Duale Hochschule Baden-Württemberg Mannheim

Studienarbeit

Konstruktion eines kryptographischen Verfahrens basierend auf Reed-Solomon-Codes und Diskussion möglicher Angriffsmethoden

Studiengang Informatik

Studienrichtung Cyber Security

Verfasser:	Roman Wetenkamp
Matrikelnummer:	5533869
Kurs:	TINF20CS1
Studiengangsleiter:	Prof. Dr. Konstantin Bayreuther
Wissenschaftlicher Betreuer:	Prof. Dr. Reinhold Hübl
Bearbeitungszeitraum:	18.10.2022 – 18.04.2023

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Titel „*Konstruktion eines kryptographischen Verfahrens basierend auf Reed-Solomon-Codes und Diskussion möglicher Angriffsmethoden*“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Freiburg im Breisgau, 18.04.2023 Roman Wetenkamp

Danksagung

Zuallererst möchte ich Ihnen, Herrn Professor Reinhold Hübl, ausdrücklich für Ihre herausragende und intensive Betreuung dieser Arbeit herzlich danken. Neben Ihrem stetigen konstruktiv-kritischen Blick auf dutzende Zwischenstände, der Suche nach ergänzenden Quellen oder sogar Ihrem Anfertigen zusätzlicher Ausarbeitungen sind es so zahlreiche zielführende und wegleitende Gedanken, die wesentlich dafür waren, dass diese Arbeit nun diese mich zufriedenstellende Gestalt annehmen konnte. Mein stetiger Wille, mich diesem Thema auch trotz seines aus meiner Sicht herausfordernden Charakters zu stellen und es mir nicht zu einfach zu machen, ist zu einem sehr großen Teil Ihrer so umfangreichen und wertschätzenden Unterstützung zu verdanken!

Auch fernab des inhaltlichen Austausches bin ich für Ihre Angebote und Ratschläge, die die Planung meines nächsten Studienabschnittes betreffen, außerordentlich dankbar – auf Ihren Rat und Ihre Erfahrung zählen zu dürfen, ist eine wichtige Quelle meiner Zuversicht.

Außerdem möchte ich Ihnen, Herrn Professor Johannes Bauer, dafür danken, dass Sie mein Interesse an Kryptographie durch Ihre exzellente Vorlesung „Kryptoanalyse und Methoden-Audit“ neu entfacht und die Brücke zwischen den mathematischen Hintergründen und der Implementierung kryptographischer Verfahren gebaut haben. Die – stellenweise ‚pfuschigen‘ – *SageMath*-Skripte im Anhang der Arbeit widme ich daher Ihnen, denn Sie stießen mich damals auf dieses geniale Werkzeug, dessen ‚Fanbase‘ sich nun mit mir mindestens um eins inkrementiert hat.

Zu zweifeln und keine sonderlich hohe Meinung von den eigenen Fähigkeiten zu haben, ist ein Problem, mit dem ich vermutlich nicht alleine bin unter allen Informatiker:innen (vermutlich sogar Studierenden und Forschenden nahezu jeder Disziplin), die tagelang versuchen, den einen Fehler in der Logik zu finden, der perfiderweise konsequent die Lösungen verfälscht. Deshalb möchte ich dir, meiner lieben Laila, sehr dafür danken, dass du nicht aufhörst, an mich zu glauben und mich kontinuierlich mit deiner Liebe bestärkst. Auch für deine intensive Recherchehilfe und deine Erfahrung bezüglich des wissenschaftlichen Arbeitens, die ich mir nun langsam selbst erarbeiten möchte, danke ich dir sehr herzlich. Es tut so gut, dass es dich gibt! Ich freue mich auf unsere zukünftigen gemeinsamen Projekte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quelltextverzeichnis

Algorithmenverzeichnis

Abkürzungsverzeichnis

AES	Advanced Encryption Standard
BSC	Binary Symmetric Channel
EDV	Elektronische Datenverarbeitung
GRS	Generalized Reed Solomon
ITS	Information Transmission System
JSON	JavaScript Object Notation
KEM	Key Encapsulation Mechanism
MDPC	Moderate Density Parity-Check
NIST	National Institute of Standards and Technology
PEM	Privacy Enhanced Mail
PKC	Public Key Cryptosystem
PKE	Public Key Exchange
QR	Quick Response
RS	Reed-Solomon-Codes

Kurzfassung (Abstract)

Abstract

While researchers in the field of cryptography get worried about the upcoming development of quantum computers, some well-studied but nevertheless not widely applied cryptosystems like the McEliece and Niederreiter scheme seem to be quantum resistent since they are not based on typical mathematical problems like the factorization or discrete logarithm problem. This thesis describes the theory behind those schemes including finite fields, polynomials and error-correcting codes like *Reed-Solomon* or *Goppa* codes. Furthermore two cryptosystems built on this class of codes are constructed, implemented in *SageMath* and reviewed in terms of security and applicability. Both cryptosystems share the same security properties and are relevant for the *Post Quantum Cryptography* research field, as some submissions to the NIST standardization process convey.

Zusammenfassung

Durch die voranschreitende Entwicklung von Quantencomputern sieht sich das Forschungsfeld der Kryptographie vor neuen Herausforderungen. Umfangreich erforschte, aber in der Praxis kaum eingesetzte Kryptosysteme wie jene von McEliece und Niederreiter sind vielversprechende Kandidaten für quanten-resistente Verfahren, da sie auf anderen Problemen als dem Faktorisierungsproblem oder dem Problem des diskreten Logarithmus basieren. In dieser Arbeit werden zunächst die theoretischen Hintergründe anhand von endlichen Körpern, Polynomen und fehler-korrigierenden Codes wie *Reed-Solomon-* oder *Goppa-*Codes erläutert, bevor die beiden genannten Kryptosysteme definiert, im Mathematik-Softwaresystem *SageMath* implementiert und in Hinblick auf Sicherheit und Anwendbarkeit verglichen werden. Beide Verfahren zeigen ähnliche Sicherheitseigenschaften und sind somit relevant für das Gebiet der *Post-Quanten-Kryptographie*, da einige Einreichungen im NIST-Standardisierungsprozess auf diesen Verfahren basieren.

1 Einleitung

„The lesson here is that it is insufficient to protect ourselves with laws;
we need to protect ourselves with mathematics."
– Bruce Schneier in [4]

In einer Welt, in der so viele Daten wie nie zuvor übertragen werden, digitale Kriegsführung und *Nation-state attacks* nicht mehr bloß Gegenstand dystopischer Science-Fiction-Literatur, sondern Alltag sind, steigt die Relevanz kryptographischer Verfahren, die die Vertraulichkeit und Integrität schützenswerter Daten selbst unter der Annahme sicherstellen, dass Angreifenden nahezu unbegrenzte Ressourcen zur Verfügung stünden.

Das Forschungsgebiet der *Post-Quanten-Kryptographie* [vgl. 5] hat die Entwicklung kryptographischer Systeme zum Gegenstand, die selbst mit den durch Quantentechnologie anzunehmenden Rechenleistungssteigerungen nicht gebrochen werden können. Ein aussichtsreicher Kandidat dafür ist das McEliece-Kryptosystem, das auf linearen, fehlerkorrigierenden Codes basiert. Jene Verbindung der Codierungstheorie und Kryptographie ist Gegenstand dieser Arbeit: Ausgehend vom McEliece-Kryptosystem soll ein verwandter Ansatz von Harald Niederreiter betrachtet werden, der im Vergleich zum McEliece-Kryptosystem nicht auf *Goppa*-, sondern auf *Reed-Solomon*-Codes basiert und dadurch zwar bessere Rechenzeiten erreicht, jedoch auch an Sicherheit einbüßt.

