist der präparierte Zustand gegeben durch

$$|\Psi_S\rangle = \frac{1}{\sqrt{4}} \left(|0\rangle\,|0\rangle + |1\rangle\,|1\rangle + |2\rangle\,|+\rangle + |3\rangle\,|-\rangle\right). \tag{6.10}$$

Auch diesen Zustand kann man durch einige einfache Umformungen auf Dimension zwei bringen

$$\begin{aligned} |\Psi_S\rangle &= \frac{1}{\sqrt{4}} \left[\left(|0\rangle + \frac{|2\rangle + |3\rangle}{\sqrt{2}}\right)|0\rangle + \left(|1\rangle + \frac{|2\rangle - |3\rangle}{\sqrt{2}}\right)|1\rangle\right] \\ &= \frac{1}{\sqrt{2}} \left(\left|\tilde{0}\right\rangle|0\rangle + \left|\tilde{1}\right\rangle|1\rangle\right) = \left|\Phi^+\right\rangle, \end{aligned} \tag{6.11}$$

wobei $\left|\tilde{0}\right\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + \frac{|2\rangle+|3\rangle}{\sqrt{2}}\right)$ und $\left|\tilde{1}\right\rangle = \frac{1}{\sqrt{2}}\left(|1\rangle + \frac{|2\rangle-|3\rangle}{\sqrt{2}}\right)$ die neuen Basisvektoren von Alices Qubit in der Schmidtzerlegung sind. Damit ist die allgemeine Messung, die Alice an diesem Zustand durchführt, gegeben durch
$\mathcal{A} = \{\frac{1}{2}\left|\tilde{0}\right\rangle\!\left\langle\tilde{0}\right|, \frac{1}{2}\left|\tilde{1}\right\rangle\!\left\langle\tilde{1}\right|, \frac{1}{2}\left|\tilde{+}\right\rangle\!\left\langle\tilde{+}\right|, \frac{1}{2}\left|\tilde{-}\right\rangle\!\left\langle\tilde{-}\right|\}$. Im nächsten Schritt muss der Zustand ρ_{AB} nach Eves Attacke so weit wie möglich bestimmt werden.

6.2.2 Bestimmung von ρ_{AB}

Die Bestimmung des Zustandes ρ_{AB} ist beim BB84 Protokoll nicht mehr zur Gänze möglich. Allerdings kann ein Großteil der Einträge bestimmt werden, um die Zahl der freien Parameter für die Optimierung gering zu halten. Wie schon beim 6-state Protokoll wird auch hier für die gemessenen Wahrscheinlichkeiten als Model der depolarisierende Kanal gewählt. Hier wird zum ersten Mal die Parametrisierung aus Gleichung (5.7) eingesetzt. σ_0 ist die Einheitsmatrix $\mathbb{1}_2$

und σ_x, σ_y und σ_z die Paulimatrizen. Die Tensorprodukte der Paulimatrizen ergeben

$$
\begin{aligned}
F_{0x} &= \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} & F_{0y} &= \begin{pmatrix} 0 & -i & 0 & 0 \\ i & 0 & 0 & 0 \\ 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \end{pmatrix} \\
F_{0z} &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} & F_{x0} &= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \\
F_{xx} &= \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} & F_{xy} &= \begin{pmatrix} 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ i & 0 & 0 & 0 \end{pmatrix} \\
F_{xz} &= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix} & F_{y0} &= \begin{pmatrix} 0 & 0 & -i & 0 \\ 0 & 0 & 0 & -i \\ i & 0 & 0 & 0 \\ 0 & i & 0 & 0 \end{pmatrix} \\
F_{yx} &= \begin{pmatrix} 0 & 0 & 0 & -i \\ 0 & 0 & -i & 0 \\ 0 & i & 0 & 0 \\ i & 0 & 0 & 0 \end{pmatrix} & F_{yy} &= \begin{pmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix} \\
F_{yz} &= \begin{pmatrix} 0 & 0 & -i & 0 \\ 0 & 0 & 0 & i \\ i & 0 & 0 & 0 \\ 0 & -i & 0 & 0 \end{pmatrix} & F_{z0} &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
F_{zx} &= \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \end{pmatrix} & F_{zy} &= \begin{pmatrix} 0 & -i & 0 & 0 \\ i & 0 & 0 & 0 \\ 0 & 0 & 0 & i \\ 0 & 0 & -i & 0 \end{pmatrix} \\
F_{zx} &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\end{aligned}
\tag{6.12}
$$

Da Alice und Bob sowohl in der x- und in der z-Basis messen, erhält man ein einfaches Schema der bekannten und unbekannten Parameter

$$\begin{array}{c|cccc} r_{ij} & \mathbb{1} & \sigma_x & \sigma_y & \sigma_z \\ \hline \mathbb{1} & 1 & 0 & ? & 0 \\ \sigma_x & 0 & 1-2Q & ? & 0 \\ \sigma_y & 0 & ? & ? & ? \\ \sigma_z & 0 & 0 & ? & 1-2Q \end{array} . \tag{6.13}$$

Damit kann der Zustand ρ_{AB} mit den Tensorprodukten der Paulimatrizen $F_{ij} = \frac{1}{2}\sigma_i \otimes \sigma_j$ geschrieben werden als

$$\begin{aligned} \rho_{AB} = & \begin{pmatrix} \frac{1-Q}{2} & 0 & 0 & \frac{1-2Q}{2} \\ 0 & \frac{Q}{2} & 0 & 0 \\ 0 & 0 & \frac{Q}{2} & 0 \\ \frac{1-2Q}{2} & 0 & 0 & \frac{1-Q}{2} \end{pmatrix} \\ & + \frac{1}{2}\left(r_{0y}F_{0y} + r_{xy}F_{xy} + r_{yx}F_{yx} + r_{yy}F_{yy} + r_{yz}F_{yz} + r_{zy}F_{zy}\right). \end{aligned} \tag{6.14}$$

Der Zustand ρ_{AB} ist so weit wie möglich bestimmt. Damit ist der nächste Schritt die Optimierung über die unbekannten Parameter von ρ_{AB}.

6.2.3 Die Optimierung der Schlüsselrate

Die Bestimmung der Schlüsselrate ist ein nichtlineares konvexes Optimierungsproblem und stellt sich wie folgt dar

$$\begin{array}{ll} \min & r(\rho_{AB}(x)) \\ \text{u.d.Nb.} & \rho_{AB}(x) \geq 0. \end{array} \tag{6.15}$$

Die Grundlage zur Lösung dieses Problems bietet des Programmpaket *CVXOPT* für Python [MAV14]. Da die Zielfunktion nichtlinear ist, kommt das Programm *solvers.cp* zum Einsatz. Das Programm benötigt zur Optimierung einen zulässigen Startpunkt und die Berechnung der ersten und zweiten Ableitung der Schlüsselrate, deren Ergebnisse in der Hauptfunktion des Programms an den Solver übergeben werden.

```
# Hauptfunktion fuer den Solver

def CVXOpt(x = None, z = None):
          # Startpunkt der Optimierung festlegen
     if x is None:
          return 0, x_start

     # Bestimmung der Dichtematrix
     rho = define_rho(x)

     # Ist die Dichtematrix postiv-semidefinit?
     ew = Eigenwerte(copy.copy(rho),0)
     for i in range(rho.size[0]):
          if ew[i] <= 0:
               return None

     # Bestimmung der Holevo-Groesse
     Chi = S(copy.copy(rho)) - 0.25 * ( S(cond(copy.copy(rho),0)) + S(cond(
   copy.copy(rho),1)) + S(cond(copy.copy(rho),2)) + S(cond(copy.copy(rho),3)
   ))

     # Berechnung der Schluesselrate
     r = (I - 0.5 * Chi)

     # Erste Ableitung der Schluesselrate
     Dr = - 0.5 * DChi(rho)
     if z is None:
          return r, Dr

     # Zweite Ableitung der Schluesselrate
     H = - 0.5 * z[0]  * D2Chi(rho)
     return r, Dr, H
```

Quellcodeausschnitt 6.1: Hauptfunktion im Programm zur Optimierung der Schlüsselrate des BB84

Als Startpunkt wird der Punkt festgelegt, für den der Zustand

$$\rho_{AB} = \frac{1}{4}\begin{pmatrix} 2-p & 0 & 0 & (1-p)-r_{yy} \\ 0 & p & (1-p)+r_{yy} & 0 \\ 0 & (1-p)+r_{yy} & p & 0 \\ (1-p)-r_{yy} & 0 & 0 & 2-p \end{pmatrix} + \frac{i}{4}\begin{pmatrix} 0 & -r_{0y}-r_{zy} & -r_{yz} & -r_{xy}-r_{yx} \\ r_{0y}+r_{zy} & 0 & r_{xy}-r_{yx} & r_{yz} \\ r_{yz} & -r_{xy}+r_{yx} & 0 & -r_{0y}+r_{zy} \\ r_{xy}+r_{yx} & -r_{yz} & r_{0y}-r_{zy} & 0 \end{pmatrix}$$

dem Zustand des Depolarisierenden Kanals

$$\rho_{AB} = \frac{1}{4}\begin{pmatrix} 2-p & 0 & 0 & 2(1-p) \\ 0 & p & 0 & 0 \\ 0 & 0 & p & 0 \\ 2(1-p) & 0 & 0 & 2-p \end{pmatrix}$$

entspricht. Dabei ist $p = 2Q$. Das bedeutet, dass für den Parameter $r_{yy} = -(1-p)$ gilt. Für alle anderen Parameter gilt $r_{ij} = 0$.

```
# Startpunkt
print 'Startpunkt: '
x_start = matrix(0.0,(len(Pauli),1))
x_start[0] = -(1-p)
print x_start.T
```

Quellcodeausschnitt 6.2: Festlegung des Startpunktes

Das Programm berechnet jeweils zu den Parametern r_{ij} die Dichtematrix ρ_{AB}. Dabei ist zu beachten, dass die Dichtematrix eine komplexe Matrix ist und zur Verwendung innerhalb des Programms in eine reelle Matrix verwandelt werden muss. Dazu wird einer komplexen $n \times n$ Matrix $Z = X + iY$ eine reelle $2n \times 2n$ Matrix

$$Z = \begin{pmatrix} X & Y \\ -Y & X \end{pmatrix} \tag{6.16}$$

zugeordnet.

```
# Matrizen der Paulidarstellung F fuer die freien Parameter x
```

```
Pauli = []
F = []

Pauli.append(matrix([[0, 0, 0, -1], [0, 0, 1, 0], [0, 1, 0, 0], [-1, 0, 0,
    0]]))
Pauli.append(matrix([[0, +1j, 0, 0],[-1j, 0, 0, 0], [0, 0, 0, +1j], [0, 0,
    -1j, 0]]))
Pauli.append(matrix([[0, 0, 0, 1j],[0, 0, -1j, 0], [0, 1j, 0, 0],[-1j, 0, 0,
     0]]))
Pauli.append(matrix([[0, 0, 0, 1j],[0, 0, 1j, 0], [0, -1j, 0, 0],[-1j, 0, 0,
     0]]))
Pauli.append(matrix([[0, 0, +1j, 0],[0, 0, 0, -1j],[-1j, 0, 0, 0],[0, +1j,
    0, 0]]))
Pauli.append(matrix([[0, 1j, 0, 0],[-1j, 0, 0, 0], [0, 0, 0, -1j], [0, 0, 1j
    , 0]]))
Pauli.append(matrix([[0, 0, 1j, 0],[0, 0, 0, 1j], [-1j, 0, 0, 0], [0, -1j,
    0, 0]]))

for i in range(len(Pauli)):
     mat = matrix(0.0, (2*Pauli[0].size[0],2*Pauli[0].size[1]))
     for j in range(Pauli[0].size[0]):
          for k in range(Pauli[0].size[1]):
               mat[j,k] = Pauli[i][j,k].real
               mat[j+Pauli[0].size[0],k+Pauli[0].size[0]] = Pauli[i][j,k].
    real
               mat[j+Pauli[0].size[0],k] = Pauli[i][j,k].imag
               mat[j,k+Pauli[0].size[0]] = -Pauli[i][j,k].imag
     F.append(0.25*mat)
```

Quellcodeausschnitt 6.3: Paulimatrizen erweitert auf reelle $2n \times 2n$ Matrizen

```
     # Fixer Teil der Dichtematrix
     rho_fix = 0.25*matrix([[2-p, 0, 0, (1-p), 0, 0, 0, 0],[0, p, (1-p), 0,
    0, 0, 0, 0], [0, (1-p), p, 0, 0, 0, 0, 0], [(1-p), 0, 0, 2-p, 0, 0, 0,
    0], [0, 0, 0, 0, 2-p, 0, 0, (1-p)],[0, 0, 0, 0, 0, p, (1-p), 0], [0, 0,
    0, 0, 0, (1-p), p, 0], [0, 0, 0, 0, (1-p), 0, 0, 2-p]])
```

Quellcodeausschnitt 6.4: Fester Anteil der Dichtematrix

```
# Definition der Dichtematrix

def define_rho(x):
```

```
    param = 0.0
    for i in range(len(F)):
        param += x[i]*F[i]
    AB =  rho_fix + (param)
    return AB
```

Quellcodeausschnitt 6.5: Bestimmung der Dichtematrix

Die Prüfung der Eigenwerte der Dichtematrix in der Hauptfunktion ist notwendig, da durch negative Eigenwerte, die bei den verschiedenen Iterationen des Solvers entstehen können, das Programm mit Fehler abbrechen würde. Die Berechnung der Schlüsselrate erfolgt dann durch getrennte Berechnung der klassischen Information und der Holevogröße. Der Wert für die klassische Information ändert sich während der Optimierung eines einzelnen Punktes nicht und kann daher außerhalb der Hauptfunktion berechnet werden.

```
    # Bestimmung der klassischen Information

    I = 1 - h(qerror) - h(1-qerror)

    Chi_start = S(copy.copy(rho_start)) - 0.25 * ( S(cond(copy.copy(
  rho_start),0)) + S(cond(copy.copy(rho_start),1)) + S(cond(copy.copy(
  rho_start),2)) + S(cond(copy.copy(rho_start),3)))
    Chi_end = S(copy.copy(rho_end)) - 0.25 * ( S(cond(copy.copy(rho_end),0)
  ) + S(cond(copy.copy(rho_end),1)) + S(cond(copy.copy(rho_end),2)) + S(
  cond(copy.copy(rho_end),3)))
    print 'Schluesselrate r_Start: '
    print I - 0.5 * Chi_start
    print 'Schluesselrate r_Ende: '
    r_ende = I - 0.5*Chi_end
    print r_ende

    if r_ende <= 0:
                break

    xdata.append(qerror)
    ydata.append(r_ende)

plt.title('Secret Key Rate of the BB84')
plt.axis([0.0, 0.15, 0., 1.])
plt.plot(xdata, ydata, 'ro')
plt.xlabel('QBER q')
plt.ylabel('Key Rate r')
```

```
plt.grid(True)
plt.show()
```

Quellcodeausschnitt 6.6: Berechnung der klassischen Information außerhalb der Hauptfunktion

Die Holevogröße wird innerhalb der Hauptfunktion berechnet. Dazu wird zunächst die Konditionierung der Dichtematrix auf das jeweilige Ergebnis von Alice durchgeführt und daraus dann die von-Neumann-Entropie bestimmt.

```
# Berechnung der auf das Resultat a knoditionierten Dichtematrizen

def cond(rho,a):
     AB = matrix(complex(0,0),(rho.size[0]/2,rho.size[1]/2))
     for i in range(AB.size[0]):
          for l in range(AB.size[1]):
               AB[i,l] =complex(rho[i,l],rho[i+rho.size[0]/2,l])
     B = matrix(complex(0,0),(rho.size[0]/4,rho.size[1]/4))

     # Die verschiedenen Konditionierungen

         if a == 0:
          trAB = spdiag([1.0,1.0,0,0]) * AB * 2
     if a == 1:
          trAB = spdiag([0,0,1.0,1.0]) * AB * 2
     if a == 2:
          trAB = matrix
   ([[1.0,0,1.0,0],[0,1.0,0,1.0],[1.0,0,1.0,0],[0,1.0,0,1.0]]) * AB
     if a == 3:
          trAB = matrix
   ([[1.0,0,-1.0,0],[0,1.0,0,-1.0],[-1.0,0,1.0,0],[0,-1.0,0,1]]) * AB
     B[0,0] = trAB[0,0] + trAB[2,2]
     B[1,1] = trAB[1,1] + trAB[3,3]
     B[0,1] = trAB[0,1] + trAB[2,3]
     B[1,0] = trAB[1,0] + trAB[3,2]

     # Erweiterung der konditionierten Dichtematrix auf (2n x 2n)

     B_real = matrix(0.0, (rho.size[0]/2,rho.size[1]/2))
     for m in range(B.size[0]):
          for n in range(B.size[1]):
               B_real[m,n] = B[m,n].real
```

```
                B_real[m+B.size[0],n+B.size[1]] = B[m,n].real
                B_real[m+B.size[0],n] = B[m,n].imag
                B_real[m,n+B.size[0]] = -B[m,n].imag
    return B_real
```

Quellcodeausschnitt 6.7: Bestimmung der konditionierten Dichtematrizen

```
# Berechnung der von-Neumann Entropie einer Dichtematrix

def S(AB):
    ew = 0.0
    ew = Eigenwerte(copy.copy(AB),0)
    val = 0.0
    for i in range(AB.size[0]):
        val += h(ew[i])
    return val
```

Quellcodeausschnitt 6.8: Berechnung der von-Neumann-Entropie

Für die Ableitungen werden, wie aus Gl. (5.8) und (5.9) ersichtlich, die auf das jeweilige Ergebnis von Alice konditionierten Pauli-Matrizen benötigt. Dazu werden diese außerhalb der Hauptfunktion berechnet.

```
# Konditionierte Paulimatrizen G0, G1, G2, G3

G0 = []
G1 = []
G2 = []
G3 = []

for a in range(len(F)):
    G0.append(cond(F[a],0))
    G1.append(cond(F[a],1))
    G2.append(cond(F[a],2))
    G3.append(cond(F[a],3))
```

Quellcodeausschnitt 6.9: Konditionierte Paulimatrizen erweitert auf reelle $2n \times 2n$ Matrizen