Diese Arbeit stellt zunächst die theoretischen Hintergründe der Codierungstheorie für kryptographische Zwecke dar, bevor basierend darauf die Arbeiten von McEliece und Niederreiter analysiert werden. Abschließend wird die Sicherheit der Verfahren unter anderem anhand der Arbeit von Sidelnikov und Shestakov betrachtet und die Bedeutung jener Verfahren im Kontext der *Post-Quanten-Kryptographie* diskutiert.

Ein Ziel dieser Arbeit ist es, die theoretischen Hintergründe hinter einzelnen Verfahren und Entwicklungen verständlich und dennoch fundiert zu vermitteln. Ergänzende Implementierungen in einem Mathematiksoftwaresystem sollen die Algorithmen verdeutlichen und zum Verständnis beitragen. Leser:innen dieser Arbeit sollen grundlegende Kenntnisse

über code-basierte Kryptographie erlangen, um die aktuellen Entwicklungen nachvollziehen zu können.

1.1 Thematische Übersicht

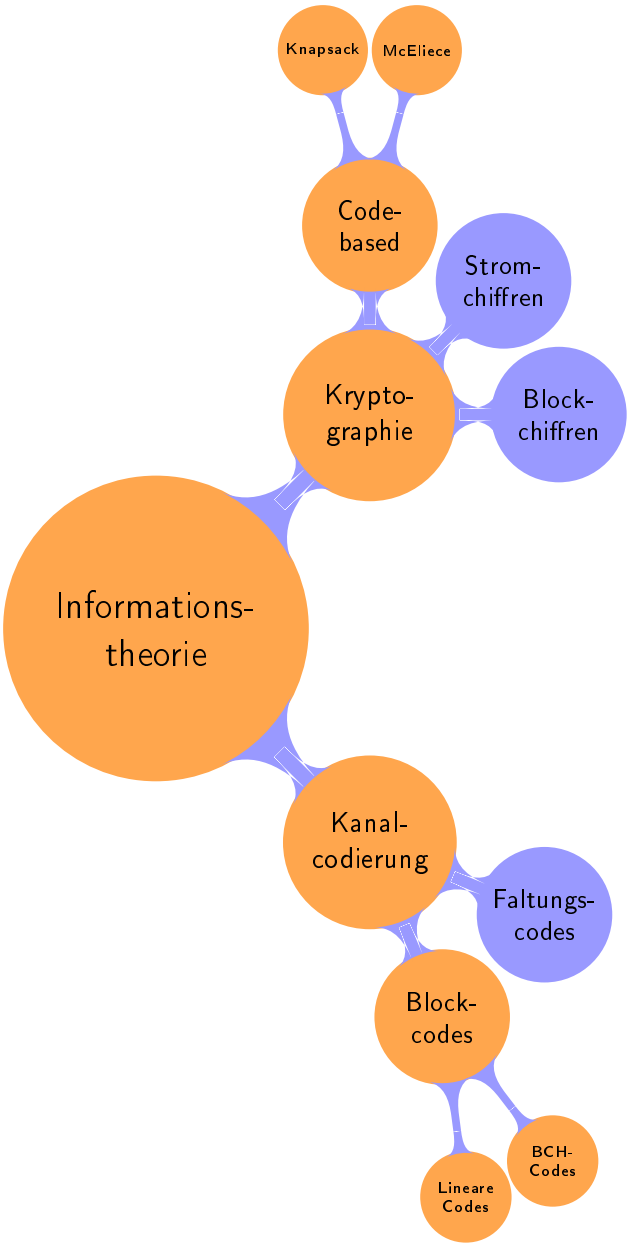

Abbildung 1.1: Übersicht über für diese Arbeit relevante und abgegrenzte Teilgebiete der Informationstheorie

2 Codierungstheorie

Da im Rahmen dieser Arbeit ein kryptographisches Verfahren entwickelt wird, das auf Elementen der Codierungstheorie basiert, wird diese zunächst in Definitionen und theoretischen Hintergründen dargestellt.

2.1 Übermittlung von Informationen

Sowohl in der Kryptographie, als auch in der Codierungstheorie fungieren **Nachrichten**, die über mit bestimmten Eigenschaften behaftete **Kanäle** übertragen werden, als die Objekte der Anschauung.

Definition 1
Eine **Nachricht** m sei definiert als eine endliche Folge von Zeichen $a_i \in \Sigma$, wobei Σ eine endliche Menge von Zeichen (genannt **Alphabet**) bezeichnet.

$$m = \langle a_0, a_1, ..., a_{n-2}, a_{n-1} \rangle \quad \forall i = 0, ..., n-1 \colon\ a_i \in \Sigma$$

Ein typisches Alphabet sind die Zeichen der *ASCII*-Kodierung, mit denen nahezu alle Worte und Sätze der natürlichen englischen Sprache gebildet werden können [vgl. 6]. Dieses Alphabet besteht nicht aus Zeichen der natürlichen Sprache, sondern aus 7 Bit langen Zahlenwerten, was die Anwendung von Codes oder kryptographischen Verfahren ermöglicht. Im Rahmen dieser Arbeit wird angenommen, dass Zeichen stets in einem Zahlenformat repräsentiert werden.

Die Definition einer informationstheoretischen Nachricht impliziert eine Autorenschaft, folglich muss jeder Nachricht eine Partei (eine natürliche Person, ein System oder ein Dienst) zugeordnet werden können, die im Folgenden als **Sender**[1] der Nachricht bezeichnet wird.

[1] Da sich die Anwendung der modernen Kryptographie sehr überwiegend mit dem Austausch von verschlüsselten Nachrichten zwischen Systemen und nicht unmittelbar zwischen natürlichen Personen befasst, wird hier die männliche Form verwendet (Sender = Dienst/System).

Wird diese Nachricht nun über einen Kanal an eine andere Partei übertragen, so nennen wir diese den **Empfänger**.

Ein **Kanal** bezeichne ein Medium zur Datenübertragung wie beispielsweise einen elektrischen Leiter, einen Lichtwellenleiter oder die Luft für eine drahtlose Verbindung. Es gibt Kanäle, die Informationen **digital** übertragen, also als diskrete Binärwerte, und im Gegensatz dazu Kanäle, die fortlaufend und stetig Signale übertragen [vgl. 7, S. 1]. **Rauschen** bezeichne nicht-deterministische Daten, die Nachrichten bei einer Übertragung über einen Kanal unbeabsichtigt hinzugefügt werden und so die Nachricht verändern, sie folglich mit **Fehlern** versetzen [vgl. 8, S. 1].

Definition 2
Ein **Kanal** K mit der Menge der Eingabewerte I und der Menge der Ausgabewerte J sei definiert als Matrix

$$K_{i,j} = \mathsf{Pr}(j \mid i) \quad \forall i \in I,\, j \in J$$

wobei $\mathsf{Pr}(j \mid i)$ definiert sei als die Wahrscheinlichkeit für die Ausgabe j unter der Bedingung der Eingabe i.

Sofern mindestens ein Koeffizient $i \neq j$ der Matrix einen Wert größer als 0 aufweist, wird der Kanal als **rauschend** bezeichnet [vgl. 9, S. 73f.].

Beispiel 1
Ein **Binary Symmetric Channel (BSC)** Γ mit einer Bitfehlerwahrscheinlichkeit e ist definiert durch die Matrix

$$\Gamma = \begin{pmatrix} \Gamma_{00} & \Gamma_{01} \\ \Gamma_{10} & \Gamma_{11} \end{pmatrix} = \begin{pmatrix} 1-e & e \\ e & 1-e \end{pmatrix}.$$

Dass ein Kanal eine Nachricht ohne Rauschen überträgt, ist zwar ein erstrebenswerter Zustand, praktisch jedoch aufgrund physikalischer Effekte nicht zu erreichen. Jede physische Datenübertragung verläuft nicht fehlerfrei, weshalb die Beziehung $m = m'$ daher nur in einem theoretischen Idealfall gilt. Unter anderem diese Feststellung motiviert die Begründung für die Beschäftigung mit der Codierungstheorie.

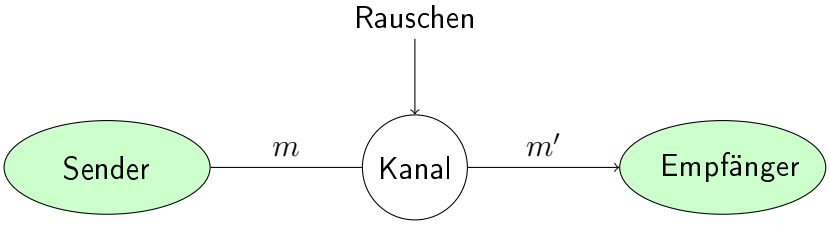

Abbildung 2.1: Gegenstand der Codierungstheorie (nach [10, S. 1])

2.2 Problemstellung und Zielsetzung

Da die Datenübertragung über eine Vielzahl von Kanälen eben nicht fehlerfrei verläuft, liegt es nahe, die Daten so zu übertragen, dass fehlende Bits aus dem Rest der Nachricht erschlossen werden können. Dies ist auch bei natürlicher Sprache der Fall: Unsere Worte enthalten häufig Buchstaben, die nicht zwingend erforderlich sind, um das gemeinte Wort zu erkennen [vgl. 8, S. 3].

Überträgt man diese Erkenntnis auf Nachrichten einer beliebigen Sprache, so lassen sich auch im allgemeinen Fall durch das Hinzufügen von redundanten Informationen Nachrichten erzeugen, deren Informationsgehalt sich durch die Übermittlung nicht verringert hat. Ein spezieller Typ dieser Verfahren wird als **fehlerkorrigierende Codes** bezeichnet [vgl. 8, S. 3]. Diese Codes sind dem Gebiet der **Kanalcodierung** zuzurechnen, die das Ziel hat, die Qualität der Übertragung auf verlustbehafteten Kanälen sicherzustellen. Sie grenzt sich ab von der **Quellencodierung**, die Verfahren bündelt, welche die Transformation der zu versendenden Daten zum Ziel hat, beispielsweise zur Kompression und Redundanzverringerung [vgl. 11, S. 1]. Der Fokus dieser Arbeit liegt dabei auf der Kanalcodierung, da die Anwendung dieser Codes für die Kryptographie seit im Vergleich längerer Zeit diskutiert und erforscht wird. Auch Codes zur Quellencodierung können für kryptographische Zwecke genutzt werden, jedoch sind diese Verfahren im Vergleich signifikant weniger erforscht [vgl. 12].

2.3 Kanalcodierung

Die Relevanz der Frage, wie möglichst viele Übertragungsfehler in einer Datenübertragung vermieden oder korrigiert werden können, stieg mit der Verbreitung von EDV-Systemen und nicht zuletzt dem Internet rasant an [vgl. 7, S. 209], während relevante Beiträge der

Grundlagenforschung wie jene von HAMMING aus dem Jahr 1950 unverändert als gültig anzusehen sind und weiterhin die Basis aufbauender Definitionen bilden.

2.3.1 Grundbegriffe

Um Codier- und Decodiervorgänge beschreiben zu können, ist eine Erweiterung und Präzisierung der Abbildung 2.1 erforderlich. Die Zuordnung eines Codewortes c zu einer Nachricht m wird als **Codierung** (der Nachricht) bezeichnet.

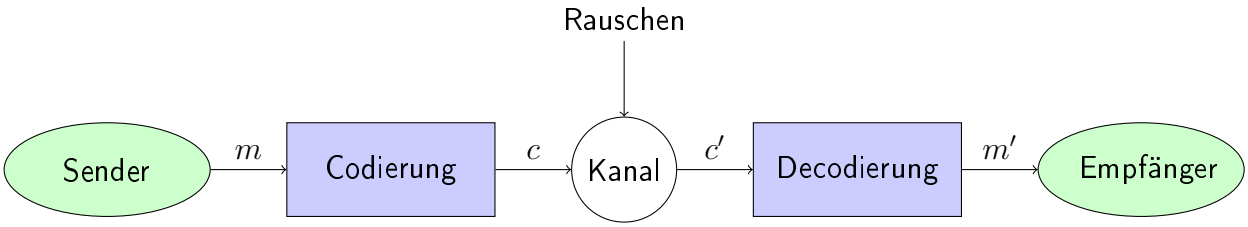

Abbildung 2.2: Vereinfachte Darstellung eines ITS (nach [7, S. 3])

Definition 3
Sei A ein Alphabet und $n \in \mathbb{N}$.
Dann sei A^n die Menge aller n-Tupel der Form $A^n = \{\langle a_0, a_1, ..., a_{n-1}\rangle \mid a_i \in A\}$.

Ein **Blockcode** C der Länge n über dem Alphabet A ist definiert als $C \subseteq A^n$, wobei $|C| > 0$ gelten muss.

Ist $o = |C|$ und B mit $|B| < o$ die Menge zu codierender Informationseinheiten über einem Alphabet A', so ist jede injektive Abbildung $f\colon B \to C$ eine **Codierfunktion** [vgl. 11, S. 10].

Neben Blockcodes können auch sogenannte **Faltungscodes** zur Fehlerkorrektur verwendet werden. Hierbei werden die Informationen nicht unabhängig voneinander blockweise, sondern über Schieberegister in Abhängigkeit zueinander codiert, wodurch sich eine bessere Performanz im Gegensatz zu Blockcodes ergibt [vgl. 14, S. 752]. Diese Ansätze, Daten blockweise oder als Datenstrom über Schieberegister zu codieren, weisen Parallelen zur kryptographischen Unterscheidung zwischen Block- und Stromchiffren auf. Der Fokus dieser Arbeit wird aufgrund der kryptographischen Eigenschaften von Blockchiffren wie der

höheren Diffusion und dem Lawineneffekt auf Blockcodes liegen.

Als Maß der Qualität einer Übertragung beziehungsweise der Verfälschung einer Nachricht durch Rauschen, aber auch für die Unterscheidbarkeit von Codeworten, bietet sich die **Hamming-Distanz** an.

Definition 4
Seien $a = \langle a_0, a_1, ..., a_{n-1} \rangle$, $b = \langle b_0, b_1, ..., b_{n-1} \rangle \in A^n$ und A ein Alphabet, so ist die **Hamming-Distanz** d eine Metrik auf A^n. Sie ist definiert als die Anzahl der abweichenden Stellen in zwei Codeworten a und b:

$$d(a, b) = |\{i \mid 1 \leq i \leq n, a_i \neq b_i\}|$$

Des Weiteren sei die **Minimaldistanz** eines Blockcodes C mit $|C| > 1$ definiert als

$$d(C) = \min\{d(x, y) \mid x, y \in C, x \neq y\}$$

[vgl. 11, S. 11] [vgl. 15, S. 105] [vgl. 13, S. 155].