Die Ableitungen können dann für jeden Iterationsschritt leicht bestimmt werden.

```
# 1.Ableitung der Entropie
```

```
def DS(AB,i,flag):
     [ew,ev] = Eigenwerte(copy.copy(AB),1)
     t=0.0
     if flag == 'F':
          for j in range(ew.size[0]):
               t += (1 + log(ew[j]))* eMe(ev[:,j],F[i],ev[:,j])
     if flag == 'G0':
          for j in range(ew.size[0]):
               t += (1 + log(ew[j]))* eMe(ev[:,j],G0[i],ev[:,j])
     if flag == 'G1':
          for j in range(ew.size[0]):
               t += (1 + log(ew[j]))* eMe(ev[:,j],G1[i],ev[:,j])
     if flag == 'G2':
          for j in range(ew.size[0]):
               t += (1 + log(ew[j]))* eMe(ev[:,j],G2[i],ev[:,j])
     if flag == 'G3':
          for j in range(ew.size[0]):
               t += (1 + log(ew[j]))* eMe(ev[:,j],G3[i],ev[:,j])
     return -t/log(2)

# 2. Ableitung der Entropie

def D2S(AB,i,j,flag):
     [ew,ev] = Eigenwerte(copy.copy(AB),1)
     summe = 0.0
     val = 0.0
     for k in range(ew.size[0]):
          for l in range(ew.size[0]):

               if flag == 'F':
                    val = 2*eMe(ev[:,k],F[i],ev[:,l])*eMe(ev[:,l],F[j],ev[:,
   k]) * (DPhi(ew[k],ew[l]) + 0.5*D2Phi(ew[k],ew[l])* ew[k])

               if flag == 'G0':
                    val = 2*eMe(ev[:,k],G0[i],ev[:,l])*eMe(ev[:,l],G0[j],ev
   [:,k]) * (DPhi(ew[k],ew[l]) + 0.5*D2Phi(ew[k],ew[l]) * ew[k])

               if flag == 'G1':
                    val = 2*eMe(ev[:,k],G1[i],ev[:,l])*eMe(ev[:,l],G1[j],ev
   [:,k]) * (DPhi(ew[k],ew[l]) + 0.5*D2Phi(ew[k],ew[l]) * ew[k])

               if flag == 'G2':
```

```
                    val = 2*eMe(ev[:,k],G2[i],ev[:,l])*eMe(ev[:,l],G2[j],ev
    [:,k]) * (DPhi(ew[k],ew[l]) + 0.5*D2Phi(ew[k],ew[l]) * ew[k])

                if flag == 'G3':
                    val = 2*eMe(ev[:,k],G3[i],ev[:,l])*eMe(ev[:,l],G3[j],ev
    [:,k]) * (DPhi(ew[k],ew[l]) + 0.5*D2Phi(ew[k],ew[l]) * ew[k])

                summe += val
      return -summe/log(2)

# 1. Ableitung von Holevo

def DChi(AB):
      val = matrix(0.0,(len(F),1))
      for i in range(len(F)):
          val[i] =  DS(AB,i,'F')  - 0.25 * ( DS(cond(AB,0),i,'G0') + DS(cond
    (AB,1),i,'G1') + DS(cond(AB,1),i,'G2') + DS(cond(AB,1),i,'G3') )
      return val.T

# 2. Ableitung von Holevo

def D2Chi(AB):
      val = matrix(0.0,(len(F),len(F)))
      for i in range(val.size[0]):
          for j in range(val.size[0]):
              val[i,j] = D2S(AB,i,j,'F') - 0.25* ( D2S(cond(AB,0),i,j,'G0')
     + D2S(cond(AB,1),i,j,'G1') + D2S(cond(AB,1),i,j,'G2') + D2S(cond(AB,1),i
    ,j,'G3'))
      return val
```

Quellcodeausschnitt 6.10: Berechnung der Ableitungen

```
def DPhi(k,l):
      val = 0.0
      if (k-l > 1e-9):
          val = ((log(k) - log(l)) / (k - l))
      else:
          val = 1/k
      return val

def D2Phi(k,l):
      val = 0.0
      if (k-l > 1e-9):
```

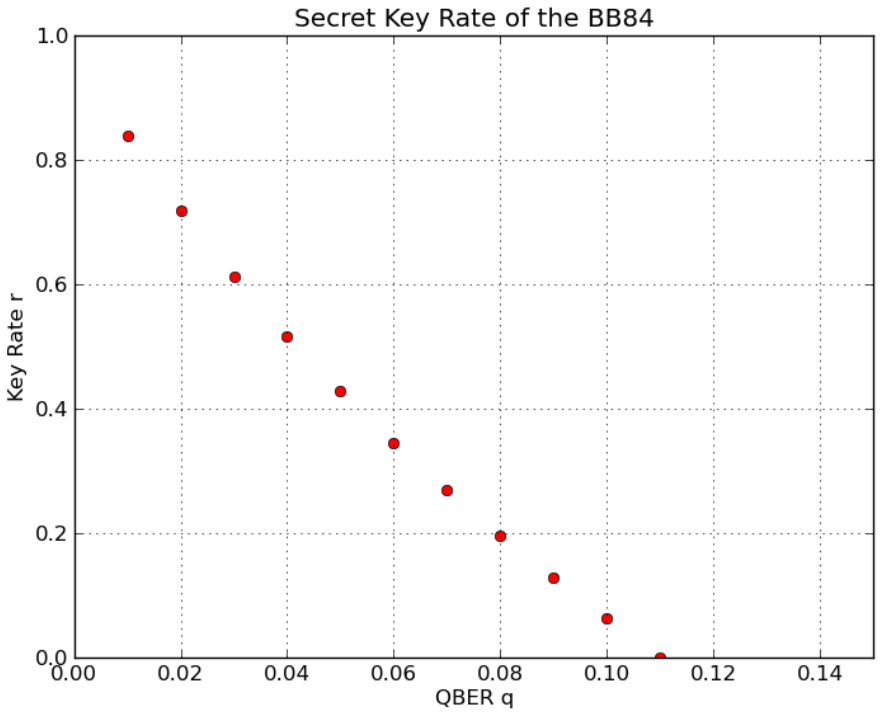

Abbildung 6.2: Schlüsselrate des BB84 abhängig von der gemessenen QBER.

```
            val = ((1/k - (log(1)-log(k))/(1-k)) / (k - 1))
      else:
            val = -1/k**2
      return val

def eMe(k,A,1):
      val = 0.0
      val = (k.T * A * 1)[0]
      return val
```

Quellcodeausschnitt 6.11: Hilfsfunktionen zur Berechnung der Ableitungen

Mit diesem Programm kann die Optimierung für verschiedene QBER durchgeführt werden. In Abb. 6.3 sind die Ergebnisse der Optimierung dargestellt. Es zeigt sich, dass in Übereinstimmung mit [RGK05] bis zu einer QBER von etwa 0.11 ein sicherer Schlüssel übertragen werden kann.

6.2.4 Verschiedene Fehlerraten für Messungen in x- und z-Richtung

Bisher wurde angenommen, dass die gemessene Qubitfehlerrate (QBER) sowohl bei Messungen in der x-Basis wie auch bei Messungen in der z-Basis gleich groß ist. Betrachtet man das Protokoll bei unterschiedlichen Fehlerraten für x- und z-Richtung, so folgt aus

$$\begin{array}{c|cccc} r_{ij} & \mathbb{1} & \sigma_x & \sigma_y & \sigma_z \\ \hline \mathbb{1} & 1 & 0 & ? & 0 \\ \sigma_x & 0 & 1-2Q_x & ? & 0 \\ \sigma_y & 0 & ? & ? & ? \\ \sigma_z & 0 & 0 & ? & 1-2Q_z \end{array} \tag{6.17}$$

für den bekannten Teil der Dichtematrix

$$\rho_{\text{fix}} = \frac{1}{2}\begin{pmatrix} 1-Q_z & 0 & 0 & \frac{1-2Q_x}{2} \\ 0 & Q_z & \frac{1-2Q_x}{2} & 0 \\ 0 & \frac{1-2Q_x}{2} & Q_z & 0 \\ \frac{1-2Q_x}{2} & 0 & 0 & 1-Q_z \end{pmatrix}. \tag{6.18}$$

Die Optimierung ist bis auf diese Veränderung identisch zur Optimierung des normalen BB84 Protokolls im Abschnitt zuvor.

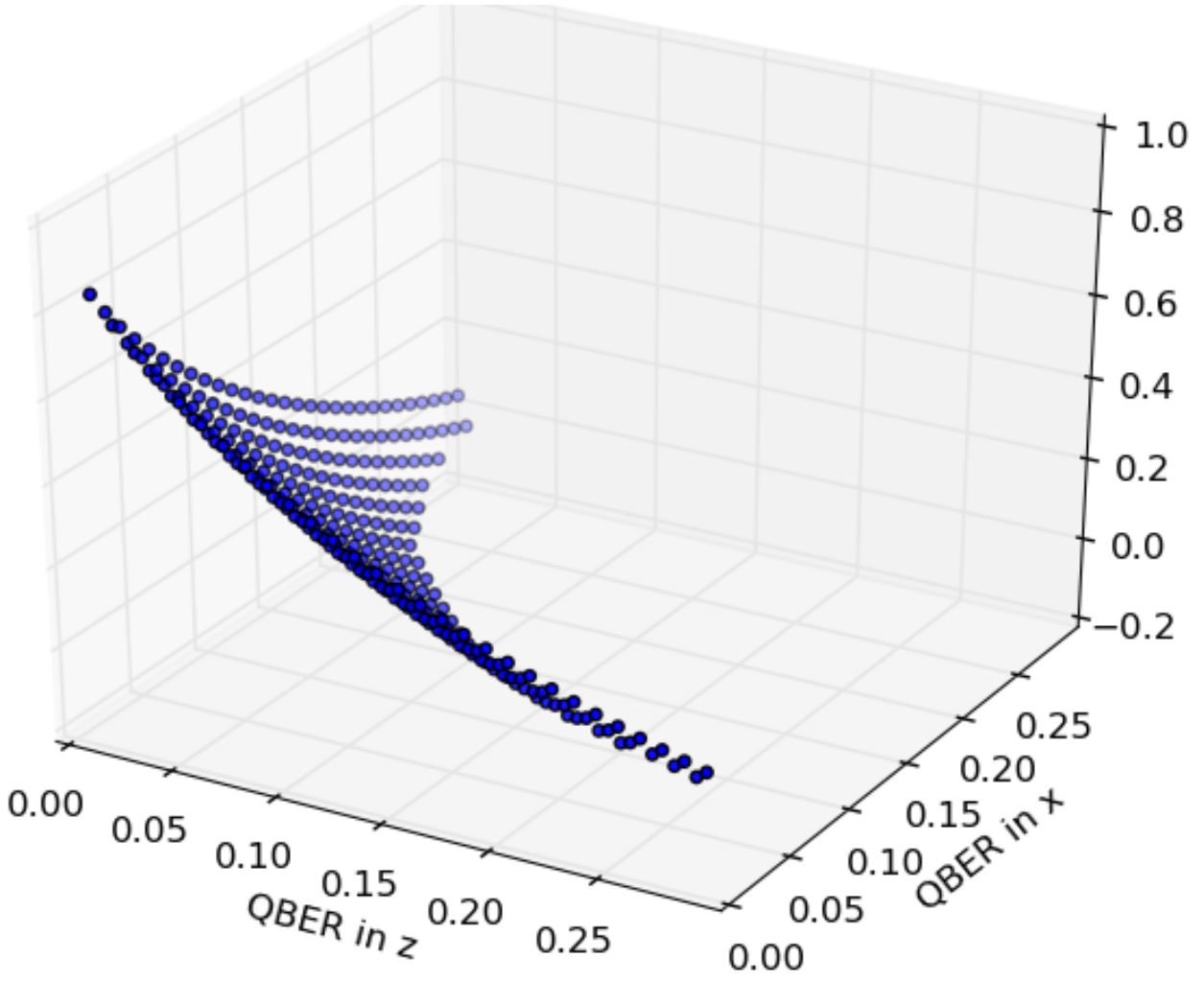

Abbildung 6.3: Schlüsselrate des BB84 für verschiedene QBER in x- und z-Richtung.

6.3 Das BB84 mit x-z Korrelation

Eine weiteres Problem ist die Frage, ob das Protokoll auch bei Daten, die nicht ganz dem depolarisierenden Kanal entsprechen, sicher ist. Dazu wird ein Kanal betrachtet, bei dem die Daten eine leichte Korrelation in der x- und z-Basis aufweisen.

r_{ij}	$\mathbb{1}$	σ_x	σ_y	σ_z
$\mathbb{1}$	1	0	?	0
σ_x	0	$1-2Q$	?	e
σ_y	0	?	?	?
σ_z	0	e	?	$1-2Q$

(6.19)

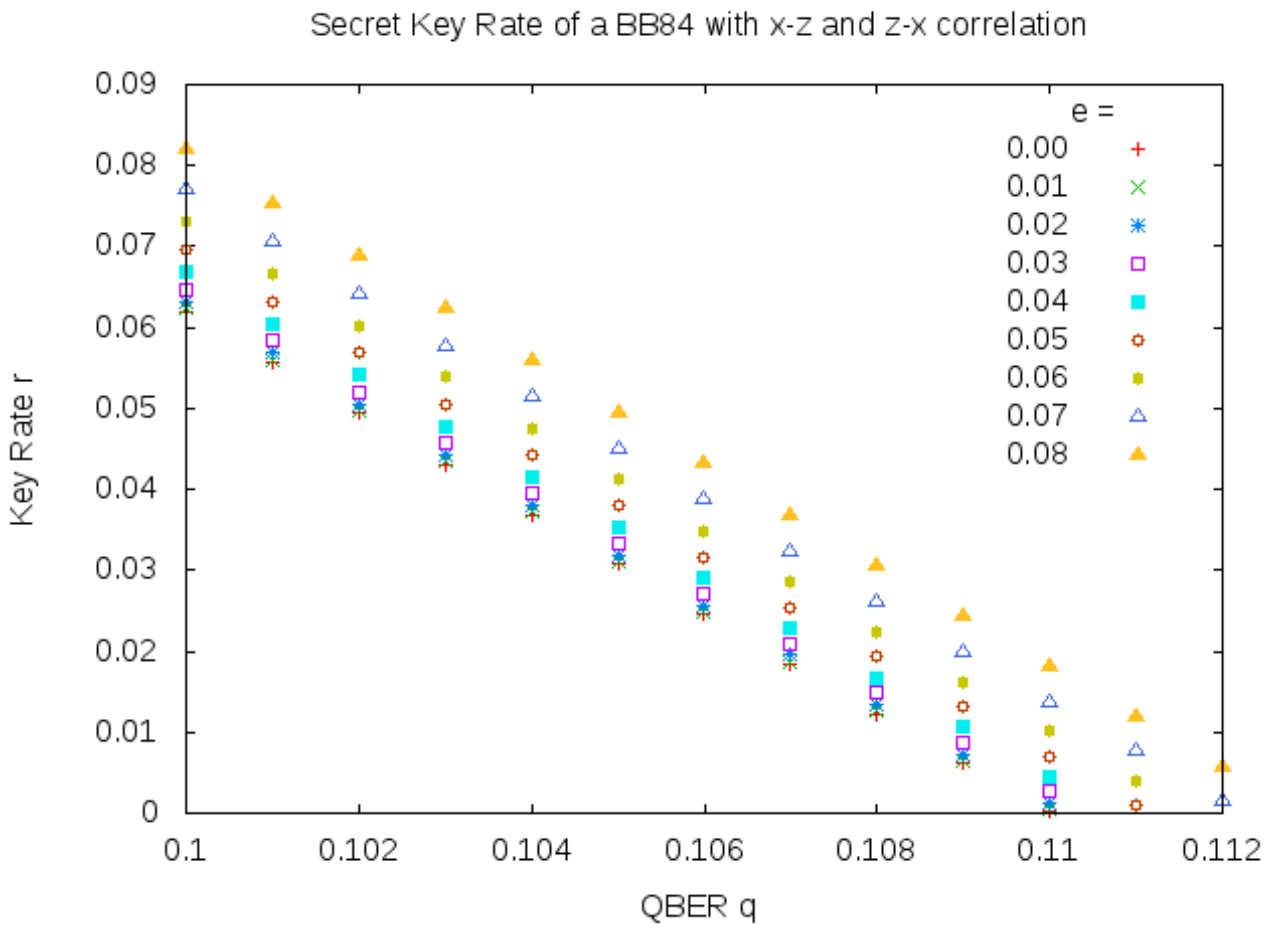

Abbildung 6.4: Schlüsselrate des BB84 bei einer x-z Korrelation

Auch für diese Daten kann mit dem Programm der vorherigen Abschnitte optimiert werden. Bemerkenswert ist hier auf den ersten Blick, dass die sichere Schlüsselrate für zunehmendes e größer wird. Allerdings ist leicht einzusehen, dass diese Korrelation in den Daten die möglichen Attacken von Eve einschränkt und so der Informationsgewinn von Eve kleiner wird. Allerdings ist zu bemerken das es für größer werdendes e sehr schwierig wird, zulässige Startpunkte der Optimierung zu finden.

6.4 Das 2-state Protokoll

Das 2-state Protokoll arbeitet, wie der Name schon ausdrückt, nur mit zwei nicht-orthogonalen Zuständen. Dazu sendet Alice einen der Zustände

$$|u_{\pm}\rangle = \beta\,|0\rangle \pm \alpha\,|1\rangle \tag{6.20}$$

an Bob. Dabei ist $\beta = \sqrt{1-\alpha^2}$ und $0 < \alpha < 1/\sqrt{2}$. Bob misst die orthogonalen Zustände in den beiden von Alice gewählten Basen

$$|\bar{u}_\pm\rangle = \alpha\,|0\rangle \mp \beta\,|1\rangle\,. \tag{6.21}$$

Falls Bob in der von Alice gewählten Basis des jeweiligen Qubits misst, so erhält er immer eine negatives Ergebnis $\langle \bar{u}_+|u_+\rangle = 0$. Unterscheiden sich die beiden Basen, so misst Bob das von Alice gesendete Qubit mit einer Wahrscheinlichkeit von $|\langle \bar{u}_-|u_+\rangle|^2 = |2\alpha\beta|^2$. Bob teilt Alice über den öffentlichen Kanal nur die Position des gemessenen Qubits mit. Damit erhalten Alice und Bob einen Schlüssel, der mit den bekannten Fehlerkorrektur und Privacy Amplification Verfahren bearbeitet werden kann.