Beispiel 2
Seien $A = \{0, 1\}$ und $n = 5$. Dann ist $A^n = \{\langle 0,0,0,0,0 \rangle, \langle 0,0,0,0,1 \rangle, ..., \langle 1,1,1,1,1 \rangle\}$. Dann gilt für einen Blockcode C:

- $C = A^n \quad \rightarrow \quad d(C) = 1$
- $C = \{\langle 0,0,0,0,1 \rangle, \langle 0,0,0,1,0 \rangle, ..., \langle 1,0,0,0,0 \rangle\} \quad \rightarrow \quad d(C) = 2$
- $C = \{\langle 0,1,0,1,0 \rangle, \langle 1,0,1,0,1 \rangle\} \quad \rightarrow \quad d(C) = 5$

Wenn die Werte einer Codierungsfunktion im Bildbereich weit verstreut sind, der Code also eine große Minimaldistanz aufweist, werden die einzelnen Codeworte unterscheidbarer. Folglich ergibt sich aus der Minimaldistanz ein relevantes Kriterium für die Eigenschaft eines Codes, **fehlererkennend** oder sogar **fehlerkorrigierend** zu sein. Sie beeinflusst, wie viele Fehler erkannt beziehungsweise korrigiert werden können [vgl. 13, S. 155].

HAMMING nutzt für seine Argumentation in [13] eine geometrische Betrachtung mithilfe von Kugeln:

Definition 5
Eine **abgeschlossene Kugel** $B(a, r)$ um einen Punkt a mit Radius r in einem (metrischen) Raum X sei definiert durch

$$B(a, r) = \{x \in X \mid d(a, x) \leq r\}$$

Als Abstandsmaß d kann hier beispielsweise die eingeführte HAMMING-Distanz verwendet werden.

Definition 6
Sei B eine Kugel mit Radius $r \geq 2$ und Mittelpunkt $x \in A^n$. Dann liegen alle $a_i \in A^n$ mit $d(a_i, x) = r$ auf der Kugeloberfläche von B. Sei $C \subseteq A^n$ nun ein Blockcode mit $d(C) = r$ und $x \in C$. Dann liegen alle $c \in (C \setminus \{x\})$ auf der Kugeloberfläche oder außerhalb der Kugel B.

Bemerkung 1
Ein Punkt $a_j \neq x$ mit $d(a_i, x) < r$ (unterscheidet sich in in weniger als r Stellen von x) liegt nicht auf der Kugeloberfläche und kann folglich kein fehlerfreies Codewort sein. Ein Code C mit Minimaldistanz 2 ist damit **fehlererkennend** für maximal ein falsch übertragenes Zeichen (notiert: „1-fehlererkennend“).

Ferner ist ein Code C' mit Minimaldistanz 3 **fehlerkorrigierend**: Sei a_r ein gültiges Codewort mit einer Minimaldistanz von 3 zu allen anderen Codeworten in C'. Für ein in einer Stelle abweichendes Wort a_f gilt folglich $d(a_r, a_f) = 1$, aber $\forall a_i \in C' \setminus \{a_r, a_f\} : d(a_i, a_f) > 1$. Damit lässt sich a_f eindeutig a_r zuordnen, wodurch sich der Fehler korrigieren lässt und C' **1-fehlerkorrigierend** ist [vgl. 13, S. 155f.].

Die Eigenschaft eines Codes, fehlererkennend oder fehlerkorrigierend zu sein, ist skalierbar:

Definition 7
Seien $c, c' \in C; c \neq c'$ Codeworte eines Blockcodes $C \in A^n$, $\epsilon \in \mathbb{N}$, $B_\epsilon(c) = \{x \in A^n \mid d(x, c) \leq \epsilon\}$ und $B_\epsilon(c')$ analog $B_\epsilon(c') = \{x \in A^n \mid d(x, c') \leq \epsilon\}$.
Dann ist C **ϵ-fehlerkorrigierend**, wenn $\forall c, c' \in C : B_\epsilon(c) \cap B_\epsilon(c') = \emptyset$ gilt [vgl. 11, S. 12f.].

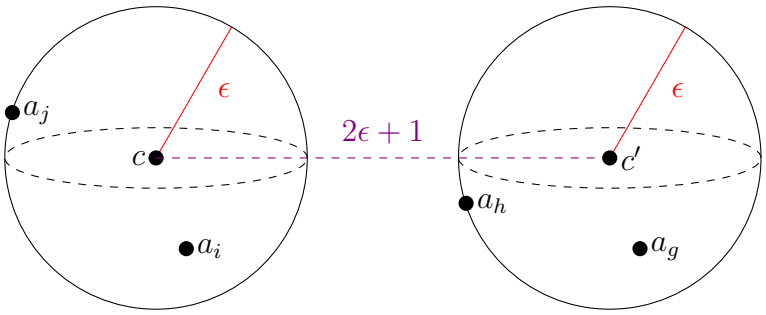

Abbildung 2.3: Visualisierung der Kugelinterpretation (nach [13])

Bemerkung 2

Ein Blockcode C ist ϵ-fehlerkorrigierend, wenn $d(C) \geq 2 \cdot \epsilon + 1$ gilt [vgl. 11, S. 12f.].

Definition 8

Sei $c \in C$ Codewort eines Blockcodes $C \in A^n$ und $B_t(c) = \{x \in A^n \mid d(x, c) \leq t\}$ die zu c gehörige Kugel mit allen Elementen aus A^n, die höchstens t entfernt sind.
Dann ist C t-**fehlererkennend**, wenn $\forall c \in C \colon B_t(c) \cap (C \setminus \{c\}) = \emptyset$, die Kugel um c also keine anderen Codeworte enthält [vgl. 11, S. 13].

Bemerkung 3

Ein Blockcode C ist t-fehlererkennend, wenn $d(C) \geq t + 1$ gilt [vgl. 11, S. 13].

Beispiel 3

Sei $A = \{0, 1\}$ und $A^3 = \{\langle 0,0,0\rangle, \langle 0,0,1\rangle, ..., \langle 1,1,1\rangle\}$. Ferner sei $C \in A^3$ ein Blockcode für Nachrichten der Form $m = \langle m_1, m_2\rangle$ mit der Codierfunktion $f \colon M \to C, \langle m_1, m_2\rangle \mapsto \langle m_1, m_2, ((m_1 + m_2) \mod 2)\rangle$.

Für die Nachricht $x = \langle 0, 1\rangle$ ergibt sich also $f(x) = \langle 0, 1, 1\rangle$.

Dann ist $C = \{\langle 0,0,0\rangle, \langle 0,1,1\rangle, \langle 1,0,1\rangle, \langle 1,1,0\rangle\}$. Daraus folgt $d(C) = 2$. Damit ist C 1-fehlererkennend, da $2 = 1 + 1$, aber nicht fehlerkorrigierend, da $2 \ngeq 2 \cdot 1 + 1$ gilt.

Welches Potenzial von fehlerkorrigierenden Codes für das zugrundeliegende Problem ausgeht, verdeutlicht das folgende Theorem:

2.3.2 Noisy Channels Coding Theorem

SHANNON konnte 1948 zeigen, dass es für einen binären, symmetrischen und rauschenden Kanal (BSC) Codes gibt, bei denen die Wahrscheinlichkeit für das Auftreten von Fehlern beliebig klein wird, solange sich die Datenrate der Übertragung unterhalb der Kapazität des Kanals befindet.

Theorem 1
Sei C die **Kapazität** eines binären, rauschenden Kanals, n die Länge uniformer Codeworte, $P(E)$ die Fehlerwahrscheinlichkeit und D die Datenübertragungsrate, wobei $D < C$. Dann gilt

$$P(E) \leq 2^{-n \cdot e(D)}$$

wobei $e(D)$ eine vollständig von Kanalparametern abhängige positive Funktion, genannt **Fehlerexponent**, ist.

Folglich ist es möglich, Datenübertragungsvorgänge unter der Wahl einer geeigneten Datenübertragungsrate und einer Codierung nahezu fehlerfrei umzusetzen, völlig unabhängig von der Länge der zu übertragenden Nachricht. Es folgt auch, dass sich die Fehleranzahl nicht linear zur Fehleranfälligkeit (angegeben durch die Kapazität) des Kanals verhält [vgl. 7, S. 209ff.].