6.4.1 Bestimmung von ρ_{AB}

Insgesamt ergibt sich für den präparierten Zustand von Alice und Bob

$$|\varphi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle\,|u_+\rangle + |1\rangle\,|u_-\rangle\right) \tag{6.22}$$

und damit

$$\rho_{AB} = \frac{1}{2}\begin{pmatrix} \beta^2 & \alpha\beta & \beta^2 & -\alpha\beta \\ \alpha\beta & \alpha^2 & \alpha\beta & -\alpha^2 \\ \beta^2 & \alpha\beta & \beta^2 & -\alpha\beta \\ -\alpha\beta & -\alpha^2 & -\alpha\beta & \alpha^2 \end{pmatrix}. \tag{6.23}$$

Auch hier gehen wir wieder von einem depolarisierenden Kanal aus, womit sich der Zustand folgendermaßen verändert

$$\begin{aligned} \rho'_{AB} &= (1-p)\rho_{AB} + \frac{p}{2}\left(\rho_A \otimes I\right) \\ &= \frac{1}{2}\begin{pmatrix} \beta^2(1-p)+\frac{p}{2} & \alpha\beta(1-p) & \beta^2-\frac{p}{2} & -\alpha\beta(1-p) \\ \alpha\beta(1-p) & \alpha^2(1-p)+\frac{p}{2} & \alpha\beta(1-p) & -\alpha^2+\frac{p}{2} \\ \beta^2-\frac{p}{2} & \alpha\beta(1-p) & \beta^2(1-p)+\frac{p}{2} & -\alpha\beta(1-p) \\ -\alpha\beta(1-p) & -\alpha^2+\frac{p}{2} & -\alpha\beta(1-p) & \alpha^2(1-p)+\frac{p}{2} \end{pmatrix}. \end{aligned} \tag{6.24}$$

Bevor Bob den Bitwert misst, durchläuft der Signalzustand einen Filter, der diesen in die z-Basis abbildet. In der Optimierung wird die sichere Schlüsselrate konditioniert auf den Erfolgsfall des Filters $r = r^{\text{success}}$ bestimmt. Der, wenn der Filter das Signal durchlässt. Daher ist der Zustand

nach dem Filter der entscheidende Zustand für die Optimierung. Der Filter

$$F = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

muss dabei so gewählt werden, dass die Signalzustände auf den passenden Zustand in der z-Basis abgebildet werden. Es gilt

$$\begin{aligned} \langle 0| F |u_+\rangle &= 2\alpha\beta, \\ \langle 1| F |u_+\rangle &= 0, \\ \langle 0| F |u_-\rangle &= 0, \\ \langle 1| F |u_-\rangle &= 2\alpha\beta. \end{aligned} \tag{6.25}$$

(6.26)

Aus diesen Voraussetzungen ergibt sich als Filter der Kraus-Operator

$$F = \begin{pmatrix} \alpha & \beta \\ \alpha & -\beta \end{pmatrix}. \tag{6.27}$$

Mit dem Filter verändert sich der Zustand ρ'_{AB} zu

$$\begin{aligned} \rho_{AB}^{\text{fil}} &= \frac{(\mathbb{1} \otimes F)\rho'_{AB}(\mathbb{1} \otimes F)^T}{\operatorname{tr}\left((\mathbb{1} \otimes F)\rho'_{AB}(\mathbb{1} \otimes F)^T\right)} \\ &= \frac{1}{4p_F} \begin{pmatrix} 8\alpha^2\beta^2(1-p)+p & (\alpha^2-\beta^2)p & -(\alpha^2-\beta^2)p & 4\alpha^2\beta^2(2-p)-p \\ (\alpha^2-\beta^2)p & p & (4\alpha^2\beta^2-1)p & -(\alpha^2-\beta^2)p \\ -(\alpha^2-\beta^2)p & (4\alpha^2\beta^2-1)p & p & (\alpha^2-\beta^2)p \\ 4\alpha^2\beta^2(2-p)-p & -(\alpha^2-\beta^2)p & (\alpha^2-\beta^2)p & 8\alpha^2\beta^2(1-p)+p \end{pmatrix} \end{aligned} \tag{6.28}$$

mit $p_F = 4\alpha^2\beta^2(1-p)+p$. Falls der Kanal rauschfrei ist und Eve nicht eingreift, ergibt sich nach dem Filter der maximal verschränkte Zustand

$$|\varphi_{\text{fil}}\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle|0\rangle + |1\rangle|1\rangle\right) \tag{6.29}$$

mit einer Wahrscheinlichkeit von $4\alpha^2\beta^2$. Nach dem Filter wird nun mit der Messung in der z-Basis die Hauptdiagonale des Zustandes ρ_{AB}^{fil} bestimmt. Damit sind die folgenden Parameter

der Dichtematrix ρ'_{AB} bekannt

r_{ij}	$\mathbb{1}$	σ_x	σ_y	σ_z
$\mathbb{1}$	1	0	?	$2(1-p)(\beta^2-\alpha^2)$
σ_x	$(\beta^2-\alpha^2)$	?	?	?
σ_y	0	?	?	?
σ_z	0	$2\alpha\beta(1-p)$	?	0

. (6.30)

Über die verbleibenden Parameter wird mit einer modifizierten Version des Programms der vorherigen Kapitel optimiert.

6.4.2 Die Schlüsselrate

Die Bestimmung der Schlüsselrate ist erneut ein nichtlineares konvexes Optimierungsproblem

$$\begin{aligned} &\min && r(\rho_{AB}^{\text{fil}}(x)) \\ &\text{u.d.Nb.} && \rho_{AB}^{\text{fil}}(x) \geq 0. \end{aligned} \tag{6.31}$$

Die wichtigste Änderung ist zunächst, dass bei der Berechnung der Dichtematrix der Filter auf diese angewendet wird.

```
# Definition der Dichtematrix

def define_rho(x):
     param = complex(0.0,0.0)
     for i in range(len(Pauli)):
          param += x[i]*Pauli[i]
     AB =  1/pF*F*(rho_fix + (param))*F.T
     return AB
```

Quellcodeausschnitt 6.12: Berechnung der Dichtematrix bei der Optimierung des B92

Dabei entspricht das hier angegebene pF der Wahrscheinlichkeit p_F, welche jeweils zu Beginn der Optimierung berechnet wird.

```
     # Bekannte Parameter
     pF = 4*(1-p)*alpha**2*beta**2 + p
     qerror = p/(2*pF)
```

Quellcodeausschnitt 6.13: Berechnung der Wahrscheinlichkeit p_F bei der Optimierung des B92

Um die Ableitungen bestimmen zu können, muss auf die Tensorprodukte der Paulimatrizen der Filter angewendet werden.

```
FPauli.append(1/pF*F*Pauli[a]*F.T)
FG0.append(1/pF*FBob*G0[a]*FBob.T)
FG1.append(1/pF*FBob*G1[a]*FBob.T)
```

Quellcodeausschnitt 6.14: Anwendung des Filters auf die Tensorprodukte der Paulimatrizen zur Bestimmung der Ableitungen

Entscheidend ist auch die Festlegung des Startpunktes der Optimierung. Eine gute Möglichkeit ergibt sich durch den gewählten Punkt $r_{yy} = 2(1-p)\alpha\beta$, $r_{xz} = (1-p)$.

```
x_start[0] = 4*0.5*(1-p)*alpha*beta
x_start[1] = (1-p)
```

Quellcodeausschnitt 6.15: Startpunkt bei der Optimierung des B92

Damit ist die Dichtematrix zu Beginn der Optimierung

$$\begin{aligned}\rho_{\text{Start}} &= \frac{1}{2}\begin{pmatrix} \beta^2(1-p)+\frac{p}{2} & \alpha\beta(1-p) & \frac{\beta^2-\alpha^2}{2}+\frac{1-p}{2} & -\alpha\beta(1-p) \\ \alpha\beta(1-p) & \alpha^2(1-p)+\frac{p}{2} & \alpha\beta(1-p) & \frac{\beta^2-\alpha^2}{2}-\frac{1-p}{2} \\ \frac{\beta^2-\alpha^2}{2}+\frac{1-p}{2} & \alpha\beta(1-p) & \beta^2(1-p)+\frac{p}{2} & -\alpha\beta(1-p) \\ -\alpha\beta(1-p) & \frac{\beta^2-\alpha^2}{2}-\frac{1-p}{2} & -\alpha\beta(1-p) & \alpha^2(1-p)+\frac{p}{2} \end{pmatrix} \\ &= \frac{1}{2}\begin{pmatrix} \beta^2(1-p)+\frac{p}{2} & \alpha\beta(1-p) & \beta^2-\frac{p}{2} & -\alpha\beta(1-p) \\ \alpha\beta(1-p) & \alpha^2(1-p)+\frac{p}{2} & \alpha\beta(1-p) & -\alpha^2+\frac{p}{2} \\ \beta^2-\frac{p}{2} & \alpha\beta(1-p) & \beta^2(1-p)+\frac{p}{2} & -\alpha\beta(1-p) \\ -\alpha\beta(1-p) & -\alpha^2+\frac{p}{2} & -\alpha\beta(1-p) & \alpha^2(1-p)+\frac{p}{2} \end{pmatrix}\end{aligned} \tag{6.32}$$

und entspricht der Dichtematrix des depolarisierenden Kanals aus Gl. 6.24.

Optimiert man nun mit diesem modifiziertem Programm, so erhält man, wie in Abbildung 6.7 ersichtlich ist, eine deutliche Verbesserung der unteren Grenze für die sichere Schlüsselrate. Die in Sicherheitsschwelle von Christiandl, Renner und Ekert [MCE] lässt sich von den angegebenen $p \approx 0.48$ auf $p \approx 0.88$ verbessern.

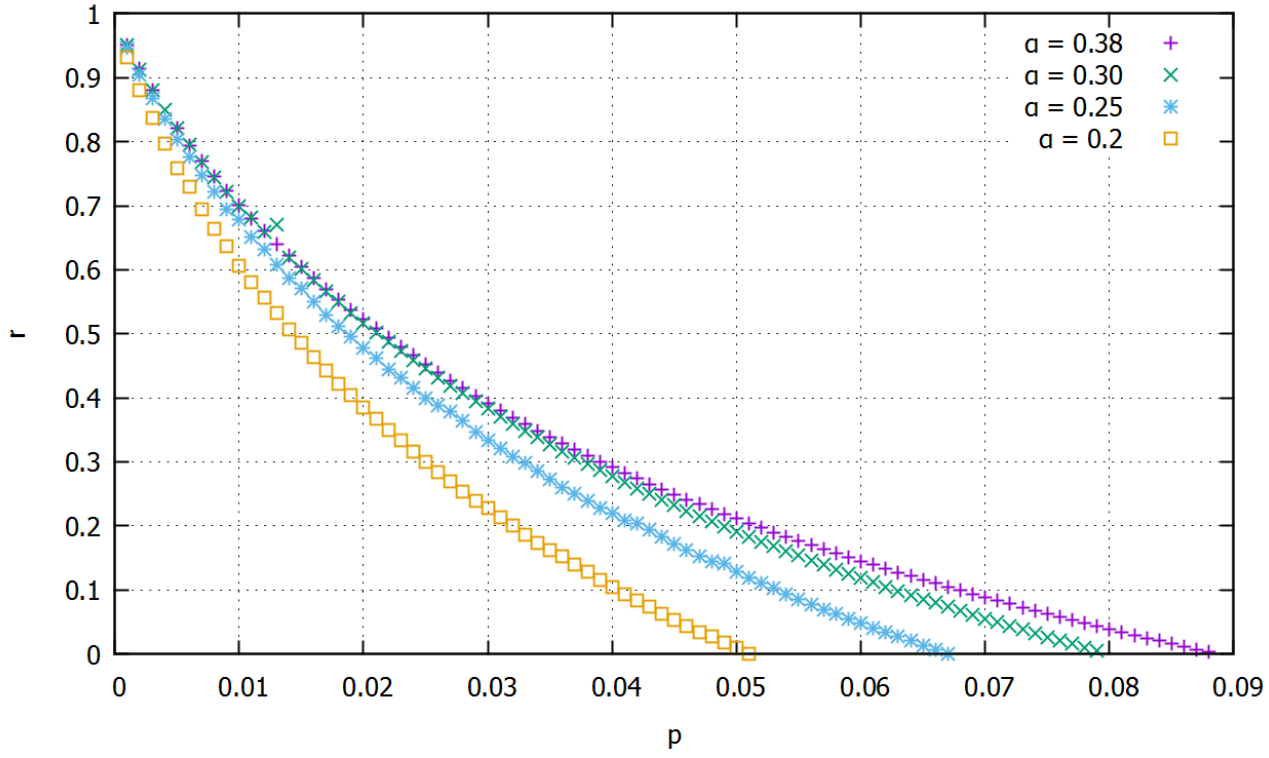

Abbildung 6.5: Schlüsselrate des 2-state Protokolls für $\alpha = 0.20$, $\alpha = 0.25$, $\alpha = 0.30$ und $\alpha = 0.38$ konditioniert auf den Erfolgsfall des Filters.

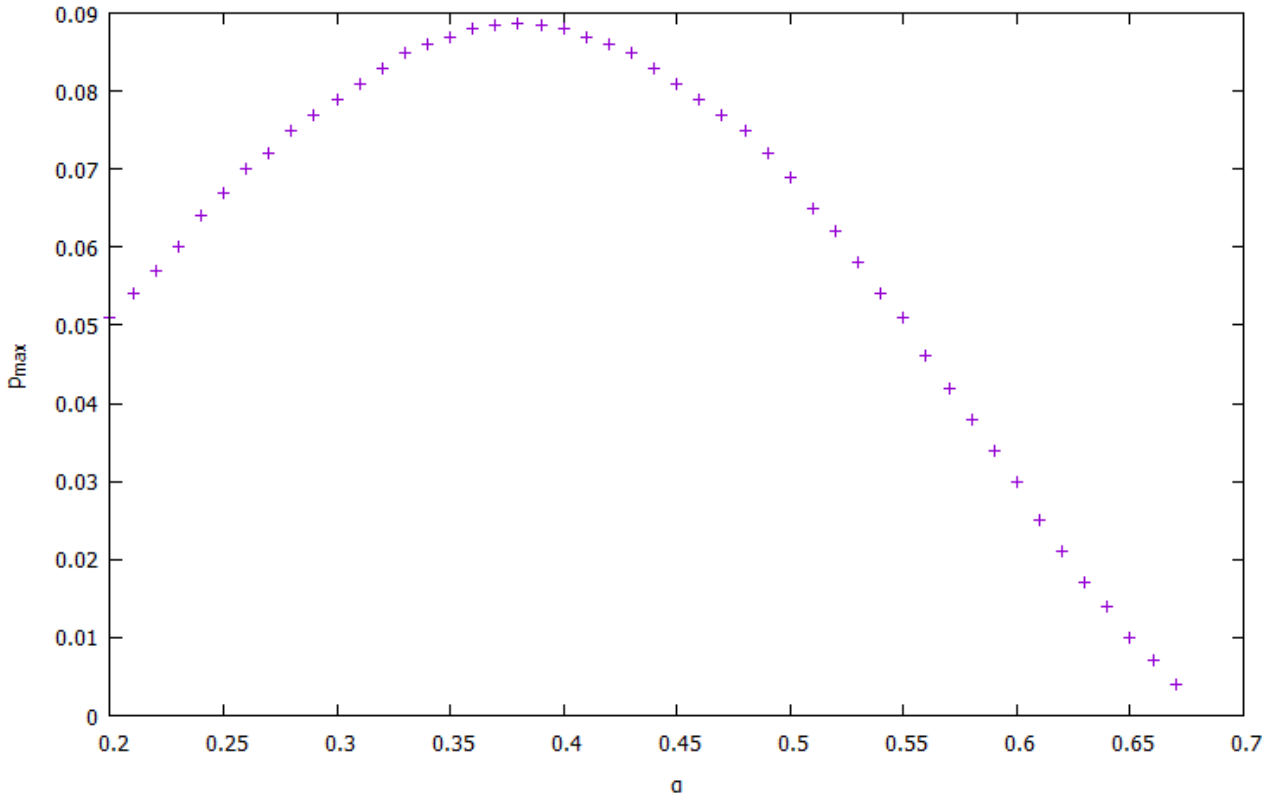

Abbildung 6.6: Maximal erlaubter Fehler beim 2-state Protokoll für verschiedene α des präparierten Zustands.

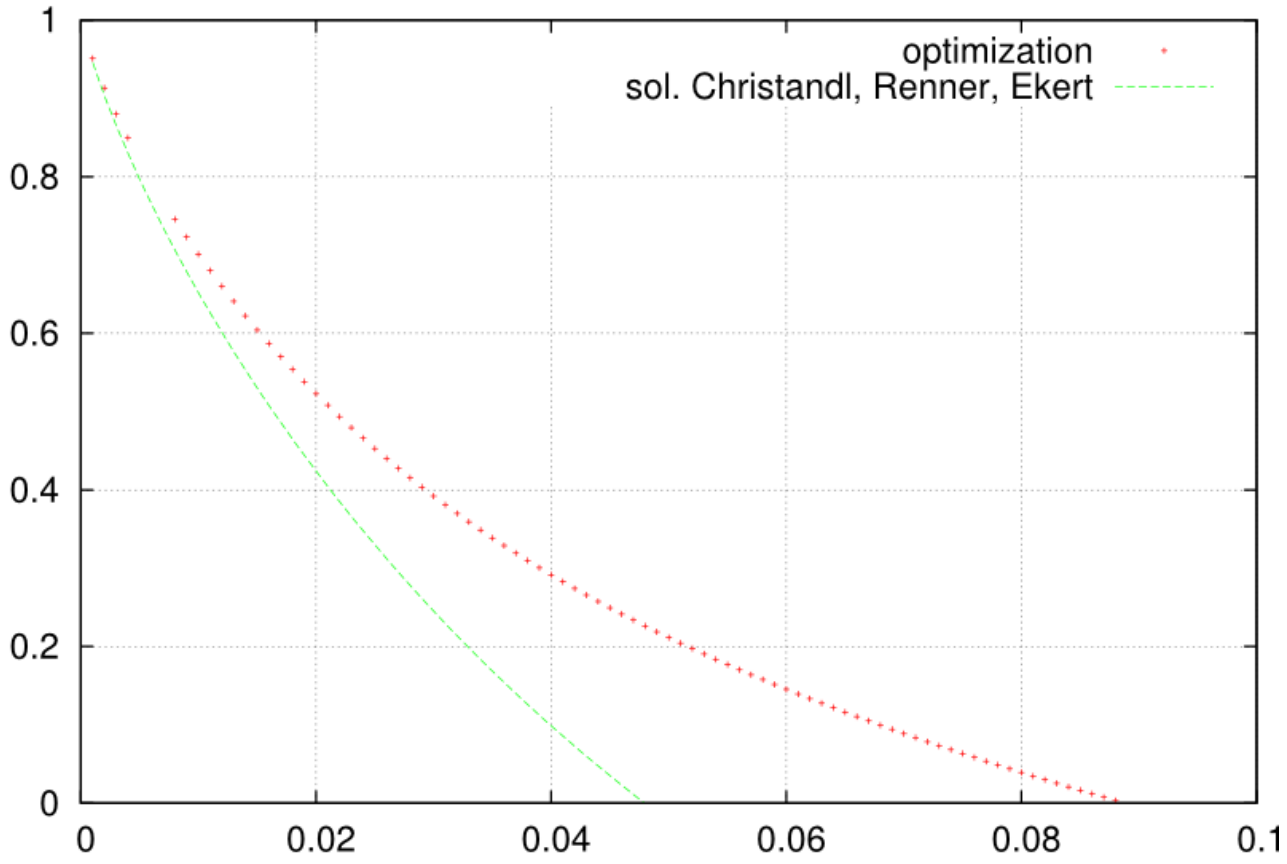

Abbildung 6.7: Vergleich des Ergebnisses der konvexen Optimierung mit der Lösung aus [MCE].

6.5 2-state Protokoll mit Kanalverlusten

Im letzten Teil dieser Arbeit soll das 2-state Protokoll so modifiziert werden, dass es Kanalverluste mit einbezieht. Diese Verluste senken die Sicherheitsschwelle des Protokolls deutlich, da Eve der Grund für den Verlust der Zustände sein kann. Um diesen zusätzlichen Informationsgewinn von Eve mit einzubeziehen ist es notwendig, die Zahl der Parameter in der Optimierung deutlich (von 8 auf 27) zu erhöhen.

6.5.1 Bestimmung von ρ_{AB}

Um den Verlust im Kanal und damit die bei Bob ankommenden Vakuumzustände mit in die Optimierung einbeziehen zu können muss die Dichtematrix aus Gl. 6.24 erweitert werden. Erhält Bob mit einer Wahrscheinlichkeit L einen Vakuumzustand so gilt für die daraus resultierende Dichtematrix

$$\begin{aligned}\rho'_{AB} &= (1-L)\left[(1-p)\rho_{AB} + \frac{p}{2}(\rho_A \otimes I)\right] \\ &\quad + \frac{L}{2}(|0\rangle|\text{vac}\rangle + |1\rangle|\text{vac}\rangle)(\langle 0|\langle\text{vac}| + \langle 1|\langle\text{vac}|) \qquad (6.33)\end{aligned}$$

$$= \frac{1-L}{2}\begin{pmatrix} \beta^2(1-p)+\frac{p}{2} & \alpha\beta(1-p) & 0 & \beta^2-\frac{p}{2} & -\alpha\beta(1-p) & 0 \\ \alpha\beta(1-p) & \alpha^2(1-p)+\frac{p}{2} & 0 & \alpha\beta(1-p) & -\alpha^2+\frac{p}{2} & 0 \\ 0 & 0 & \frac{L}{1-L} & 0 & 0 & \frac{L}{1-L} \\ \beta^2-\frac{p}{2} & \alpha\beta(1-p) & 0 & \beta^2(1-p)+\frac{p}{2} & -\alpha\beta(1-p) & 0 \\ -\alpha\beta(1-p) & -\alpha^2+\frac{p}{2} & 0 & -\alpha\beta(1-p) & \alpha^2(1-p)+\frac{p}{2} & 0 \\ 0 & 0 & \frac{L}{1-L} & 0 & 0 & \frac{L}{1-L} \end{pmatrix}.$$

Auch in diesem Fall durchläuft das Signal einen Filter. Dieser muss allerdings nicht nur die Signalzustände abbilden, sondern auch die Vakuumzustand passieren lassen. Damit wird aus dem Filter aus Gl. 6.27

$$F = \begin{pmatrix} \alpha & \beta & 0 \\ \alpha & -\beta & 0 \\ 0 & 0 & 1 \end{pmatrix}. \qquad (6.34)$$

Dann resultiert daraus der Zustand

$$\begin{aligned}\rho_{AB}^{\text{fil}} &= (1-L)\left[(1-p)\rho_{AB} + \frac{p}{2}(\rho_A \otimes I)\right] \\ &\quad + \frac{L}{2}(|0\rangle|\text{vac}\rangle + |1\rangle|\text{vac}\rangle)(\langle 0|\langle\text{vac}| + \langle 1|\langle\text{vac}|) \end{aligned} \tag{6.35}$$

$$= \frac{(1-L)}{p_F}\begin{pmatrix} \alpha^2\beta^2(1-p)+\frac{p}{4} & \frac{p}{4}(\alpha^2-\beta^2) & 0 & \frac{p}{4}(\beta^2-\alpha^2) & \alpha^2\beta^2(1-p)-\frac{p}{4} & 0 \\ \frac{p}{4}(\alpha^2-\beta^2) & \frac{p}{4} & 0 & p(\alpha^2\beta^2-\frac{1}{4}) & \frac{p}{4}(\beta^2-\alpha^2) & 0 \\ 0 & 0 & \frac{L}{2(1-L)} & 0 & 0 & \frac{L}{2(1-L)} \\ \frac{p}{4}(\beta^2-\alpha^2) & p(\alpha^2\beta^2-\frac{1}{4}) & 0 & \frac{p}{4} & \frac{p}{4}(\alpha^2-\beta^2) & 0 \\ \alpha^2\beta^2(1-p)-\frac{p}{4} & \frac{p}{4}(\beta^2-\alpha^2) & 0 & \frac{p}{4}(\alpha^2-\beta^2) & \alpha^2\beta^2(1-p)+\frac{p}{4} & 0 \\ 0 & 0 & \frac{L}{2(1-L)} & 0 & 0 & \frac{L}{2(1-L)} \end{pmatrix}$$

mit einer Erfolgswahrscheinlichkeit von $p_F = L + (L-1)\left[4\alpha^2\beta^2(1-p) + p\right]$. Wie zuvor auch folgt eine Messung in der z-Basis.

6.5.2 Parametrisierung der Dichtematrix

Da das System von Bob durch den Vakuumzustand um eine Dimension erweitert wurde, reicht es nicht mehr aus, die Dichtematrix wie in Gl. 5.6 durch die Tensorprodukte der Paulimatrizen zu parametrisieren. Statt einer Basis der SU(2) wird nun eine Basis für die SU(3) benötigt. Eine Basis dieser Gruppe sind die Gell-Mann Matrizen

$$\begin{aligned} \lambda_1 &= \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, & \lambda_2 &= \begin{pmatrix} 0 & -i & 0 \\ i & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, & \lambda_3 &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \\ \lambda_4 &= \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, & \lambda_5 &= \begin{pmatrix} 0 & 0 & -i \\ 0 & 0 & 0 \\ i & 0 & 0 \end{pmatrix}, & \lambda_6 &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \\ \lambda_7 &= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -i \\ 0 & i & 0 \end{pmatrix}, & \lambda_8 &= \frac{1}{\sqrt{3}}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -2 \end{pmatrix}. \end{aligned} \tag{6.36}$$

Zusammen mit den Pauli Matrizen bilden sie eine Basis für das Gesamtsystem SU(2)⊗SU(3) von Alice und Bob. Der Zustand kann dann durch

$$\rho = \frac{1}{6}\mathbb{1}_6 + \left(\frac{1}{6}\sum_{i=1}^{3} r_{i0}\ F_{i0} + \frac{1}{4} r_{0i}\ F_{0i}\right) + \frac{1}{4}\sum_{i=1}^{3}\sum_{i=j}^{3} r_{ij}\ F_{ij} \tag{6.37}$$

ausgedrückt werden, wobei hier $F_{ij} = \sigma_i \otimes \lambda_j$ das Tensorprodukt von Pauli und Gell-Mann Matrizen ist.
In dieser Basis sind die folgenden Parameter bekannt

$$\begin{aligned} r_{00} &= 1, \\ r_{10} &= (\beta^2 - \alpha^2), \\ r_{20} &= 0, \\ r_{30} &= 0, \\ r_{01} &= 0, \\ r_{31} &= 2(1-L)(1-p)\alpha\beta, \\ r_{03} &= 2(1-L)(1-p)(\beta^2 - \alpha^2), \\ r_{33} &= 0, \\ r_{08} &= \frac{1}{\sqrt{3}}(1-3L). \end{aligned} \tag{6.38}$$

Über die übrigen 27 Parameter muss optimiert werden.

6.5.3 Die Schlüsselrate

Die Bestimmung der Schlüsselrate im Fall des 2-state Protokolls mit Kanalverlusten ist ein konvexes Optimierungsproblem

$$\begin{aligned} &\min && r(\rho_{AB}^{\text{fil}}(x)) \\ &\text{u.d.Nb.} && \rho_{AB}^{\text{fil}}(x) \geq 0. \end{aligned} \tag{6.39}$$

Offensichtlich müssen die Tensorprodukte der Pauli und der Gell-Mann Matrizen als Basis definiert und die dazugehörigen auf das Ergebnis von Alice konditionierten Matrizen bestimmt werden.

```
# Definition der zweiparteien Pauli Gell-Mann Basis
def define_Paulis(Pauli, Fex, G0, G1):
    #---Startpunkte---#
        Pauli.append(1/4.*matrix([[0, 0, 0, 0, -1, 0], [0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [-1, 0, 0, 0, 0, 0], [0, 0, 0, 0,
     0, 0]]))
```

```
    Pauli.append(1/4.*matrix([[0, 0, 0, 1, 0, 0], [0, 0, 0, 0, -1, 0],
[0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0], [0, -1, 0, 0, 0, 0], [0, 0, 0, 0,
 0, 0]]))
    Pauli.append(1/(4.*math.sqrt(3.0))*matrix([[0, 0, 0, 1, 0, 0], [0,
0, 0, 0, 1, 0], [0, 0, 0, 0, 0, -2], [1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0,
0], [ 0, 0, -2, 0, 0, 0]]))

    #--4 bis 10--#
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 1, 0], [0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0], [0, 0, 0, 0,
0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 1j, 0], [0, 0, 0, 1j, 0, 0],
[0, 0, 0, 0, 0, 0], [0, -1j, 0, 0, 0, 0], [-1j, 0, 0, 0, 0, 0], [0, 0, 0,
 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 1j, 0, 0, 0, 0], [-1j, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1j, 0], [0, 0, 0, -1j, 0, 0], [0, 0, 0,
 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 1j, 0], [0, 0, 0, -1j, 0, 0],
 [0, 0, 0, 0, 0, 0], [0, 1j, 0, 0, 0, 0], [-1j, 0, 0, 0, 0, 0], [0, 0, 0,
 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 1j, 0, 0, 0, 0], [-1j, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, -1j, 0], [0, 0, 0, 1j, 0, 0], [0, 0, 0,
 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 1j, 0, 0], [0, 0, 0, 0, -1j, 0],
 [0, 0, 0, 0, 0, 0], [-1j, 0, 0, 0, 0, 0], [0, 1j, 0, 0, 0, 0], [0, 0, 0,
 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0], [0, 0, 0, 1,
0, 0]]))

    #--11 bis 20--#
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0], [1, 0, 0, 0,
0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 1j], [0, 0, 0, 0, 0, 0],
[0, 0, 0, 1j, 0, 0], [0, 0, -1j, 0, 0, 0], [0, 0, 0, 0, 0, 0], [-1j, 0,
0, 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, -1], [0, 0, 0, 0, 0, 0], [0, 0, 0,
-1, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 1j, 0, 0, 0], [0, 0, 0, 0, 0, 0],
[-1j, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1j], [0, 0, 0, 0, 0, 0], [0, 0, 0,
-1j, 0, 0]]))
```

```
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 1j], [0, 0, 0, 0, 0, 0],
[0, 0, 0, -1j, 0, 0], [0, 0, 1j, 0, 0, 0], [0, 0, 0, 0, 0, 0], [-1j, 0,
0, 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, -1], [0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0], [-1, 0, 0, 0,
 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 1j, 0, 0, 0], [0, 0, 0, 0, 0, 0],
[-1j, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, -1j], [0, 0, 0, 0, 0, 0], [0, 0, 0,
 1j, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0],
[0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1], [ 0, 0, 0, 0,
 1, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1],
[0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 1, 0, 0,
0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1j],
[0, 0, 0, 0, 1j, 0], [0, 0, 0, 0, 0, 0], [0, 0, -1j, 0, 0, 0], [0, -1j,
0, 0, 0, 0]]))

    #--21 bis 26--#
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0],
[0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, -1], [ 0, 0, 0,
0, -1, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 1j, 0, 0, 0],
[0, -1j, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1j], [ 0, 0, 0,
 0, -1j, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1j],
[0, 0, 0, 0, -1j, 0], [0, 0, 0, 0, 0, 0], [0, 0, 1j, 0, 0, 0], [0, -1j,
0, 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, -1],
[0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, -1, 0, 0,
 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 1j, 0, 0, 0],
[0, -1j, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, -1j], [ 0, 0,
0, 0, 1j, 0]]))
    Pauli.append(1/(4.*math.sqrt(3.0))*matrix([[0, 0, 0, 1j, 0, 0], [0,
0, 0, 0, 1j, 0], [0, 0, 0, 0, 0, -2j], [-1j, 0, 0, 0, 0, 0], [0, -1j, 0,
0, 0, 0], [ 0, 0, 2j, 0, 0, 0]]))

    for i in range(len(Pauli)):
      mat = matrix(0.0, (2*Pauli[0].size[0],2*Pauli[0].size[1]))
      for j in range(Pauli[0].size[0]):
           for k in range(Pauli[0].size[1]):
```

```
                    mat[j,k] = Pauli[i][j,k].real
                    mat[j+Pauli[0].size[0],k+Pauli[0].size[0]] = Pauli[i][j,
k].real
                    mat[j+Pauli[0].size[0],k] = Pauli[i][j,k].imag
                    mat[j,k+Pauli[0].size[0]] = -Pauli[i][j,k].imag
        Fex.append(mat)

    for a in range(len(Pauli)):
        G0.append(cond(copy.copy(Pauli[a]),0))
        G1.append(cond(copy.copy(Pauli[a]),1))
```

Quellcodeausschnitt 6.16: Definition der Basis des B92 mit Kanalverlusten

Daneben werden außerdem noch die bekannten Einträge der Dichtematrix nach den Angaben aus Gl. 6.38 mit dem passenden Vorfaktor aus Gl. 6.37 modifiziert. Mit der passenden Wahl des Startpunktes $r_{y2} = 2(1-p)(1-L)\alpha\beta$, $r_{x3} = (1-p)(1-L)$ und $r_{x8} = 1/\sqrt{3}(\beta^2 - \alpha^2)(1-3L)$ ergibt sich der Zustand des depolarisierenden Kanals.

```
    #Startpunkt
    x_start[0] = 2*(1-p)*(1-L)*alpha*beta
    x_start[1] = (1-p)*(1-L)
    x_start[2] = 1/math.sqrt(3.0)*(beta**2-alpha**2)*(1-3*L)

    # Bekannte Parameter
    gamma = (1-L)*(2*(1-p)*alpha**2*beta**2 + 0.5*p + 0.5*L/(1-L))
    pF = 2*gamma
    a = 1/6.0*complex(1,0)
    b = 1/4.0*complex((1-L)*(1-p)*(beta**2-alpha**2),0)
    c = 1/12.0*complex((1-3*L),0)
    d = 1/4.0*complex(2*(1-L)*(1-p)*alpha*beta,0)
    e = 1/6.0*complex(beta**2-alpha**2,0)
    rho_fix = matrix([[a+b+c, d, 0, e, 0, 0], [d, a-b+c, 0, 0, e, 0], [0,
0, a-2*c, 0, 0, e], [e, 0, 0, a+b+c, -d, 0], [0, e, 0, -d, a-b+c, 0], [0,
 0, e, 0, 0, a-2*c]])
```

Quellcodeausschnitt 6.17: Festlegung des Startpunktes beim B92 mit Kanalverlusten

Die letzte Entscheidende Veränderung ist die Berechnung der klassischen Information. Bob erhält im Messprozess nicht mehr nur die Ergebnisse 0 und 1, sondern misst auch mit einer Wahrscheinlichkeit L den Vakuumzustand. Damit verändert sich die gemeinsame klassische

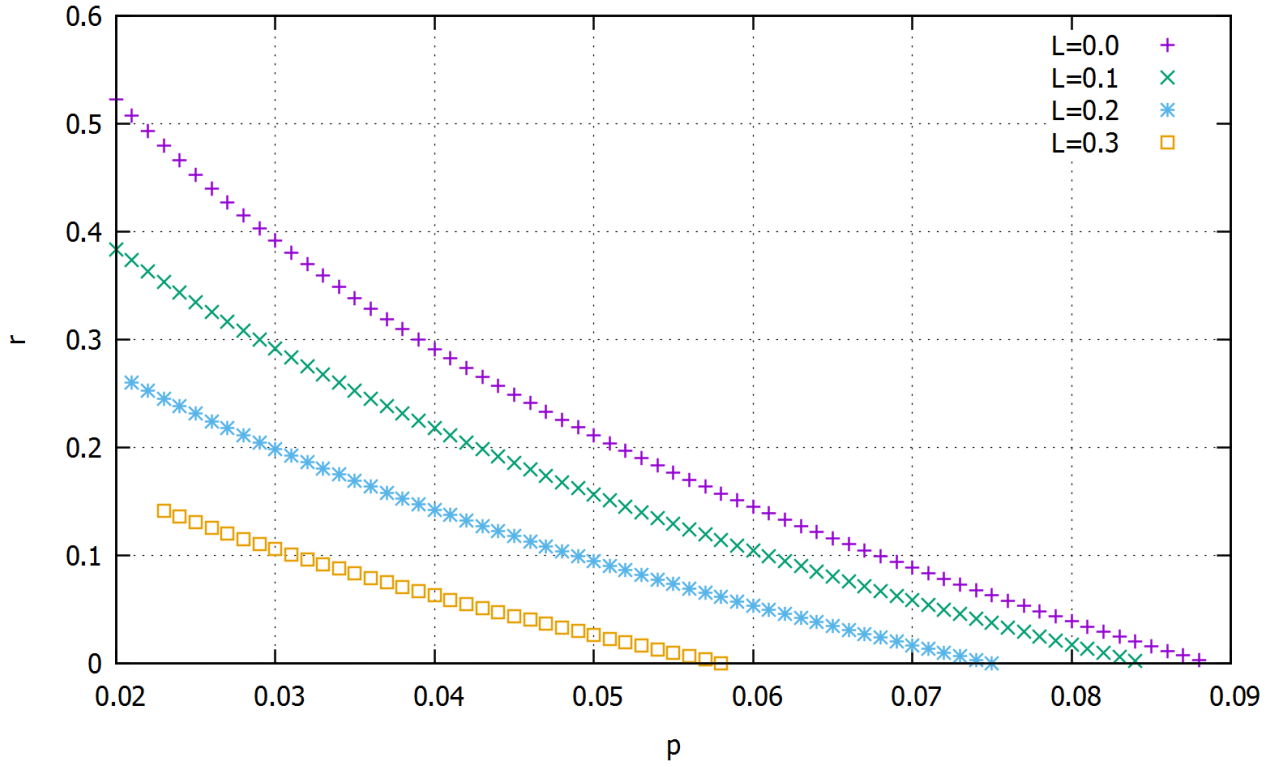

ABBILDUNG 6.8: Schlüsselrate des B92 mit Kanalverlusten für $L = 0.0$, $L = 0.1$, $L = 0.2$ und $L = 0.3$ konditioniert auf den Erfolgsfall des Filters

Information von Alice und Bob zu

$$I(X:Y) = (1-L)\left[1 + H(Q) + H(1-Q)\right]. \tag{6.40}$$

```
I = (1-L)*(1 - h(qerror) - h(1-qerror))
```

QUELLCODEAUSSCHNITT 6.18: klassiche Information beim B92 mit Kanalverlusten

Mit diesem veränderten Programm kann die Schlüsselrate im Fall des 2-state Protokolls mit Kanalverlusten aus Gl. 6.39 optimiert werden. Die Ergebnisse sind in Abbildung 6.8 dargestellt. Leider schafft es der Algorithmus des Programms nicht, die Schlüsselrate für kleine p zu optimieren, da die Anzahl der Iterationsschritte für kleine p immer größer wird und dazu führt, dass der Speicher vollläuft. Der Bereich der Sicherheitsschwelle ist allerdings optimierbar und es zeigt sich wie im verlustfreien Fall aus Abschnitt 6.4 eine deutliche Verbesserung im Vergleich zur Lösung aus [TL]. Dort liegt die Sicherheitsschwelle bei $p \approx 0.031$ im Vergleich zu der hier erreichten Sicherheitsschwelle von $p \approx 0.075$ für einen Kanalverlust von $L = 0.2$.