2.4 Finitfeldarithmetik

Viele der in der Praxis häufig eingesetzten Codierfunktionen basieren auf der Arithmetik in endlichen Körpern und *Galois*-Feldern, die im Folgenden eingeführt werden.

Ausgangspunkt der Definition von endlichen Körpern ist nach LIDL UND NIEDERREITER die algebraische Struktur der **Gruppe**.

2.4.1 Gruppen

Definition 9
Sei G eine Menge und $\circ$ eine binäre Relation $\circ\colon G \times G \to G$ mit folgenden Eigenschaften:

1. **Assoziativität:**
   $\forall\, a, b, c \in G\colon \quad a \circ (b \circ c) \Leftrightarrow (a \circ b) \circ c$
2. **Existenz eines neutralen Elements:**
   $\exists\, e \in G,\, \forall\, a \in G\colon \quad a \circ e \Leftrightarrow e \circ a \Leftrightarrow a$
3. **Existenz inverser Elemente:**
   $\forall\, a \in G,\, \exists\, a^{-1} \in G\colon \quad a \circ a^{-1} \Leftrightarrow a^{-1} \circ a \Leftrightarrow e$

Dann ist $(G, \circ)$ eine **Gruppe**.

Gilt für eine Gruppe zudem $\forall\, a, b \in G\colon \quad a \circ b \Leftrightarrow b \circ a$, so ist sie **kommutativ**, auch bezeichnet als **abelsch** [vgl. 16, S. 2].

Beispiel 4
Gegeben seien die Menge der ganzen Zahlen $\mathbb{Z}$ und eine binäre Relation.

- $(\mathbb{Z}, +)$, wobei $+$ die Addition bezeichne, ist eine **abelsche Gruppe**, da die folgenden Eigenschaften erfüllt sind:
  1. Assoziativität:
     Seien $a, b, c \in \mathbb{Z}$. Dann gilt mit dem Distributivgesetz der Addition $a+(b+c) \Leftrightarrow a + b + c \Leftrightarrow (a + b) + c$
  2. Existenz eines neutralen Elements:
     $\forall\, x \in \mathbb{Z}\colon \quad x + 0 \Leftrightarrow 0 + x \Leftrightarrow x$
  3. Existenz inverser Elemente:
     $\forall\, x \in \mathbb{Z},\, \exists\, -x \in \mathbb{Z}\colon \quad x + (-x) = 0$
  4. Kommutativität:
     $\forall\, a, b \in \mathbb{Z}\colon \quad a + b \Leftrightarrow b + a$
- $(\mathbb{Z}, \cdot)$, wobei $\cdot$ die Multiplikation bezeichne, ist keine Gruppe, da die inversen Elemente zu Elementen von $\mathbb{Z}$ nicht Teil von $\mathbb{Z}$ sind.

$$\forall\, x \in \mathbb{Z} \setminus \{1, -1\}: \quad x \cdot \frac{1}{x} = 1, \text{ aber } \frac{1}{x} \notin \mathbb{Z}$$

Definition 10
Eine multiplikative Gruppe ist **zyklisch**, sofern

$$\exists\, a \in G, \forall\, b \in G, \exists\, j \in \mathbb{Z}: \quad b = a^j$$

gilt. Dann ist a ein **Generator** der multiplikativen Gruppe[2] G [vgl. 16, S. 3f.]. Die Gruppe ist durch diesen Generator a hinreichend beschrieben.

Kryptographische Bedeutung In Bezug auf Gruppen existieren einige mathematische Probleme, die sich für die Konstruktion kryptographischer Verfahren nutzen lassen. MYASNIKOV, USHAKOV und SHPILRAIN stellen in [17] kryptographische Protokolle dar, die beispielsweise auf dem *Konjugationssuchproblem* nicht-kommutativer Gruppen basieren: Für zwei Elemente $u, v \in G$ einer Gruppe G soll mindestens ein $x \in G$ gefunden werden, sodass $u^x = v$ gilt. Ist kein anderer Algorithmus für dieses Problem für die Gruppe G bekannt, so lässt sich $x \to u^x$ als Einwegfunktion auffassen und so ein kryptographisches Protokoll konstruieren [vgl. 17, S. 37f.].

Abhängig davon, ob eine Gruppe abelsch ist oder nicht, lassen sich unterschiedliche Such- oder Entscheidungsprobleme zur Konstruktion kryptographischer Verfahren verwenden, die in [17] dargestellt, aber in dieser Studienarbeit nicht weiter behandelt werden.

2.4.2 Körper

Definition 11
Ein **Ring** $(R, +, \circ)$ ist eine nicht-leere Menge R mit den zwei auf ihr definierten binären Relationen, notiert durch $+$ und $\circ$, für die gilt:

- R ist abelsche Gruppe bezüglich der Operation $+$
- Die Operation $\circ$ ist assoziativ, folglich gilt

$$\forall\, a, b, c \in R: \quad (a \circ b) \circ c \Leftrightarrow a \circ (b \circ c)$$

[2] Wird zu einer Gruppe keine binäre Relation angegeben, handelt es sich implizit um eine multiplikative Verknüpfung.

- Das Distributivgesetz gilt, also ist

$$\forall\, a, b, c \in R: \quad \big(a \circ (b + c) \Leftrightarrow a \circ b + a \circ c\big) \wedge \big((b + c) \circ a \Leftrightarrow b \circ a + c \circ a\big)$$

Ein Ring ist **kommutativ**, wenn die Operation $\circ$ kommutativ ist. Ferner ist ein Ring ein **Divisionsring**, falls $(R \setminus \{0\}, \circ)$ eine Gruppe bildet [vgl. 16, S. 13][vgl. 18, S. 1f.].

Ein Ring beinhaltet folglich eine Gruppe, jedoch auch eine Operation, die nicht notwendigerweise die Anforderungen an eine Gruppe erfüllt. Da beispielsweise der Menge $\mathbb{Z}$ inverse Elemente bezüglich der Multiplikation fehlen, ist $(\mathbb{Z}, +, \circ)$ dennoch ein Ring, wenn auch kein Divisionsring.

Definition 12
Ein kommutativer Divisionsring wird als **Feld** oder **Körper** bezeichnet [vgl. 16, S. 13].

Um nun endliche Körper und *Galois-Körper* definieren zu können, ist das Konzept der **Restklasse** unabdingbar, das nun zunächst wieder auf Ringen definiert wird.

Definition 13
Sei R ein Ring. Dann ist J ein **Unterring** von R, falls J abgeschlossen ist unter beiden Operationen $+$ und $\circ$ [vgl. 16, S. 13].

Bemerkung 4
Abgeschlossenheit bezeichnet hier, dass das Ergebnis einer Operation mit zwei beliebigen Elementen des Rings als Operanden einem Element entspricht, das wiederum Element des Rings ist.

Definition 14
Ferner ist J ein **Ideal** von R, falls J ein Unterring von R ist und

$$\forall\, a \in J,\, r \in R: \quad \big((ar \in J) \wedge (ra \in J)\big)$$

gilt [vgl. 16, S. 13].

Gilt $I = (a)$ für ein $a \in I$, so ist I ein von a erzeugtes **Hauptideal**.

Bemerkung 5
Ein Ideal ist nun nicht mehr nur bezüglich eigenen Elementen und beiden Operationen abgeschlossen, sondern in Bezug auf die Multiplikation auch bezüglich Elementen des übergeordneten Rings.