Kapitel 7

Zusammenfassung

In dieser Arbeit wurde zunächst ein Konzept entwickelt, wie es möglich ist, die untere Schranke der sicheren Schlüsselrate numerisch zu bestimmen. Dazu wurde gezeigt, dass sich die Schlüsselrate unabhängig von Eve ausdrücken lässt. Damit ist es möglich die Schlüsselrate mittels numerischer Optimierungsverfahren zu berechnen. Die Schlüsselrate ist, wie gezeigt wurde, eine konvexe Funktion. Somit liegt ein konvexes Optimierungsproblem vor, das mit Hilfe des Programmpakets *cvxopt* gelöst werden kann. Dazu werden die erste und zweite Ableitung der Schlüsselrate benötigt, die mit Hilfe der Matrix Analysis leicht zu bestimmen sind.
Die Anwendung des Konzepts beim BB84 liefert das erwartete Ergebnis aus [RGK05] mit einer Sicherheitsschwelle von $Q \approx 0.11$. Danach wurden Sonderfälle des BB84 betrachtet. Erhält man bei der Durchführung des Protokolls in x- und z-Richtung verschiedene Fehlerraten Q_x und Q_z, so ist trotzdem ein sicherer Schlüssel möglich.
Die interessanteren Fälle sind die Protokolle, für die es nur analytisch abgeschätzte Schlüsselraten gibt. Dazu zählt zum Beispiel das 2-state Protokoll (B92). Da nur zwei Signalzustände benutzt werden, ist hier die Zahl der unbekannten Parameter mit 8 um zwei höher als beim BB84. Die konvexe Optimierung liefert eine Sicherheitsschwelle von $p \approx 0.088$. Im Vergleich dazu liefert die Abschätzung aus [MCE] einen deutlich geringeren Wert mit $p \approx 0.048$.
Dieses Protokoll kann modifiziert werden, indem man Kanalverluste mitberücksichtigt. Auch wenn Bob Vakuumzustände erhält, lässt dies auf das Vorhandensein eines Lauschers schließen.Die Zahl der unbekannten Parameter steigt durch das Hinzukommen des Vakuumzustandes auf 27 und verlangsamt den Optimierungsalgorithmus deutlich. Durch den zusätzlichen Informationsgewinn von Eve sinkt die Sicherheitsschwelle ab liegt aber mit $p \approx 0.075$ für eine Verlustrate von $L = 0.2$ noch deutlich über dem analytischen Ergebnis aus [TL].

Anhang A

Quellcode des Optimierungsprogramms für das BB84

In diesem Abschnitt ist das Programm zur Ermittlung der sicheren Schlüsselrate des BB84 aus Kapitel 6.2 dargestellt.

```
import math
import numpy
import cvxopt
import cvxopt.lapack
from cvxopt import matrix, log, div, spdiag, solvers, spmatrix
import pdb
import copy
import matplotlib.pyplot as plt
from numpy import array

# Definition der Dichtematrix

def define_rho(x):
     param = 0.0
     for i in range(len(F)):
          param += x[i]*F[i]
     AB =  rho_fix + (param)
     return AB

# Berechnung der Eigenwerte

def Eigenwerte(A,i):
     W = matrix(0.0,(A.size[0],1))
```

```
     EV = matrix(0.0,A.size)
     if i==0:
          cvxopt.lapack.syevr(A, W, jobz = 'N')
          return W
     if i==1:
          cvxopt.lapack.syevr(A, W, jobz = 'V', range = 'A', Z = EV)
          return [W, EV]

# Binaere Entropie
def h(p):
     val = 0
     if p > 0:
          val = -p*(log(p)/log(2))
     return val

# Berchnung der auf das Resultat a knoditionierten Dichtematrizen

def cond(rho,a):
     AB = matrix(complex(0,0),(rho.size[0]/2,rho.size[1]/2))
     for i in range(AB.size[0]):
          for l in range(AB.size[1]):
               AB[i,l] =complex(rho[i,l],rho[i+rho.size[0]/2,l])
     B = matrix(complex(0,0),(rho.size[0]/4,rho.size[1]/4))

     # Die verschiedenen Konditionierungen

         if a == 0:
          trAB = spdiag([1.0,1.0,0,0]) * AB * 2
     if a == 1:
          trAB = spdiag([0,0,1.0,1.0]) * AB * 2
     if a == 2:
          trAB = matrix
   ([[1.0,0,1.0,0],[0,1.0,0,1.0],[1.0,0,1.0,0],[0,1.0,0,1.0]]) * AB
     if a == 3:
          trAB = matrix
   ([[1.0,0,-1.0,0],[0,1.0,0,-1.0],[-1.0,0,1.0,0],[0,-1.0,0,1]]) * AB
     B[0,0] = trAB[0,0] + trAB[2,2]
     B[1,1] = trAB[1,1] + trAB[3,3]
     B[0,1] = trAB[0,1] + trAB[2,3]
     B[1,0] = trAB[1,0] + trAB[3,2]

     # Erweiterung der konditionierten Dichtematrix auf (2n x 2n)
```

```
    B_real = matrix(0.0, (rho.size[0]/2,rho.size[1]/2))
    for m in range(B.size[0]):
        for n in range(B.size[1]):
            B_real[m,n] = B[m,n].real
            B_real[m+B.size[0],n+B.size[1]] = B[m,n].real
            B_real[m+B.size[0],n] = B[m,n].imag
            B_real[m,n+B.size[0]] = -B[m,n].imag
    return B_real

# Berechnung der von-Neumann Entropie einer Dichtematrix

def S(AB):
    ew = 0.0
    ew = Eigenwerte(copy.copy(AB),0)
    val = 0.0
    for i in range(AB.size[0]):
        val += h(ew[i])
    return val

def DPhi(k,l):
    val = 0.0
    if (k-l > 1e-9):
        val = ((log(k) - log(l)) / (k - l))
    else:
        val = 1/k
    return val

def D2Phi(k,l):
    val = 0.0
    if (k-l > 1e-9):
        val = ((1/k - (log(l)-log(k))/(l-k)) / (k - l))
    else:
        val = -1/k**2
    return val

def eMe(k,A,l):
    val = 0.0
    val = (k.T * A * l)[0]
    return val

# 1.Ableitung der Entropie

def DS(AB,i,flag):
```

```
    [ew,ev] = Eigenwerte(copy.copy(AB),1)
    t=0.0
    if flag == 'F':
        for j in range(ew.size[0]):
            t += (1 + log(ew[j]))* eMe(ev[:,j],F[i],ev[:,j])
    if flag == 'G0':
        for j in range(ew.size[0]):
            t += (1 + log(ew[j]))* eMe(ev[:,j],G0[i],ev[:,j])
    if flag == 'G1':
        for j in range(ew.size[0]):
            t += (1 + log(ew[j]))* eMe(ev[:,j],G1[i],ev[:,j])
    if flag == 'G2':
        for j in range(ew.size[0]):
            t += (1 + log(ew[j]))* eMe(ev[:,j],G2[i],ev[:,j])
    if flag == 'G3':
        for j in range(ew.size[0]):
            t += (1 + log(ew[j]))* eMe(ev[:,j],G3[i],ev[:,j])
    return -t/log(2)

# 2. Ableitung der Entropie

def D2S(AB,i,j,flag):
    [ew,ev] = Eigenwerte(copy.copy(AB),1)
    summe = 0.0
    val = 0.0
    for k in range(ew.size[0]):
        for l in range(ew.size[0]):

            if flag == 'F':
                val = 2*eMe(ev[:,k],F[i],ev[:,l])*eMe(ev[:,l],F[j],ev[:,
k]) * (DPhi(ew[k],ew[l]) + 0.5*D2Phi(ew[k],ew[l])* ew[k])

            if flag == 'G0':
                val = 2*eMe(ev[:,k],G0[i],ev[:,l])*eMe(ev[:,l],G0[j],ev
[:,k]) * (DPhi(ew[k],ew[l]) + 0.5*D2Phi(ew[k],ew[l]) * ew[k])

            if flag == 'G1':
                val = 2*eMe(ev[:,k],G1[i],ev[:,l])*eMe(ev[:,l],G1[j],ev
[:,k]) * (DPhi(ew[k],ew[l]) + 0.5*D2Phi(ew[k],ew[l]) * ew[k])

            if flag == 'G2':
                val = 2*eMe(ev[:,k],G2[i],ev[:,l])*eMe(ev[:,l],G2[j],ev
[:,k]) * (DPhi(ew[k],ew[l]) + 0.5*D2Phi(ew[k],ew[l]) * ew[k])
```

```
                if flag == 'G3':
                    val = 2*eMe(ev[:,k],G3[i],ev[:,l])*eMe(ev[:,l],G3[j],ev
    [:,k]) * (DPhi(ew[k],ew[l]) + 0.5*D2Phi(ew[k],ew[l]) * ew[k])

                summe += val
        return -summe/log(2)

# 1. Ableitung von Holevo

def DChi(AB):
        val = matrix(0.0,(len(F),1))
        for i in range(len(F)):
            val[i] =  DS(AB,i,'F')  - 0.25 * ( DS(cond(AB,0),i,'G0') + DS(cond
    (AB,1),i,'G1') + DS(cond(AB,1),i,'G2') + DS(cond(AB,1),i,'G3') )
        return val.T

# 2. Ableitung von Holevo

def D2Chi(AB):
        val = matrix(0.0,(len(F),len(F)))
        for i in range(val.size[0]):
            for j in range(val.size[0]):
                val[i,j] = D2S(AB,i,j,'F') - 0.25* ( D2S(cond(AB,0),i,j,'G0')
     + D2S(cond(AB,1),i,j,'G1') + D2S(cond(AB,1),i,j,'G2') + D2S(cond(AB,1),i
    ,j,'G3'))
        return val

# Hauptfunktion fuer den Solver

def CVXOpt(x = None, z = None):
            # Startpunkt der Optimierung festlegen
        if x is None:
            return 0, x_start

        # Bestimmung der Dichtematrix
        rho = define_rho(x)

        # Ist die Dichtematrix postiv-semidefinit?
        ew = Eigenwerte(copy.copy(rho),0)
        for i in range(rho.size[0]):
            if ew[i] <= 0:
                return None
```

```
    # Bestimmung der Holevo-Groesse
    Chi = S(copy.copy(rho)) - 0.25 * ( S(cond(copy.copy(rho),0)) + S(cond(
  copy.copy(rho),1)) + S(cond(copy.copy(rho),2)) + S(cond(copy.copy(rho),3)
  ))

    # Berechnung der Schluesselrate
    r = (I - 0.5 * Chi)

    # Erste Ableitung der Schluesselrate
    Dr = - 0.5 * DChi(rho)
    if z is None:
        return r, Dr

    # Zweite Ableitung der Schluesselrate
    H = - 0.5 * z[0]  * D2Chi(rho)
    return r, Dr, H

# Matrizen der Paulidarstellung F fuer die freien Parameter x
Pauli = []
F = []

Pauli.append(matrix([[0, 0, 0, -1], [0, 0, 1, 0], [0, 1, 0, 0], [-1, 0, 0,
  0]]))
Pauli.append(matrix([[0, +1j, 0, 0],[-1j, 0, 0, 0], [0, 0, 0, +1j], [0, 0,
  -1j, 0]]))
Pauli.append(matrix([[0, 0, 0, 1j],[0, 0, -1j, 0], [0, 1j, 0, 0],[-1j, 0, 0,
   0]]))
Pauli.append(matrix([[0, 0, 0, 1j],[0, 0, 1j, 0], [0, -1j, 0, 0],[-1j, 0, 0,
   0]]))
Pauli.append(matrix([[0, 0, +1j, 0],[0, 0, 0, -1j],[-1j, 0, 0, 0],[0, +1j,
  0, 0]]))
Pauli.append(matrix([[0, 1j, 0, 0],[-1j, 0, 0, 0], [0, 0, 0, -1j], [0, 0, 1j
  , 0]]))
Pauli.append(matrix([[0, 0, 1j, 0],[0, 0, 0, 1j], [-1j, 0, 0, 0], [0, -1j,
  0, 0]]))

for i in range(len(Pauli)):
    mat = matrix(0.0, (2*Pauli[0].size[0],2*Pauli[0].size[1]))
    for j in range(Pauli[0].size[0]):
        for k in range(Pauli[0].size[1]):
            mat[j,k] = Pauli[i][j,k].real
```

```
                mat[j+Pauli[0].size[0],k+Pauli[0].size[0]] = Pauli[i][j,k].
    real
                mat[j+Pauli[0].size[0],k] = Pauli[i][j,k].imag
                mat[j,k+Pauli[0].size[0]] = -Pauli[i][j,k].imag
      F.append(0.25*mat)

# Konditionierte Paulimatrizen G0, G1, G2, G3

G0 = []
G1 = []
G2 = []
G3 = []

for a in range(len(F)):
      G0.append(cond(F[a],0))
      G1.append(cond(F[a],1))
      G2.append(cond(F[a],2))
      G3.append(cond(F[a],3))

xdata = []
ydata = []

I=0

for qpoint in range(150):

      # Bekannte Parameter
      qerror = (qpoint+1)*0.001
      print '-----------------------------------'
      print '--QBER--'
      print qerror
      p = 2*qerror

      # Startpunkt
      print 'Startpunkt: '
      x_start = matrix(0.0,(len(Pauli),1))
      x_start[0] = -(1-p)
      print x_start.T

      p0x = 0.5
      p0y = 0.5

      # Fixer Teil der Dichtematrix
```

```
  rho_fix = 0.25*matrix([[2-p, 0, 0, (1-p), 0, 0, 0, 0],[0, p, (1-p), 0,
0, 0, 0, 0], [0, (1-p), p, 0, 0, 0, 0, 0], [(1-p), 0, 0, 2-p, 0, 0, 0,
0], [0, 0, 0, 0, 2-p, 0, 0, (1-p)],[0, 0, 0, 0, 0, p, (1-p), 0], [0, 0,
0, 0, 0, (1-p), p, 0], [0, 0, 0, 0, (1-p), 0, 0, 2-p]])

      rho_start = define_rho(x_start)

  SDconstraint = matrix(0.0, (F[0].size[0]*F[0].size[1], x_start.size[0])
)
  for i in range(len(F)):
       SDconstraint[:,i] = - matrix([F[i][:,0], F[i][:,1], F[i][:,2], F[i
][:,3], F[i][:,4], F[i][:,5], F[i][:,6], F[i][:,7]])
  sdconstraint = matrix([rho_fix[:,0], rho_fix[:,1], rho_fix[:,2],
rho_fix[:,3], rho_fix[:,4], rho_fix[:,5], rho_fix[:,6], rho_fix[:,7]])
  dims = {'l': 0, 'q': [], 's': [8]}

  solvers.options['maxiters']= 100
  solvers.options['show_progress']=False
  sol = solvers.cp(CVXOpt,SDconstraint,sdconstraint,dims)

  x_end = sol['x']
  rho_end=(define_rho(x_end))

  # Bestimmung der klassischen Information

  I = 1 - h(qerror) - h(1-qerror)

  Chi_start = S(copy.copy(rho_start)) - 0.25 * ( S(cond(copy.copy(
rho_start),0)) + S(cond(copy.copy(rho_start),1)) + S(cond(copy.copy(
rho_start),2)) + S(cond(copy.copy(rho_start),3)))
  Chi_end = S(copy.copy(rho_end)) - 0.25 * ( S(cond(copy.copy(rho_end),0)
) + S(cond(copy.copy(rho_end),1)) + S(cond(copy.copy(rho_end),2)) + S(
cond(copy.copy(rho_end),3)))
  print 'Schluesselrate r_Start: '
  print I - 0.5 * Chi_start
  print 'Schluesselrate r_Ende: '
  r_ende = I - 0.5*Chi_end
  print r_ende

  if r_ende <= 0:
              break

  xdata.append(qerror)
```

```
    ydata.append(r_ende)


plt.title('Secret Key Rate of the BB84')
plt.axis([0.0, 0.15, 0., 1.])
plt.plot(xdata, ydata, 'ro')
plt.xlabel('QBER q')
plt.ylabel('Key Rate r')
plt.grid(True)
plt.show()
```