Definition 15
Sei R eine Restklasse, $a \in R$ und J ein Ideal von R. Die **Restklasse** von $a \mod J$ sei definiert als Menge aller Elemente der Form $(a + c) \in R, \forall\, c \in J$ [vgl. 16, S. 13][vgl. 18, S. 10]:

$$[a] = a + J = \{(a + c) \in R;\, c \in J\}$$

Bemerkung 6
Zwei Elemente $a, b \in R$ sind *kongruent* modulo J, wenn sie Elemente der gleichen Restklasse bezüglich J sind, geschrieben $a \equiv b \mod J$.

Auch auf Restklassen lassen sich die Operationen eines Rings anwenden:

$$(a + J) + (b + J) = (a + b) + J$$
$$(a + J) \circ (b + J) = ab + J$$

Bemerkung 7
Durch diese beiden Operationen auf Restklassen entsteht ein Ring, der als **Restklassenring** bezeichnet und mit R/J notiert wird [vgl. 16, S. 13f.] [vgl. 18, S. 10].

Beispiel 5
Ein klassisches Beispiel für einen auf diese Weise entstandenen Restklassenring ist $\mathbb{Z}/(p),\ p \in \mathbb{P}$, wobei $\mathbb{P}$ die Menge aller Primzahlen bezeichnet. Hier ist (p) ein durch p generiertes Hauptideal. Dieser Ring ist ein Körper, wie in [16] bewiesen wird, und aufgrund der endlichen Anzahl an Elementen **finit**. Die Elemente sind beschrieben durch

$$\{[0] = 0 + (n), [1] = 1 + (n), ...\}$$

Der Restklassenring $\mathbb{Z}/(3)$ enthält die Elemente $[0], [1]$ und $[2]$.

Nun ist die Definition der *Galois-Körper* unmittelbar möglich:

Definition 16
Sei $p \in \mathbb{P}$. Dann sei $\mathbb{F}_p$ der Körper, der durch den Restklassenring $\mathbb{Z}/(p) = \mathbb{Z}/p\mathbb{Z}$ beschrieben wird und folglich die Struktur $\mathbb{F}_p = \{[0], [1], ..., [p-1]\}$ aufweist. $\mathbb{F}_p$ wird dann als **Galois-Körper** der Ordnung p bezeichnet [vgl. 16, S. 15][vgl. 19, S. 75].

Beispiel 6
Beispiele für endliche Körper sind:

- $\mathbb{F}_2 = \{[0], [1]\}$
- $\mathbb{F}_3 = \{[0], [1], [2]\},$ beispielsweise $\{0, 1, -1\}$

Neben GALOIS-Körpern, die auf diese Weise erzeugt wurden, bilden auch alle Primzahlpotenzen $p^f, p \in \mathbb{P}$ und $f \in \mathbb{N}$ jeweils einen endlichen Körper.

Theorem 2
Für jede Primzahlpotenz $p^f, p \in \mathbb{P}$ und $f \in \mathbb{N}$ gibt es (bis auf Isomorphie) genau einen endlichen Körper K mit $|K| = p^f$.

Die durch Primzahlpotenzen entstandenen endlichen Körper basieren nicht bloß auf den Restklassen des Unterkörpers $\mathbb{F}_p$, sondern werden mit **Polynomen** erweitert.

2.4.3 Polynome

Durch die Verkettung der beiden Operationen eines Körpers $\mathbb{K}$ lassen sich **Polynome** erzeugen. Die Menge aller Polynome über einem Körper $\mathbb{K}$ wird definiert als

$$\mathbb{K}[x] = \{p(x) = a_n x^n + ... + a_1 x + a_0 \mid a_i \in \mathbb{K}\}$$

[vgl. 20, S. 118f.] und bildet einen kommutativen Ring, aufgrund der fehlenden Abgeschlossenheit der Division jedoch keinen Körper. Wird ein Polynom durch ein anderes dividiert, so kann dabei ein Rest entstehen, was die Anwendung des Kongruenzbegriffs nahelegt. Die Polynomdivision aller Polynome aus einem Körper $\mathbb{K}$ mit einem Polynom $m(x)$ erzeugt einen **Restklassenring** $\mathbb{K}[x]/(m(x))$ mit der bekannten Addition und Multiplikation auf Polynomen.

Die Antwort auf die Frage, ob jener Ring auch ein (endlicher) Körper ist, ergibt sich daraus,

ob zu jedem Polynom ein multiplikatives Inverses Teil des Rings ist. Für diese Bedingung hinreichend ist folgende Eigenschaft eines Polynoms:

Definition 17
Ein Polynom $p(x) \in \mathbb{K}$ vom Grad (also dem höchsten Exponenten des Polynoms) $\deg(p) > 1$ wird als **irreduzibel** bezeichnet, falls es kein Polynom $q(x) \in \mathbb{K}$ mit $0 < \deg(q) < \deg(p)$ gibt, das $p(x)$ teilt [vgl. 20, S. 129].

Wird ein Polynomring nun durch ein reduzibles Polynom dividiert, also durch ein Polynom, das sich selbst weiter dividieren lässt, existiert zu jedem seiner Teiler kein multiplikatives Inverses mehr, weshalb dadurch kein Körper beschrieben wird.

Theorem 3
Ein Ring $\mathbb{K}[x]/(m(x))$ ist genau dann ein Körper, wenn $m(x)$ irreduzibel über $\mathbb{K}$ ist [vgl. 20, S. 130].

Ein GALOIS-Körper $\mathbb{F}_p$ kann auf diese Weise zu einem Körper $\mathbb{F}_{p^k}$ erweitert werden, indem aus dem Polynomring $\mathbb{F}_p[x]$ ein irreduzibles Polynom $m(x)$ vom Grad k ausgewählt wird. Dann ist

$$\mathbb{F}_{p^k} \quad \Leftrightarrow \quad \mathbb{F}_p[x]/(m(x)).$$

Beispiel 7
Typische Beispiele für endliche Körper sind:

- Über dem Körper $\mathbb{F}_2$ ist x^2+x+1 ein irreduzibles Polynom, das zur Körpererweiterung genutzt werden kann und nun den endlichen Körper $\mathbb{F}_{2^2} = \{[0], [1], [x], [x+1]\}$ erzeugt.
- Für das Advanced Encryption Standard (AES)-Kryptosystem und eine Vielzahl weiterer kryptographischer und codierungstheoretischer Anwendungen wird der Körper $\mathbb{F}_{2^8}$ mit dem definierenden Polynom $x^8 + x^4 + x^3 + x + 1$ verwendet. Er enthält 256 Elemente [vgl. 20, S. 131].

3 Lineare fehlerkorrigierende Codes für kryptographische Zwecke

Mithilfe der vorgestellten Polynome soll nun eine Klasse von Blockcodes betrachtet werden, zu der auch *Reed-Solomon*-Codes und *Goppa*-Codes, die beiden Codes aus den Verfahren von NIEDERREITER und MCELIECE, zählen. Diese Klasse wird als **lineare Codes** bezeichnet.

Für die Definition linearer Codes ist eine Wiederholung der algebraischen Struktur des **Vektorraums** notwendig. In [19] ist folgende Definition zu finden:

Definition 18
Sei K ein Körper. Eine nicht-leere Menge V mit einer additiven Verknüpfung

$$+ \colon V \times V \to V; \quad \langle v, w \rangle \mapsto v + w$$

und einer Skalarmultiplikation

$$\circ \colon K \times V \to V; \quad \langle \lambda, v \rangle \mapsto \lambda \circ v$$

heißt K-**Vektorraum**, sofern gilt:

- $(V, +)$ ist **abelsche Gruppe**.
- Die Skalarmultiplikation ist sowohl bezüglich des Skalars als auch der Komponente distributiv und assoziativ, ferner ist das Element $1 \in K$ das neutrale Element für diese Skalarmultiplikation. So muss $\forall\, \lambda, \mu \in K$ und $\forall\, v, w \in V$ gelten:

$$\begin{aligned}(\lambda + \mu) \circ v &= \lambda \circ v + \mu \circ v \\ \lambda \circ (v + w) &= \lambda \circ v + \lambda \circ w \\ \lambda \circ (\mu \circ v) &= (\lambda \circ \mu) \circ v \\ 1 \circ v &= v\end{aligned}$$

[vgl. 19, S. 95]

Vektorräume können mit einer beliebigen Dimension konstruiert werden und beinhalten folglich Vektoren dieser Dimension. In dieser Arbeit werden insbesondere Vektorräume mit Dimension $n \geq 2$ verwendet, auf denen eine komponentenweise Addition und skalare Multiplikation definiert sind. Für die Definition linearer Codes ist es erforderlich, Vektorräume $\mathbb{F}_q^n$ auf endlichen Körpern $\mathbb{F}_q$ zu definieren.