Quellcodeausschnitt A.1: Quellcode des Optimierungsprogramms für das BB84

Anhang B

Quellcode des Optimierungsprogramms für das BB84 mit verschiedenen Fehlerraten in x- und z-Richtung

In diesem Kapitel befindet sich das vollständige Programm zur Ermittlung der Schlüsselrate aus Kapitel 6.24.

```
import math
import numpy
import cvxopt
import cvxopt.lapack
from cvxopt import matrix, log, div, spdiag, solvers, spmatrix
import pdb
import copy
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from numpy import array

# Definition der Dichtematrix

def define_rho(x):
     param = 0.0
     for i in range(len(F)):
          param += x[i]*F[i]
     AB =  rho_fix + (param)
     return AB
```

```
#Berechnung der Eigenwerte

def Eigenwerte(A,i):
     W = matrix(0.0,(A.size[0],1))
     EV = matrix(0.0,A.size)
     if i==0:
          cvxopt.lapack.syevr(A, W, jobz = 'N')
          return W
     if i==1:
          cvxopt.lapack.syevr(A, W, jobz = 'V', range = 'A', Z = EV)
          return [W, EV]


# Binaere Entropie
def h(p):
     val = 0
     if p > 0:
          val = -p*(log(p)/log(2))
     return val

# Berchnung der auf das Resultat a knoditionierten Dichtematrizen

def cond(rho,a):
     AB = matrix(complex(0,0),(rho.size[0]/2,rho.size[1]/2))
     for i in range(AB.size[0]):
          for l in range(AB.size[1]):
               AB[i,l] =complex(rho[i,l],rho[i+rho.size[0]/2,l])
     B = matrix(complex(0,0),(rho.size[0]/4,rho.size[1]/4))
     if a == 0:
          trAB = spdiag([1.0,1.0,0,0]) * AB * 2
     if a == 1:
          trAB = spdiag([0,0,1.0,1.0]) * AB * 2
     if a == 2:
          trAB = matrix
   ([[1.0,0,1.0,0],[0,1.0,0,1.0],[1.0,0,1.0,0],[0,1.0,0,1.0]]) * AB
     if a == 3:
          trAB = matrix
   ([[1.0,0,-1.0,0],[0,1.0,0,-1.0],[-1.0,0,1.0,0],[0,-1.0,0,1]]) * AB
     B[0,0] = trAB[0,0] + trAB[2,2]
     B[1,1] = trAB[1,1] + trAB[3,3]
     B[0,1] = trAB[0,1] + trAB[2,3]
     B[1,0] = trAB[1,0] + trAB[3,2]
     B_real = matrix(0.0, (rho.size[0]/2,rho.size[1]/2))
```

```
    for m in range(B.size[0]):
        for n in range(B.size[1]):
            B_real[m,n] = B[m,n].real
            B_real[m+B.size[0],n+B.size[1]] = B[m,n].real
            B_real[m+B.size[0],n] = B[m,n].imag
            B_real[m,n+B.size[0]] = -B[m,n].imag
    return B_real


# Berechnung der von-Neumann Entropie einer Dichtematrix

def S(AB):
    ew = 0.0
    ew = Eigenwerte(copy.copy(AB),0)
    val = 0.0
    for i in range(AB.size[0]):
        val += h(ew[i])
    return val

def DPhi(k,l):
    val = 0.0
    if (abs(k-l) > 1e-12):
        val = ((log(k) - log(l)) / (k - l))
    else:
        val = 1/k
    return val

def D2Phi(k,l):
    val = 0.0
    if (abs(k-l) > 1e-12):
        val = ((1/k - (log(l)-log(k))/(l-k)) / (k - l))
    else:
        val = -1/k**2
    return val

def eMe(k,A,l):
    val = 0.0
    val = (k.T * A * l)[0]
    return val

# 1.Ableitung der Entropie

def DS(AB,i,flag):
```

```
    [ew,ev] = Eigenwerte(copy.copy(AB),1)
    t=0.0
    if flag == 'F':
        for j in range(ew.size[0]):
            t += (1 + log(ew[j]))* eMe(ev[:,j],F[i],ev[:,j])
    if flag == 'G0':
        for j in range(ew.size[0]):
            t += (1 + log(ew[j]))* eMe(ev[:,j],G0[i],ev[:,j])
    if flag == 'G1':
        for j in range(ew.size[0]):
            t += (1 + log(ew[j]))* eMe(ev[:,j],G1[i],ev[:,j])
    if flag == 'G2':
        for j in range(ew.size[0]):
            t += (1 + log(ew[j]))* eMe(ev[:,j],G2[i],ev[:,j])
    if flag == 'G3':
        for j in range(ew.size[0]):
            t += (1 + log(ew[j]))* eMe(ev[:,j],G3[i],ev[:,j])
    return -t/log(2)


# 2. Ableitung der Entropie


def D2S(AB,i,j,flag):
    [ew,ev] = Eigenwerte(copy.copy(AB),1)
    summe = 0.0
    val = 0.0
    for k in range(ew.size[0]):
        for l in range(ew.size[0]):

            if flag == 'F':
                val = 2*eMe(ev[:,k],F[i],ev[:,l])*eMe(ev[:,l],F[j],ev[:,
k]) * (DPhi(ew[k],ew[l]))# + 0.5*D2Phi(ew[k],ew[l])* ew[k])

            if flag == 'G0':
                val = 2*eMe(ev[:,k],G0[i],ev[:,l])*eMe(ev[:,l],G0[j],ev
[:,k]) * (DPhi(ew[k],ew[l]))# + 0.5*D2Phi(ew[k],ew[l]) * ew[k])

            if flag == 'G1':
                val = 2*eMe(ev[:,k],G1[i],ev[:,l])*eMe(ev[:,l],G1[j],ev
[:,k]) * (DPhi(ew[k],ew[l]))# + 0.5*D2Phi(ew[k],ew[l]) * ew[k])

            if flag == 'G2':
```

```
                    val = 2*eMe(ev[:,k],G2[i],ev[:,l])*eMe(ev[:,l],G2[j],ev
    [:,k]) * (DPhi(ew[k],ew[l]))# + 0.5*D2Phi(ew[k],ew[l]) * ew[k])

                if flag == 'G3':
                    val = 2*eMe(ev[:,k],G3[i],ev[:,l])*eMe(ev[:,l],G3[j],ev
    [:,k]) * (DPhi(ew[k],ew[l]))# + 0.5*D2Phi(ew[k],ew[l]) * ew[k])

                summe += val
      return -summe/log(2)


# 1. Ableitung von Holevo

def DChi(AB):
     val = matrix(0.0,(len(F),1))
     for i in range(len(F)):
          val[i] =  DS(AB,i,'F')  - 0.25 * ( DS(cond(AB,0),i,'G0') + DS(cond
    (AB,1),i,'G1') + DS(cond(AB,1),i,'G2') + DS(cond(AB,1),i,'G3') )
      return val.T

# 2. Ableitung von Holevo

def D2Chi(AB):
     val = matrix(0.0,(len(F),len(F)))
     for i in range(val.size[0]):
          for j in range(val.size[0]):
               val[i,j] = D2S(AB,i,j,'F') - 0.25* ( D2S(cond(AB,0),i,j,'G0')
     + D2S(cond(AB,1),i,j,'G1') + D2S(cond(AB,1),i,j,'G2') + D2S(cond(AB,1),i
    ,j,'G3'))
     return val

# Funktion fuer den Solver

def CVXOpt(x = None, z = None):
     if x is None:
          return 0, x_start
     rho = define_rho(x)
     ew = Eigenwerte(copy.copy(rho),0)
     for i in range(rho.size[0]):
          if ew[i] < 1e-12:
               return None
```

```
    Chi = S(copy.copy(rho)) - 0.25 * ( S(cond(copy.copy(rho_start),0)) + S(
   cond(copy.copy(rho_start),1)) + S(cond(copy.copy(rho_start),2)) + S(cond(
   copy.copy(rho_start),3)))
    r = (I - 0.5 * Chi)
    Dr = - 0.5 * DChi(rho)
    if z is None:
        return r, Dr
    H = - 0.5 * z[0]  * D2Chi(rho)
    return r, Dr, H

# Matrizen der Paulidarstellung F, G0, G1 fuer die freien Parameter x
Pauli = []
F = []
G0 = []
G1 = []
G2 = []
G3 = []

#

Pauli.append(matrix([[0, 0, 0, -1], [0, 0, 1, 0], [0, 1, 0, 0], [-1, 0, 0,
   0]]))
Pauli.append(matrix([[0, +1j, 0, 0],[-1j, 0, 0, 0], [0, 0, 0, +1j], [0, 0,
   -1j, 0]]))
Pauli.append(matrix([[0, 0, 0, 1j],[0, 0, -1j, 0], [0, 1j, 0, 0],[-1j, 0, 0,
    0]]))
Pauli.append(matrix([[0, 0, +1j, 0],[0, 0, 0, -1j],[-1j, 0, 0, 0],[0, +1j,
   0, 0]]))
Pauli.append(matrix([[0, 1j, 0, 0],[-1j, 0, 0, 0], [0, 0, 0, -1j], [0, 0, 1j
   , 0]]))
Pauli.append(matrix([[0, 0, 0, 1j],[0, 0, 1j, 0], [0, -1j, 0, 0],[-1j, 0, 0,
    0]]))

for i in range(len(Pauli)):
    mat = matrix(0.0, (2*Pauli[0].size[0],2*Pauli[0].size[1]))
    for j in range(Pauli[0].size[0]):
        for k in range(Pauli[0].size[1]):
            mat[j,k] = Pauli[i][j,k].real
            mat[j+Pauli[0].size[0],k+Pauli[0].size[0]] = Pauli[i][j,k].
   real
            mat[j+Pauli[0].size[0],k] = Pauli[i][j,k].imag
            mat[j,k+Pauli[0].size[0]] = -Pauli[i][j,k].imag
    F.append(0.25*mat)
```

```
for a in range(len(F)):
     G0.append(cond(F[a],0))
     G1.append(cond(F[a],1))
     G2.append(cond(F[a],2))
     G3.append(cond(F[a],3))

xstart = None
xdata = []
ydata = []
zdata = []

I=0

r_ende = 1
qpoint = 0

qerror_z = 0.2
qerror_x = 0.01

while (r_ende > 0):

     # Bekannte Parameter

     print '-----------------------------------'
     print '--QBER X--'
     print qerror_x
     print '--QBER Z--'
     print qerror_z

     testew = 0

     rho_fix = 0.25*matrix([[2-2*qerror_z, 0, 0, 1-2*qerror_x, 0, 0, 0,
   0],[0, 2*qerror_z, 1-2*qerror_x, 0, 0, 0, 0, 0], [0, 1-2*qerror_x, 2*
   qerror_z, 0, 0, 0, 0, 0], [1-2*qerror_x, 0, 0, 2-2*qerror_z, 0, 0, 0, 0],
    [0, 0, 0, 0, 2-2*qerror_z, 0, 0, 1-2*qerror_x],[0, 0, 0, 0, 0, 2*
   qerror_z, 1-2*qerror_x, 0], [0, 0, 0, 0, 0, 1-2*qerror_x, 2*qerror_z, 0],
    [0, 0, 0, 0, 1-2*qerror_x, 0, 0, 2-2*qerror_z]])

     rho_start = None

     #Startpunkt
     x_start = matrix(1e-4,(len(Pauli),1))
```

```
  while ( testew == 0):
       testew = 1
       var = 0
       var += 1
       if qerror_z > qerror_x:
            x_start[0] = -(1-2*qerror_x) + qerror_z*(2-var*1e-1)
       else:
            x_start[0] = -(1-2*qerror_z) + qerror_x*(2-var*1e-1)
       rho_start = define_rho(x_start)
       ew = Eigenwerte(copy.copy(rho_start),0)
       for i in range(rho_start.size[0]):
            if ew[i] <= 0:
                 testew = 0

  print 'Startpunkt: '
  print x_start.T

  SDconstraint = matrix(0.0, (F[0].size[0]*F[0].size[1], x_start.size[0])
)
  for i in range(len(F)):
       SDconstraint[:,i] = - matrix([F[i][:,0], F[i][:,1], F[i][:,2], F[i
][:,3], F[i][:,4], F[i][:,5], F[i][:,6], F[i][:,7]])
  sdconstraint = matrix([rho_fix[:,0], rho_fix[:,1], rho_fix[:,2],
rho_fix[:,3], rho_fix[:,4], rho_fix[:,5], rho_fix[:,6], rho_fix[:,7]])

  dims = {'l': 0, 'q': [], 's': [8]}

  solvers.options['maxiters']= 100
  solvers.options['feastol'] = 1e-6
  solvers.options['show_progress']=True

  I = 1 + 0.5*(- h(qerror_z) - h(1-qerror_z) - h(qerror_x) - h(1-qerror_x
))
  Chi_start = S(copy.copy(rho_start)) - 0.25 * ( S(cond(copy.copy(
rho_start),0)) + S(cond(copy.copy(rho_start),1)) + S(cond(copy.copy(
rho_start),2)) + S(cond(copy.copy(rho_start),3)))
  r_start = I - 0.5 * Chi_start

  if(r_start > 0):
       sol = solvers.cp(CVXOpt,SDconstraint,sdconstraint,dims)
       x_end = sol['x']
       rho_end=(define_rho(x_end))
```

```
            Chi_end = S(copy.copy(rho_end)) - 0.25 * ( S(cond(copy.copy(
    rho_end),0)) + S(cond(copy.copy(rho_end),1)) + S(cond(copy.copy(rho_end)
    ,2)) + S(cond(copy.copy(rho_end),3)))
            print 'Schluesselrate r_Start: '
            print r_start
            print 'Schluesselrate r_Ende: '
            r_ende = I - 0.5*Chi_end
            print r_ende
      else:
            break

      if r_ende > 0:
            xdata.append(qerror_x)
            ydata.append(qerror_y)
            zdata.append(r_ende)

      if(qerror_z == qerror_x ):
            qerror_x += 0.01
            qerror_z = 0.01
      elif(qerror_z < qerror_x):
            c = 0
            c = qerror_z
            qerror_z = qerror_x
            qerror_x = c
      elif(qerror_x < qerror_z):
            c = 0
            c = qerror_x
            qerror_x = qerror_z
            qerror_z = c + 0.01


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
plt.title('Secret Key Rate of an asymmetric BB84')
plt.axis([0.0, 0.30, 0., 0.30])
ax.scatter(xdata, ydata, zdata, zdir='z', s=15, c='b')
ax.plot(xdata, ydata, zdata)
plt.xlabel('QBER in z')
plt.ylabel('QBER in r')
plt.zlabel('Key Rate r')
plt.grid(True)

fobj_out = open("xerr" + "0.2" ,"w")
```

```
for j in range(len(xdata)):
     fobj_out.write(str(xdata[j]) + " " + str(ydata[j]) + "\n")
fobj_out.close()

plt.show()
```

Quellcodeausschnitt B.1: Quellcode des Optimierungsprogramms für das BB84 mit verschiedenen Fehlerraten in x- und z-Richtung