Theorem 4
Sei $\mathbb{F}_q$ ein endlicher Körper. Für jedes $n \geq 1$ ist $\mathbb{F}_q^n$ ein $\mathbb{F}_q$-Vektorraum der Dimension n.

Beweis. Die Eigenschaften folgen unmittelbar, so ist die komponentenweise Addition definiert als

$$\langle v_0, v_1, ..., v_{n-1} \rangle + \langle w_0, w_1, ..., w_{n-1} \rangle = \langle v_0 + w_0, v_1 + w_1, ..., v_{n-1} + w_{n-1} \rangle$$

und die Skalarmultiplikation als

$$k \circ \langle v_0, v_1, ..., v_{n-1} \rangle = \langle k \circ v_0, k \circ v_1, ..., k \circ v_{n-1} \rangle.$$

Damit ist leicht zu verifizieren, dass alle Forderungen erfüllt sind. □

Definition 19
Ein **Untervektorraum** W eines Vektorraums V ist eine Teilmenge $W \subset V$, für die gilt:

- $W \neq \emptyset$
- $v, w \in W \Rightarrow v + w \in W$
- $v \in W, \lambda \in K \Rightarrow \lambda \circ v \in W$

[vgl. 19, S. 96]

Definition 20
Sei $\mathbb{F}_q$ ein endlicher Körper und $|\mathbb{F}_q| = q$. Dann ist ein Blockcode C ein **linearer Code** mit dem Alphabet $\mathbb{F}_q$ und Länge n, falls die Menge C ein **Untervektorraum** von $\mathbb{F}_q^n$ ist [vgl. 11, S. 29].

Folglich beschreibt ein linearer Code Tupel der Länge n mit Komponenten aus $\mathbb{F}_q$. Ferner stellt die Untervektorraum-Eigenschaft sicher, dass die Addition von Codeworten und die

Multiplikation eines Codewortes mit einem Skalar aus dem übergeordneten endlichen Körper $\mathbb{F}_q$ weiterhin valide Codeworte erzeugen. Dies ist eine elementare Eigenschaft linearer Codes.

Eine Unterklasse der linearen Codes sind die **zyklischen Codes**.

3.1 Zyklische Codes

Definition 21
Ein k-dimensionaler Untervektorraum C von $\mathbb{F}_q^n$ ist dann ein **zyklischer Code**, wenn

$$\forall \langle a_0, a_1, ..., a_{n-1} \rangle \in C: \quad \langle a_{n-1}, a_0, a_1, ..., a_{n-2} \rangle \in C$$

gilt [vgl. 8, S. 42].

Ein linearer Code ist folglich dann **zyklisch**, falls zu jedem Codewort auch das Codewort Teil des Codes ist, das durch eine zyklische Verschiebung entsteht. Ein solcher Code ist beispielsweise der *gewöhnliche Reed-Colomon-Code*, der neben der verallgemeinerten, nicht zyklischen Version Gegenstand des nächsten Abschnitts ist.

Theorem 5
Zyklische Codes der Länge n sind Ideale des Rings $R = \mathbb{F}_q[x]/(x^n - 1)$.

Das Polynom $x^n - 1$ fungiert hierbei als Modul, sodass der Ring R folglich die Restklassen bezüglich dieser modularen Reduktion von $\mathbb{F}_q$ enthält.

Beweis. Sei C ein linearer zyklischer Code der Länge n über einem Körper F_q und $a = a(x) = a_0 + a_1 \cdot x + ... + a_{n-1} \cdot x^{n-1} \in C$. Dann ist

$$\begin{aligned} & x \cdot a \\ &= x \cdot (a_0 + a_1 \cdot x + ... + a_{n-1} \cdot x^{n-1}) \\ &= a_0 \cdot x + a_1 \cdot x^2 + ... + a_{n-1} \cdot x^n \\ &= a_{n-1} + a_0 \cdot x + ... + a_{n-2} \cdot x^{n-1} \end{aligned}$$

genau das zyklisch verschobene Codewort $a' \in C$.
Dies folgt aus $x^n \equiv 1 \mod x^n - 1$ [vgl. 9, S. 149f.]. □

Durch die Linearität des Codes C lässt sich ein $a \in C$ nicht nur beliebig oft zyklisch verschieben, also mit x multiplizieren, sondern auch lineare Kombinationen (also Summen) gültiger Codeworte ergeben wiederum ein gültiges Codewort in C. Somit stellt auch das Produkt aus einem Codewort und einem Polynom

$$\big(a(x) \cdot c(x)\big) \in C,\ c(x) \in C,\ a \in \mathbb{F}_q[x]/(x^n - 1)$$

ein gültiges Codewort dar [vgl. 21, S. 121].

Da Ideale eines Polynomrings erzeugende Polynome besitzen, ist die Angabe eines **Generatorpolynoms** $g(x)$ für zyklische Codes möglich, für das gilt:

- $\deg g(x) = n - k$
- $g(x) \mid (x^n - 1)$

Dieses Polynom erzeugt das Ideal des Körpers und damit nach Theorem 5 genau den entsprechenden zyklischen Code.

Theorem 6
Sei C ein linearer zyklischer Code über einem endlichen Körper $\mathbb{F}_q$. Dann gibt es ein Polynom $g(x)$, das als Generator für C fungiert.

Ein Codewort beziehungsweise ein Codewortpolynom $c(x)$ eines linearen, zyklischen Codes mit Generatorpolynom $g(x)$ kann nun als Produkt von Nachrichtpolynom $m(x)$ und Generatorpolynom aufgefasst werden [vgl. 21, S. 121].

$$c(x) = m(x) \cdot g(x)$$

Bei vielen aktuellen Anwendungen von *Reed-Solomon-Codes* werden Generatorpolynome angegeben und genutzt [vgl. 22, S. 6].

3.2 Reed-Solomon-Codes

Die grundlegende Idee von linearen (zyklischen) Codes über Polynomen besteht darin, Fehler über Interpolation zu korrigieren. Bildlich gesehen soll also ein Codewort konstruiert werden, dessen einzelne Komponenten entlang einer interpolierbaren Abbildung aufgereiht sind. Sollte nun ein Übertragungsfehler die Lage eines Punktes (also einer Komponente) verändern, so ist die korrekte Lage durch die anderen Punkte jedoch hinreichend bestimmbar und kann korrigiert werden.

Idee Eine Nachricht der Länge k in Form eines Elementes eines Vektorraums $\mathbb{F}_q^k$ über einem endlichen Körper $\mathbb{F}_q$ kann als Polynom aufgefasst werden, indem die entsprechenden Komponenten der Nachricht die Koeffizienten des Polynoms bilden. Werden nun n paarweise verschiedene Stützstellen aus dem dem Code zugrundeliegenden Körper $\mathbb{F}_q^n$ ausgewählt, so ergibt die Auswertung des Polynoms an den jeweiligen Stützstellen die Komponenten eines betrachteten Codewortes.