Anhang C

Quellcode des Optimierungsprogramms für das 2-state Protokoll

In diesem Abschnitt ist das Programm zur Ermittlung der sicheren Schlüsselrate des 2-state Protokolls aus Kapitel 6.4 dargestellt.

```
import math
import numpy
import cvxopt
import cvxopt.lapack
from cvxopt import matrix, log, div, spdiag, solvers, spmatrix
import pdb
import copy
import matplotlib.pyplot as plt
from numpy import array



##############################################################################
# Funktionendefinionen                                                       #
##############################################################################


# Definition der zweiparteien Paulis
def define_Paulis(Pauli, Fex, G0, G1):

    Pauli.append(0.25*matrix([[0, 0, 0, -1], [0, 0, 1, 0], [0, 1, 0, 0],
   [-1, 0, 0, 0]]))
```

```
    Pauli.append(0.25*matrix([[0, 0, 1, 0], [0, 0, 0, -1], [1, 0, 0, 0],
[0, -1, 0, 0]]))
    Pauli.append(0.25*matrix([[0, 0, 0, 1], [0, 0, 1, 0], [0, 1, 0, 0], [1,
 0, 0, 0]]))
    Pauli.append(0.25*matrix([[0, +1j, 0, 0],[-1j, 0, 0, 0], [0, 0, 0, +1j
], [0, 0, -1j, 0]]))
    Pauli.append(0.25*matrix([[0, 0, 0, 1j],[0, 0, -1j, 0], [0, 1j, 0,
0],[-1j, 0, 0, 0]]))
    Pauli.append(0.25*matrix([[0, 0, 0, 1j],[0, 0, 1j, 0], [0, -1j, 0,
0],[-1j, 0, 0, 0]]))
    Pauli.append(0.25*matrix([[0, 0, +1j, 0],[0, 0, 0, -1j],[-1j, 0, 0,
0],[0, +1j, 0, 0]]))
    Pauli.append(0.25*matrix([[0, 1j, 0, 0],[-1j, 0, 0, 0], [0, 0, 0, -1j],
 [0, 0, 1j, 0]]))

    for i in range(len(Pauli)):
        mat = matrix(0.0, (2*Pauli[0].size[0],2*Pauli[0].size[1]))
        for j in range(Pauli[0].size[0]):
            for k in range(Pauli[0].size[1]):
                mat[j,k] = Pauli[i][j,k].real
                mat[j+Pauli[0].size[0],k+Pauli[0].size[0]] = Pauli[i][j,
k].real
                mat[j+Pauli[0].size[0],k] = Pauli[i][j,k].imag
                mat[j,k+Pauli[0].size[0]] = -Pauli[i][j,k].imag
        Fex.append(1/2.0*mat)

    for a in range(len(Pauli)):
        G0.append(cond(copy.copy(Pauli[a]),0))
        G1.append(cond(copy.copy(Pauli[a]),1))


# Definition der Dichtematrix

def define_rho(x):
    param = complex(0.0,0.0)
    for i in range(len(Pauli)):
        param += x[i]*Pauli[i]
    AB =  1/pF*F*(rho_fix + (param))*F.T
    return AB

def define_rhofix_ex():
    AB =  rho_fix
    mat = matrix(0.0, (2*Pauli[0].size[0],2*Pauli[0].size[1]))
```

```
    for j in range(Pauli[0].size[0]):
        for k in range(Pauli[0].size[1]):
            mat[j,k] = AB[j,k].real
            mat[j+Pauli[0].size[0],k+Pauli[0].size[0]] = AB[j,k].real
            mat[j+Pauli[0].size[0],k] = AB[j,k].imag
            mat[j,k+Pauli[0].size[0]] = -AB[j,k].imag
    return 1/2.0*mat

#Berechnung der Eigenwerte

def Eigenwerte(A,i):
    W = matrix(0.0 ,(A.size[0],1))
    EV = matrix(complex(0.0,0.0) ,A.size)
    if i==0:
        cvxopt.lapack.heevx(A, W, jobz = 'N')
        return W
    if i==1:
        cvxopt.lapack.heevx(A, W, jobz = 'V', range = 'A', Z = EV)
        return [W, EV]


# Binaere Entropie
def h(p):
    val = 0
    if p > 0:
        val = -p*(log(p)/log(2))
    return val

# Berechnung der auf das Resultat a knoditionierten Dichtematrizen

def cond(rho,a):
    AB = rho
    CondAB = matrix(complex(0,0),(rho.size[0]/2,rho.size[1]/2))

    if a == 0:
        trAB = spdiag([1.0,1.0,0,0]) * AB * spdiag([1.0,1.0,0,0]) * 2
    if a == 1:
        trAB = spdiag([0,0,1.0,1.0]) * AB * spdiag([0,0,1.0,1.0]) * 2

    CondAB[0,0] = trAB[0,0] + trAB[2,2]
    CondAB[1,1] = trAB[1,1] + trAB[3,3]
    CondAB[0,1] = trAB[0,1] + trAB[2,3]
    CondAB[1,0] = trAB[1,0] + trAB[3,2]
```

```
        return CondAB


# Berechnung der von-Neumann Entropie einer Dichtematrix

def S(AB):
        ew = 0.0
        ew = Eigenwerte(copy.copy(AB),0)
        val = 0.0
        for i in range(AB.size[0]):
                val += h(ew[i])
        return val

# Berechnung der ersten Ableitung der Primaerfkt
def DPhi(k,l):
        val = 0.0
        if (k!=l):
                val = (log(k) - log(l)) / (k - l)
        else:
                val = 1/k
        return val

# Berechnung der zweiten Ableitung der Primaerfkt
def D2Phi(k,l):
        val = 0.0
        if ( abs(k - l) > 1e-12):
                val = ((1/k - (log(l)-log(k))/(l-k)) / (k - l))
        else:
                val = -1/(k**2)
        return val

# Uebergangswkt
def eMe(k,A,l):
        val = 0.0
        val = (k.H * A * l)[0]
        return val

# 1. Ableitung der v.N.-Entropie

def DS(AB,i,flag):
        t=0.0
        [ew, ev] = Eigenwerte(copy.copy(AB),1)
```

```
    if flag == 'F':
        for j in range(ew.size[0]):
            t += (1 + log(ew[j]))* eMe(ev[:,j], FPauli[i], ev[:,j])

    elif flag == 'G0':
        for j in range(ew.size[0]):
           t += (1 + log(ew[j]))* eMe(ev[:,j], FG0[i], ev[:,j])

    elif flag == 'G1':
        for j in range(ew.size[0]):
            t += (1 + log(ew[j]))* eMe(ev[:,j], FG1[i], ev[:,j])
    return (-t.real/log(2))


# 2. Ableitung der v.N.-Entropie


def D2S(AB,i,j,flag):

    summe = 0.0
    [ew, ev] = Eigenwerte(copy.copy(AB),1)

    if flag == 'F':
        for k in range(ew.size[0]):
            for l in range(ew.size[0]):
                val = (eMe(ev[:,k],FPauli[i],ev[:,l])*eMe(ev[:,l],FPauli
[j],ev[:,k]) + eMe(ev[:,l],FPauli[i],ev[:,k])*eMe(ev[:,k],FPauli[j],ev[:,
l])) *(DPhi(ew[k],ew[l]))# +  0.0 *D2Phi(ew[k],ew[l])* ew[k])
                summe += val

    if flag == 'G0':
        for k in range(ew.size[0]):
            for l in range(ew.size[0]):
                val =  (eMe(ev[:,k],FG0[i],ev[:,l])*eMe(ev[:,l],FG0[j],
ev[:,k]) + eMe(ev[:,l],FG0[i],ev[:,k])*eMe(ev[:,k],FG0[j],ev[:,l])) *  (
DPhi(ew[k],ew[l]))# + 0.0 * D2Phi(ew[k],ew[l]) * ew[k])
                summe += val

    if flag == 'G1':
        for k in range(ew.size[0]):
            for l in range(ew.size[0]):
```

```
                    val =  (eMe(ev[:,k],FG1[i],ev[:,l])*eMe(ev[:,l],FG1[j],
    ev[:,k]) + eMe(ev[:,l],FG1[i],ev[:,k])*eMe(ev[:,k],FG1[j],ev[:,l])) *  (
    DPhi(ew[k],ew[l]))# + 0.0 * D2Phi(ew[k],ew[l]) * ew[k])
                    summe += val

      return (-summe.real/log(2))


# 1. Ableitung von Holevo

def DChi(AB):
      val = matrix(0.0,(len(Pauli),1))
      for i in range(len(Pauli)):
            val[i,0] =  DS(AB,i,'F')  - 0.5 * ( DS(cond(AB,0),i,'G0') + DS(
    cond(AB,1),i,'G1'))
      return val.T

# 2. Ableitung von Holevo

def D2Chi(AB):
      val = matrix(0.0,(len(Pauli),len(Pauli)))
      for i in range(val.size[0]):
            for j in range(val.size[0]):
                  val[i,j] = D2S(AB,i,j,'F') - 0.5 * ( D2S(cond(AB,0),i,j,'G0')
     + D2S(cond(AB,1),i,j,'G1'))
      return val

# Funktion fuer den Solver

def CVXOpt(x = None, z = None):
      if x is None:
            return 0, x_start

      rho = define_rho(x)

      [ew, ev] = Eigenwerte(copy.copy(rho),1)

      for i in range(rho.size[0]):
            if ew[i] <= 0:
                  return None

      Chi = S(copy.copy(rho)) - 0.5 * ( S(cond(rho,0)) + S(cond(rho,1)) )
      r = (I - Chi)
```

```
    Dr =  - DChi(rho)

    if z is None:
        return r, Dr
    H =  - z[0] * D2Chi(rho)

    return r, Dr, H

##############################################################################
# Matrizen der Paulidarstellung Pauli, G0, G1 fuer die freien Parameter x #
#                                                                            #
##############################################################################

Pauli = [] # Liste der Zweiparteien-Paulis
Fex = []   # Erweiterung auf reelle symmetrische Matrizen
G0 = []    # Zweiparteien-Pauli auf das Ergebnis 0 von Alice konditioniert
G1 = []    # Zweiparteien-Pauli auf das Ergebnis 1 von Alice konditioniert
G2 = []
G3 = []

define_Paulis(Pauli, Fex, G0, G1)


##############################################################################
# Initalisierung der Variablen                                               #
##############################################################################

x_start = matrix(0.0,(len(Pauli),1))
I=0
r_ende = 0.0
pF = 0.0

xdata = []
ydata = []

epsilon_matrix = matrix(0.0,(64,1))
epsilon = 1e-5
for i in range(Fex[0].size[0]):
    epsilon_matrix[i*(1+Fex[0].size[0]),0] = epsilon



##############################################################################
```

```
# Startparameter                                                              #
###############################################################################

alpha = 0.1
beta = math.sqrt(1-alpha**2)

p=1e-3

F = matrix([[alpha, alpha, 0, 0], [beta, -beta, 0, 0], [0, 0, alpha, alpha],
    [0, 0, beta, -beta]])
FBob = matrix([[alpha,alpha], [beta, -beta]])

###############################################################################
# Berechnungsschleife                                                         #
###############################################################################

while r_ende >= 0:
     # Bekannte Parameter
     pF = 4*(1-p)*alpha**2*beta**2 + p
     qerror = p/(2*pF)

     print '----------------------------------'
     print '--p_F--'
     print pF

     print '--p--'
     print p

     print '--QBER--'
     print qerror

     p0x = 0.5
     p0y = 0.5

     a = complex(0.5*(1-p)*beta*beta + 0.25*p,0)
     c = complex(0.5*(1-p)*alpha*alpha + 0.25*p,0)
     b = complex(0.5*(1-p)*alpha*beta,0)
     d = complex(0.25*(beta*beta - alpha*alpha),0)

     rho_fix = matrix([[a, b, d, 0], [b, c, 0, d], [d, 0, a, -b], [0, d, -b,
    c]])

     FPauli = []
```

```
  FG0 = []
  FG1 = []

  for a in range(len(Pauli)):
       FPauli.append(1/pF*F*Pauli[a]*F.T)
       FG0.append(1/pF*FBob*G0[a]*FBob.T)
       FG1.append(1/pF*FBob*G1[a]*FBob.T)

  #Startpunkt
  x_start[0] = 4*0.5*(1-p)*alpha*beta
  x_start[1] = (1-p)

  rhofix_ex = define_rhofix_ex()

  rho_start = define_rho(x_start)

  SDconstraint = matrix(0.0, (Fex[0].size[0]*Fex[0].size[1], x_start.size
[0]))
  for i in range(len(Fex)):
       SDconstraint[:,i] =  - matrix([Fex[i][:,0], Fex[i][:,1], Fex[i
][:,2], Fex[i][:,3], Fex[i][:,4], Fex[i][:,5], Fex[i][:,6], Fex[i][:,7]])
  sdconstraint = matrix([rhofix_ex[:,0], rhofix_ex[:,1], rhofix_ex[:,2],
rhofix_ex[:,3], rhofix_ex[:,4], rhofix_ex[:,5], rhofix_ex[:,6], rhofix_ex
[:,7]])
  dims = {'l': 0, 'q': [], 's': [8]}

  solvers.options['maxiters'] = 500
  #solvers.options['feastol'] = 1e-10
  solvers.options['refinement'] = 1
  solvers.options['show_progress'] = False
  try:
       sol = solvers.cp(CVXOpt, SDconstraint, sdconstraint, dims)
       x_end = sol['x']

       rho_ende = define_rho(x_end)

       I = 1 - h(qerror) - h(1-qerror)

       Chi_start = S(copy.copy(rho_start)) - 0.5 *(S(cond(copy.copy(
rho_start),0)) + S(cond(copy.copy(rho_start),1)))
       Chi_ende = S(copy.copy(rho_ende))- 0.5 * (S(cond(copy.copy(
rho_ende),0)) + S(cond(copy.copy(rho_ende),1)))
```

```
            print 'Schluesselrate r_Start: '
            r_start = ( I - Chi_start )
            print r_start

            print 'Schluesselrate r_Ende: '
            r_ende = ( I - Chi_ende )
            print r_ende

            if r_ende <= 0:
                 break

            xdata.append(p)
            ydata.append(r_ende)

       except (ValueError, TypeError):
            print "Fehlerhafter Punkt"
       p += 1e-3

# Plotten der Daten

plt.title('Secret Key Rate of the BB84 (alpha= ' +str(alpha) + ')' )
plt.axis([0.0, 0.1, 0., 1.0])
plt.plot(xdata, ydata, 'ro')
plt.xlabel('p')
plt.ylabel('Key Rate r')
plt.grid(True)
plt.show()

#Speichern in .txt

fobj_out = open("B92_alpha"+str(alpha),"w")
for i in range(len(xdata)):
     fobj_out.write(str(xdata[i]) + ' ' +  str(ydata[i]) + "\n" )
fobj_out.close()
```

Quellcodeausschnitt C.1: Quellcode des Optimierungsprogramms für das 2-state Protokoll

Anhang D

Quellcode des Optimierungsprogramms für das 2-state Protokoll mit Kanalverlusten

In diesem Kapitel befindet sich das vollständige Programm zur Ermittlung der Schlüsselrate des 2-state Protokolls mit Kanalverlusten aus Kapitel 6.5.

```
import math
import numpy
import cvxopt
import cvxopt.lapack
from cvxopt import matrix, log, div, spdiag, solvers, spmatrix
import pdb
import copy
import matplotlib.pyplot as plt
from numpy import array



#############################################################################
# Funktionendefinionen                                                      #
#############################################################################


# Definition der zweiparteien Pauli Gell-Mann Basis
def define_Paulis(Pauli, Fex, G0, G1):
    #---Startpunkte---#
```

```
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, -1, 0], [0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [-1, 0, 0, 0, 0, 0], [0, 0, 0, 0,
 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 1, 0, 0], [0, 0, 0, 0, -1, 0],
[0, 0, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0], [0, -1, 0, 0, 0, 0], [0, 0, 0, 0,
 0, 0]]))
    Pauli.append(1/(4.*math.sqrt(3.0))*matrix([[0, 0, 0, 1, 0, 0], [0,
0, 0, 0, 1, 0], [0, 0, 0, 0, 0, -2], [1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0,
0], [ 0, 0, -2, 0, 0, 0]]))

    #--4 bis 10--#
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 1, 0], [0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0], [0, 0, 0, 0,
0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 1j, 0], [0, 0, 0, 1j, 0, 0],
[0, 0, 0, 0, 0, 0], [0, -1j, 0, 0, 0, 0], [-1j, 0, 0, 0, 0, 0], [0, 0, 0,
 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 1j, 0, 0, 0, 0], [-1j, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1j, 0], [0, 0, 0, -1j, 0, 0], [0, 0, 0,
 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 1j, 0], [0, 0, 0, -1j, 0, 0],
 [0, 0, 0, 0, 0, 0], [0, 1j, 0, 0, 0, 0], [-1j, 0, 0, 0, 0, 0], [0, 0, 0,
 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 1j, 0, 0, 0, 0], [-1j, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, -1j, 0], [0, 0, 0, 1j, 0, 0], [0, 0, 0,
 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 1j, 0, 0], [0, 0, 0, 0, -1j, 0],
 [0, 0, 0, 0, 0, 0], [-1j, 0, 0, 0, 0, 0], [0, 1j, 0, 0, 0, 0], [0, 0, 0,
 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0], [0, 0, 0, 1,
0, 0]]))

    #--11 bis 20--#
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0], [1, 0, 0, 0,
0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 1j], [0, 0, 0, 0, 0, 0],
[0, 0, 0, 1j, 0, 0], [0, 0, -1j, 0, 0, 0], [0, 0, 0, 0, 0, 0], [-1j, 0,
0, 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, -1], [0, 0, 0, 0, 0, 0], [0, 0, 0,
-1, 0, 0]]))
```

```
    Pauli.append(1/4.*matrix([[0, 0, 1j, 0, 0, 0], [0, 0, 0, 0, 0, 0],
[-1j, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1j], [0, 0, 0, 0, 0, 0], [0, 0, 0,
-1j, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 1j], [0, 0, 0, 0, 0, 0],
[0, 0, 0, -1j, 0, 0], [0, 0, 1j, 0, 0, 0], [0, 0, 0, 0, 0, 0], [-1j, 0,
0, 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, -1], [0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0], [-1, 0, 0, 0,
 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 1j, 0, 0, 0], [0, 0, 0, 0, 0, 0],
[-1j, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, -1j], [0, 0, 0, 0, 0, 0], [0, 0, 0,
 1j, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0],
[0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1], [ 0, 0, 0, 0,
 1, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1],
[0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 1, 0, 0,
0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1j],
[0, 0, 0, 0, 1j, 0], [0, 0, 0, 0, 0, 0], [0, 0, -1j, 0, 0, 0], [0, -1j,
0, 0, 0, 0]]))

    #--21 bis 26--#
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0],
[0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, -1], [ 0, 0, 0,
0, -1, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 1j, 0, 0, 0],
[0, -1j, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1j], [ 0, 0, 0,
 0, -1j, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1j],
[0, 0, 0, 0, -1j, 0], [0, 0, 0, 0, 0, 0], [0, 0, 1j, 0, 0, 0], [0, -1j,
0, 0, 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, -1],
[0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, -1, 0, 0,
 0, 0]]))
    Pauli.append(1/4.*matrix([[0, 0, 0, 0, 0, 0], [0, 0, 1j, 0, 0, 0],
[0, -1j, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, -1j], [ 0, 0,
0, 0, 1j, 0]]))
    Pauli.append(1/(4.*math.sqrt(3.0))*matrix([[0, 0, 0, 1j, 0, 0], [0,
0, 0, 0, 1j, 0], [0, 0, 0, 0, 0, -2j], [-1j, 0, 0, 0, 0, 0], [0, -1j, 0,
0, 0, 0], [ 0, 0, 2j, 0, 0, 0]]))

    for i in range(len(Pauli)):
```

```
          mat = matrix(0.0, (2*Pauli[0].size[0],2*Pauli[0].size[1]))
          for j in range(Pauli[0].size[0]):
               for k in range(Pauli[0].size[1]):
                    mat[j,k] = Pauli[i][j,k].real
                    mat[j+Pauli[0].size[0],k+Pauli[0].size[0]] = Pauli[i][j,
   k].real
                    mat[j+Pauli[0].size[0],k] = Pauli[i][j,k].imag
                    mat[j,k+Pauli[0].size[0]] = -Pauli[i][j,k].imag
          Fex.append(mat)

        for a in range(len(Pauli)):
          G0.append(cond(copy.copy(Pauli[a]),0))
          G1.append(cond(copy.copy(Pauli[a]),1))



# Definition der Dichtematrix

def define_rho(x):
     param = complex(0.0,0.0)
     for i in range(len(Pauli)):
          param += x[i]*Pauli[i]
     AB =  1/pF*F*(rho_fix + (param))*F.T
     return AB

def define_rho_uf(x):
     param = complex(0.0,0.0)
     for i in range(len(Pauli)):
          param += x[i]*Pauli[i]
     AB =  rho_fix + (param)
     return AB

def define_rhofix_ex():
     AB =  rho_fix
     mat = matrix(0.0, (2*Pauli[0].size[0],2*Pauli[0].size[1]))
     for j in range(Pauli[0].size[0]):
          for k in range(Pauli[0].size[1]):
               mat[j,k] = AB[j,k].real
               mat[j+Pauli[0].size[0],k+Pauli[0].size[0]] = AB[j,k].real
               mat[j+Pauli[0].size[0],k] = AB[j,k].imag
               mat[j,k+Pauli[0].size[0]] = -AB[j,k].imag
     return mat
```

```
#Berechnung der Eigenwerte

def Eigenwerte(A,i):

     W = matrix(0.0 ,(A.size[0] ,1))
     EV = matrix(complex(0.0,0.0) ,A.size)
     if i==0:
          cvxopt.lapack.heevx(A, W, jobz = 'N', range = 'A')
          return W
     if i==1:
          cvxopt.lapack.heevx(A, W, jobz = 'V', range = 'A', Z = EV)
          return [W, EV]


# Binaere Entropie
def h(p):
     val = 0
     if p > 0:
          val = -p*(log(p)/log(2))
     return val


# Berchnung der auf das Resultat a knoditionierten Dichtematrizen

def cond(rho,a):
     AB = rho
     CondAB = matrix(complex(0,0),(rho.size[0]/2,rho.size[1]/2))
     if rho.size[0] == 6:
          if a == 0:
               trAB = 2.0 * spdiag([1.0, 1.0, 1.0, 0, 0, 0]) * AB
          if a == 1:
               trAB = 2.0 * spdiag([0, 0, 0, 1.0, 1.0, 1.0]) * AB

          CondAB[0,0] = trAB[0,0] + trAB[3,3]
          CondAB[1,1] = trAB[1,1] + trAB[4,4]
          CondAB[2,2] = trAB[2,2] + trAB[5,5]

          CondAB[0,1] = trAB[0,1] + trAB[3,4]
          CondAB[1,2] = trAB[1,2] + trAB[4,5]
          CondAB[0,2] = trAB[0,2] + trAB[3,5]

          CondAB[1,0] = trAB[1,0] + trAB[4,3]
          CondAB[2,1] = trAB[2,1] + trAB[5,4]
```

```
            CondAB[2,0] = trAB[2,0] + trAB[5,3]

      elif rho.size[0] == 4:
            if a == 0:
                  trAB = 2.0 * spdiag([1.0, 1.0, 0, 0]) * AB
            if a == 1:
                  trAB = 2.0 * spdiag([0, 0, 1.0, 1.0]) * AB

            CondAB[0,0] = trAB[0,0] + trAB[2,2]
            CondAB[1,1] = trAB[1,1] + trAB[3,3]

            CondAB[0,1] = trAB[0,1] + trAB[2,3]
            CondAB[1,0] = trAB[1,0] + trAB[3,2]
      return CondAB


# Berechnung der von-Neumann Entropie einer Dichtematrix

def S(AB):
      ew = 0.0
      ew = Eigenwerte(copy.copy(AB),0)
      val = 0.0
      for i in range(AB.size[0]):
            val += h(ew[i])
      return val

# Berechnung der ersten Ableitung der Primaerfkt
def DPhi(k,l):
      val = 0.0
      if k!=l:
            val = (log(k) - log(l)) / (k - l)
      else:
            val = 1/k
      return val

# Uebergangswkt
def eMe(k,A,l):
      val = 0.0
      val = (k.H * A * l)[0]
      return val

# 1. Ableitung der v.N.-Entropie
```

```
def DS(AB,i,flag):
     t=0.0
     [ew, ev] = Eigenwerte(copy.copy(AB),1)
     if flag == 'F':
          for j in range(ew.size[0]):
               t += log(ew[j])* eMe(ev[:,j], FPauli[i], ev[:,j])
     elif flag == 'G0':
          for j in range(ew.size[0]):
               t += log(ew[j])* eMe(ev[:,j], FG0[i], ev[:,j])
     elif flag == 'G1':
          for j in range(ew.size[0]):
               t += log(ew[j])* eMe(ev[:,j], FG1[i], ev[:,j])
     return (-t.real/log(2.0))


# 2. Ableitung der v.N.-Entropie


def D2S(AB,i,j,flag):

     summe = 0.0
     [ew, ev] = Eigenwerte(copy.copy(AB),1)

     if flag == 'F':
          for k in range(ew.size[0]):
               for l in range(ew.size[0]):
                    val = (eMe(ev[:,k],FPauli[i],ev[:,l])*eMe(ev[:,l],FPauli
   [j],ev[:,k]))*DPhi(ew[k],ew[l])
                    summe += val

     elif flag == 'G0':
          for k in range(ew.size[0]):
               for l in range(ew.size[0]):
                    val = (eMe(ev[:,k],FG0[i],ev[:,l])*eMe(ev[:,l],FG0[j],ev
   [:,k]))*DPhi(ew[k],ew[l])
                    summe += val

     elif flag == 'G1':
          for k in range(ew.size[0]):
               for l in range(ew.size[0]):
                    val = (eMe(ev[:,k],FG1[i],ev[:,l])*eMe(ev[:,l],FG1[j],ev
   [:,k]))*DPhi(ew[k],ew[l])
                    summe += val
```

```
     return (-summe.real/log(2.0))



# 1. Ableitung von Holevo

def DChi(AB):
     val = matrix(0.0,(len(Pauli),1))
     for i in range(len(Pauli)):
          val[i,0] =  DS(AB,i,'F')  - 0.5 * ( DS(cond(AB,0),i,'G0') + DS(
   cond(AB,1),i,'G1'))
     return val.T

# 2. Ableitung von Holevo

def D2Chi(AB):
     val = matrix(0.0,(len(Pauli),len(Pauli)))
     for i in range(val.size[0]):
          for j in range(val.size[0]):
               val[i,j] = D2S(AB,i,j,'F') - 0.5 * ( D2S(cond(AB,0),i,j,'G0')
    + D2S(cond(AB,1),i,j,'G1'))
     return val

# Funktion fuer den Solver

def CVXOpt(x = None, z = None):
     if x is None:
          return 0, x_start

     rho = define_rho(x)
     rho_uf = define_rho_uf(x)

     [ew, ev] = Eigenwerte(copy.copy(rho),1)
     #[ew_uf, ev_uf] =Eigenwerte(copy.copy(rho_uf),1)
     #print ew
     for i in range(rho.size[0]):
          if ew[i] < 0:
               return None

     Chi = S(rho) - 0.5 * ( S(cond(rho,0)) + S(cond(rho,1)))
     r = (I - Chi)*100
     Dr =  - DChi(rho)*100
```

```
    if z is None:
        return r, Dr
    H =  - z[0] * D2Chi(rho)*100

    del rho, rho_uf, [ew, ev], Chi
    return r, Dr, H



##############################################################################
# Matrizen der Paulidarstellung Pauli, G0, G1 fuer die freien Parameter x #
#                                                                            #
##############################################################################

Pauli = [] # Liste der Zweiparteien-Paulis
Fex = []   # Erweiterung auf reelle symmetrische Matrizen
G0 = []    # Zweiparteien-Pauli auf das Ergebnis 0 von Alice konditioniert
G1 = []    # Zweiparteien-Pauli auf das Ergebnis 1 von Alice konditioniert
G2 = []
G3 = []

define_Paulis(Pauli, Fex, G0, G1)


##############################################################################
# Initalisierung der Variablen                                               #
##############################################################################

x_start = matrix(0.0, (len(Pauli),1))
I=0
r_ende = 0.0
pF = 0.0
L= 0.0

xdata = []
ydata = []

epsilon_matrix = matrix(0.0,(144,1))
epsilon = 0
for i in range(Fex[0].size[0]):
    epsilon_matrix[(i*Fex[0].size[0]+i),0] = epsilon
```

```
###########################################################################
# Startparameter                                                          #
###########################################################################

alpha = 0.38
beta = math.sqrt(1-alpha**2)
p= 0.075
L = 0.2
F = matrix([[alpha, alpha, 0, 0, 0, 0], [beta, -beta, 0, 0, 0, 0], [0, 0, 1,
    0, 0, 0], [0, 0, 0, alpha, alpha, 0], [0,  0, 0, beta, -beta, 0], [0, 0,
    0, 0, 0, 1]])
FBob = matrix([[alpha, alpha, 0], [beta, -beta, 0], [0, 0, 1]])

#Oeffnen der Ausgabedatei
fobj_out = open("B92_alpha_"+str(alpha) +"_L_"+str(L),"w")


###########################################################################
# Berechnungsschleife                                                     #
###########################################################################

while p>0.07:

     #Solever Einstellung
     solvers.options['maxiters'] = 100000
     solvers.options['show_progress'] = True
     dims = {'l': 0, 'q': [], 's': [12]}

     #Startpunkt
     x_start[0] = 2*(1-p)*(1-L)*alpha*beta
     x_start[1] = (1-p)*(1-L)
     x_start[2] = 1/math.sqrt(3.0)*(beta**2-alpha**2)*(1-3*L)

     # Bekannte Parameter
     gamma = (1-L)*(2*(1-p)*alpha**2*beta**2 + 0.5*p + 0.5*L/(1-L))
     pF = 2*gamma
     a = 1/6.0*complex(1,0)
     b = 1/4.0*complex((1-L)*(1-p)*(beta**2-alpha**2),0)
     c = 1/12.0*complex((1-3*L),0)
     d = 1/4.0*complex(2*(1-L)*(1-p)*alpha*beta,0)
     e = 1/6.0*complex(beta**2-alpha**2,0)
```

```
  rho_fix = matrix([[a+b+c, d, 0, e, 0, 0], [d, a-b+c, 0, 0, e, 0], [0,
0, a-2*c, 0, 0, e], [e, 0, 0, a+b+c, -d, 0], [0, e, 0, -d, a-b+c, 0], [0,
 0, e, 0, 0, a-2*c]])
  rhofix_ex = define_rhofix_ex()
  rho_start = define_rho(x_start)
  qerror = rho_start[1,1].real*2.0

  #Intialisierung und Bestimmung der gefilterten Paulis
  FPauli = []
  FG0 = []
  FG1 = []

  for a in range(len(Pauli)):
       FPauli.append(1/pF*F*Pauli[a]*F.T)
       FG0.append(1/pF*FBob*G0[a]*FBob.T)
       FG1.append(1/pF*FBob*G1[a]*FBob.T)

  #Textausgabe auf dem Bildschirm
  print '-----------------------------------'
  print '--p_F--'
  print pF
  print '--p--'
  print p
  print '--QBER--'
  print qerror

  SDconstraint = matrix(0.0, (Fex[0].size[0]*Fex[0].size[1], x_start.size
[0]))
  for i in range(len(Fex)):
       SDconstraint[:,i] =  - matrix([Fex[i][:,0], Fex[i][:,1], Fex[i
][:,2], Fex[i][:,3], Fex[i][:,4], Fex[i][:,5], Fex[i][:,6], Fex[i][:,7],
Fex[i][:,8],  Fex[i][:,9], Fex[i][:,10], Fex[i][:,11] ])
  sdconstraint = matrix([rhofix_ex[:,0], rhofix_ex[:,1], rhofix_ex[:,2],
rhofix_ex[:,3], rhofix_ex[:,4], rhofix_ex[:,5], rhofix_ex[:,6], rhofix_ex
[:,7], rhofix_ex[:,8], rhofix_ex[:,9], rhofix_ex[:,10], rhofix_ex[:,11]
]) - epsilon_matrix

  try:
       I = (1-L)*(1 - h(qerror) - h(1-qerror))

       #Solver
       sol = solvers.cp(CVXOpt, SDconstraint, sdconstraint, dims)
       x_end = sol['x']
```

```
            rho_ende = define_rho(x_end)
            Chi_start = S(copy.copy(rho_start)) - 0.5 *(S(cond(copy.copy(
    rho_start),0)) + S(cond(copy.copy(rho_start),1)))
            Chi_ende = S(copy.copy(rho_ende)) - 0.5 * (S(cond(copy.copy(
    rho_ende),0)) + S(cond(copy.copy(rho_ende),1)))

            #Bildschirmausgabe
            print 'Schluesselrate r_Start: '
            r_start = ( I - Chi_start )
            print r_start
            print 'Schluesselrate r_Ende: '
            r_ende = ( I - Chi_ende )
            print r_ende

            xdata.append(p)
            ydata.append(r_ende)

            fobj_out.write(str(xdata[-1]) + ' ' +  str(ydata[-1]) + "\n" )

        except (IOError, ValueError, TypeError, ZeroDivisionError):
            print "Fehlerhafter Punkt"

        del FPauli, FG0, FG1, sol
        p -= 0.001

fobj_out.close()

# Plotten der Daten

plt.title('Secret Key Rate of the B92 with lossy channel (alpha=' +str(alpha
    ) + 'L='+str(L) + ')' )
plt.axis([0.0, 0.1, 0., 1.0])
plt.plot(xdata, ydata, 'ro')
plt.xlabel('p')
plt.ylabel('Key Rate r')
plt.grid(True)
plt.show()
```

Quellcodeausschnitt D.1: Quellcode des Optimierungsprogramms für das 2-state Protokoll mit Kanalverlusten

Anhang E

Basis der SU(2)⊗SU(3)

Es folgt die Basis der Parametrisierung der Dichtematrix aus Kapitel 6.5.

$$F_{01} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad F_{02} = \begin{pmatrix} 0 & -i & 0 & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -i & 0 \\ 0 & 0 & 0 & i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$F_{03} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad F_{04} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$F_{05} = \begin{pmatrix} 0 & 0 & -i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -i \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & i & 0 & 0 \end{pmatrix} \qquad F_{06} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$F_{07} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -i \\ 0 & 1 & 0 & 0 & -i & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & i & 0 & 0 & 0 \\ 0 & i & 0 & 0 & 0 & 0 \end{pmatrix} \qquad F_{08} = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 \end{pmatrix}$$

$$F_{x0} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \qquad F_{x1} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$F_{x2} = \begin{pmatrix} 0 & 0 & 0 & 0 & -i & 0 \\ 0 & 0 & 0 & i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -i & 0 & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad F_{x3} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$F_{x4} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad F_{x5} = \begin{pmatrix} 0 & 0 & -i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & i \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -i & 0 & 0 \end{pmatrix}$$

$$F_{x6} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad F_{x7} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -i \\ 0 & 0 & 0 & 0 & i & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -i & 0 & 0 & 0 \\ 0 & i & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$F_{x8} = \frac{1}{\sqrt{3}} \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 \end{pmatrix} \qquad F_{y0} = \begin{pmatrix} 0 & 0 & 0 & -i & 0 & 0 \\ 0 & 0 & 0 & 0 & -i & 0 \\ 0 & 0 & 0 & 0 & 0 & -i \\ i & 0 & 0 & 0 & 0 & 0 \\ 0 & i & 0 & 0 & 0 & 0 \\ 0 & 0 & i & 0 & 0 & 0 \end{pmatrix}$$

$$F_{y1} = \begin{pmatrix} 0 & 0 & 0 & 0 & -i & 0 \\ 0 & 0 & 0 & -i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & i & 0 & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad F_{y2} = \begin{pmatrix} 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$F_{y3} = \begin{pmatrix} 0 & 0 & 0 & -i & 0 & 0 \\ 0 & 0 & 0 & 0 & i & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 \\ 0 & -i & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad F_{y4} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & -i \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -i & 0 & 0 \\ 0 & 0 & i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$F_{y5} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad F_{y6} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -i & 0 \\ 0 & 0 & 0 & -i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & i & 0 & 0 & 0 \\ 0 & i & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$F_{y7} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad F_{y8} = \frac{1}{\sqrt{3}} \begin{pmatrix} 0 & 0 & 0 & -i & 0 & 0 \\ 0 & 0 & 0 & 0 & -i & 0 \\ 0 & 0 & 0 & 0 & 0 & 2i \\ i & 0 & 0 & 0 & 0 & 0 \\ 0 & i & 0 & 0 & 0 & 0 \\ 0 & 0 & -2i & 0 & 0 & 0 \end{pmatrix}$$

$$F_{z0} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \qquad F_{z1} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$F_{z2} = \begin{pmatrix} 0 & -i & 0 & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & i & 0 \\ 0 & 0 & 0 & -i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad F_{z3} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$F_{z4} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \end{pmatrix} \qquad F_{z5} = \begin{pmatrix} 0 & 0 & -i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & i \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -i & 0 & 0 \end{pmatrix}$$

$$F_{z6} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \qquad F_{z7} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -i & 0 & 0 & 0 \\ 0 & i & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & i \\ 0 & 0 & 0 & 0 & -i & 0 \end{pmatrix}$$

$$F_{z8} = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

Literaturverzeichnis

[BV04] Boyd, Stephen ; Vandenberghe, Lieven: *Convex Optimization*. Cambridge, 2004

[CT91] Cover, Thomas M. ; Thomas, Joy A.: *Elements of Information Theory*. Wiley, 1991

[Dav78] Davies, E.: Information and quantum measurement. In: *IEEE Information Theory* 24 (1978), Sep, S. 596

[DW05] Devetak, I. ; Winter, A.: Distillation of secret key and entanglement from quantum states. In: *Proc. of the Roy. Soc. of London Series A* 461 (2005), Januar, S. 207–235

[EBP] E. Brüning, A. M. H. Mäkelä M. H. Mäkelä ; Petruccione, F.: Parametrizations of density matrices.

[FL12] Ferenczi, Agnes ; Lütkenhaus, Norbert: Symmetries in quantum key distribution and the connection between optimal attacks and optimal cloning. In: *Phys. Rev. A* 85 (2012), May, S. 052310

[GK02] Geiger, Carl ; Kanzow, Christian: *Theorie und Numerik restringierter Optimierungsaufgaben*. Springer, 2002

[HJ91] Horn, Roger A. ; Johnson, Charles R.: *Topics In Matrix Analysis*. Cambridge University Press, 1991

[MAV14] Martin Andersen, Joachim D. ; Vandenberghe, Lieven: *CVXOPT*. http://cvxopt.org. Version: 2014

[MCE] Matthias Christandl, Renato R. ; Ekert, Artur: A Generic Security Proof for Quantum Key Distribution.

[Nai40] Naimark, M. A.: Spectral functions of a symmetric operator. In: *Izv. Akad. Nauk SSSR, Ser. Mat.* 4 (1940)

[NC00] Nielsen, Michael A. ; Chuang, Isaac L.: *Quantum Computation and Quantum Information*. Cambridge, 2000

[RGK05] Renner, Renato ; Gisin, Nicolas ; Kraus, Babara: An information-theoretic security proof for quantum-key-distribution protocols. In: *Phys. Rev. A* 72 (2005), Jul, S. 012332

[TL] Tamaki, Kiyoshi ; Lütkenhaus, Norbert: Unconditional Security of the Bennett 1992 quantum key-distribution over lossy and noisy channel.

Erklärung

Hiermit erkläre ich, Florian Köppen, dass ich die vorliegende Master-Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate und Ergebnisse Anderer kenntlich gemacht habe.

Hilchenbach, den 29. März 2017

Gezeichnet: _____