Durch bereits k Komponenten des Codewortes ist das Polynom hinreichend bestimmt, sodass $n-k$ Punkte, die während der Übertragung ausgelöscht wurden, und $(n-k)/2$ verfälscht übertragene Punkte als Fehler erkannt und korrigiert werden können.

Definition 22
Sei $u = \langle u_0, u_1, ..., u_{n-1}\rangle \in \mathbb{F}_q^n$, wobei $\forall i, j \in \{0, ..., n-1\}: u_i \neq u_j$.
Dann ist $a = \langle a_0, a_1, ..., a_{n-1}\rangle \in \mathbb{F}_q^n$ ein Codewort eines **Reed-Solomon-Codes** C mit Länge n, Dimension k und Minimaldistanz $d = n - k + 1$, falls es ein Polynom $f(x)$ mit Grad $\deg f < k$ gibt, sodass $f(u_i) = a_i$ ist. Die Menge aller dieser Codeworte a ist C [vgl. 23, S. 9]:

$$C = \{a \mid a_i = f(u_i), \deg f(x) < k, f \in \mathbb{F}_q^n[x]\}$$

Die Polynome f mit den Koeffizienten $f_0, \ldots, f_{k-1}$ können hierbei als Nachrichten aufgefasst werden, während die Auswertungen dieser Polynome an den Auswertungsstellen u_i die korrespondierenden Codeworte ergeben.

Beispiel 8
Betrachtet werde ein $[7, 4]_7$-*Reed-Solomon*-Code mit Länge 7 und Dimension 4 auf dem

endlichen Körper $\mathbb{F}_7$. Dieser Code ist definiert durch

$$C = \{\mathbf{a} \mid a_i = f(u_i), \deg f(x) < 4, f \in \mathbb{F}_7^7[x]\}$$

Dazu wurden 7 paarweise verschiedene Stützstellen gewählt, wobei in $\mathbb{F}_7$ die Wahlfreiheit nur in der Anordnung besteht: $u = \langle 0, 1, 2, 3, 4, 5, 6\rangle \in \mathbb{F}_7^7$.

Für eine Nachricht $m = \langle 0, 1, 2, 1\rangle \in \mathbb{F}_7^4$ ergibt sich die polynomielle Darstellung $m(X) = 0 + 1 \cdot X + 2 \cdot X^2 + 1 \cdot X^3 \in \mathbb{F}_7[X]$. Die Codierung dieser Nachricht folgt nun durch die Auswertung der Stützstellen in m:

Stützstelle u_i	0	1	2	3	4	5	6
Auswertung $m(u_i)$	0	4	4	6	2	5	0

Damit ergibt sich das Codewort $c = \langle 0, 4, 4, 6, 2, 5, 0\rangle \in \mathbb{F}_7^7$.

Bemerkung 8
Typischerweise werden *Reed-Solomon-Codes* betrachtet, deren Auswertungsstellen den Potenzen eines primitiven Elementes α entsprechen. Das Generatorpolynom zu einem solchen *Reed-Solomon-Code* mit maximaler Fehlerkorrekturkapazität $t = \frac{n-k}{2}$ ist definiert durch

$$g(x) = \prod_{j=1}^{2t}(x - \alpha^j)$$

wobei t die Anzahl der durch diesen Code korrigierbaren Fehler beschreibt und α ein primitives Element des zugrundeliegenden Körpers repräsentiert [vgl. 24, S. 17].

3.2.1 Verallgemeinerte Reed-Solomon-Codes

Reed-Solomon-Codes lassen sich verallgemeinern, indem ein zusätzlicher Vektor $v = \langle v_0, ..., v_{n-1}\rangle$ eingebracht wird.

Definition 23
Sei $u = \langle u_0, u_1, ..., u_{n-1}\rangle \in \mathbb{F}_q^n$, wobei $\forall i, j \in \{0, ..., n-1\}\colon u_i, u_j \in \mathbb{F}_q \wedge ((u_i = u_j) \Rightarrow i = j)$. Ferner sei $v = \langle v_0, v_1, ..., v_{n-1}\rangle$, $\forall j \in \{0, ..., n-1\}\colon v_j \in \mathbb{F}_q \setminus \{0\}$. Der

verallgemeinerte Reed-Solomon-Code $GRS_k(u, v)$ besteht nun aus der Menge der Codeworte

$$c = \langle v_0 f(u_0), v_1 f(u_1), ..., v_{n-1} f(u_{n-1}) \rangle$$

für Polynome $f \in \mathbb{F}_q[x]$ mit Grad $\deg f < k$ [vgl. 23, S. 13].

Verallgemeinerte Reed-Solomon-Codes sind im Allgemeinen nicht zyklisch, folglich entfällt die Möglichkeit, ein Generatorpolynom anzugeben. Die Codierung basiert entsprechend auch nicht mehr auf einer Multiplikation des Informationstupels mit dem Generatorpolynom, sondern auf einer Multiplikation mit einer Generatormatrix, die im Folgenden betrachtet wird. Verallgemeinerte Reed-Solomon-Codes werden im Ansatz von NIEDERREITER als dem Verfahren zugrundeliegende Codeklasse vorgeschlagen, da aufgrund der zusätzlich eingebrachten **Gewichte** die Anzahl der möglichen Codes signifikant steigt und die Codeklasse somit an kryptographischer Bedeutung gewinnt.

Für die weiteren Abschnitte dieser Arbeit ist die Betrachtung **dualer Codes** notwendig. Duale Codes können zu linearen Codes gebildet werden:

3.2.2 Duale Codes

Definition 24
Sei C ein linearer $[n, k, d]_q$-Code mit Länge n, Dimension k und Minimaldistanz d über einem Körper $\mathbb{F}_q$. Ferner bezeichne die Operation $\cdot$ die **Paarung** – analog zum Skalarprodukt über reellen Vektorräumen – über dem Vektorraum $\mathbb{F}_q^n$, die definiert ist durch

$$\langle a_0, a_1, ..., a_{n-1} \rangle \cdot \langle b_0, b_1, ..., b_{n-1} \rangle = \sum_{i=0}^{n-1} a_i b_i.$$

Nun ist der zu C **duale Code** $C^\perp$ definiert als

$$C^\perp = \{v \in \mathbb{F}_p^n \mid v \cdot c = 0, \forall c \in C\}$$

[vgl. 11, S. 35f.].

Diese Definition mag an die aus der linearen Algebra im Reellen bekannte Definition von Orthogonalität erinnern, jedoch lässt sich dies auf die hier betrachteten Körper nicht übertragen, unter anderem, da es auch **selbstduale** Codes gibt [vgl. 11, S. 36].

Bemerkung 9
Der zu einem $[n, k, d]_q$-Code C duale Code $C^\perp$ ist ein $[n, n-k, d']_q$-Code [vgl. 11, S. 36].

Wird der duale Code zu einem dualen Code eines linearen Codes C gebildet, so ist dieser wieder C, also gilt:

$$C^{\perp\perp} = C$$

Theorem 7
Der duale Code zu einem verallgemeinerten Reed-Solomon-Code ist ebenfalls ein verallgemeinerter Reed-Solomon-Code [vgl. 23, S. 14].

Beweis. Zunächst wird $k = n-1$ angenommen. Dann ist zu zeigen, dass $GRS_{n-1}(u, v)^\perp = GRS_1(u, v')$ gilt. Dafür genügt es, zu beweisen, dass $v'_i \neq 0$ gilt und v damit ein gültiges Gewichtstupel ist, da für alle u_i bereits bekannt ist, dass sie die Forderungen von GRS-Codes erfüllen. Die Codewortmenge von $GRS_1(u, v')$ ist dadurch, dass zulässige Polynome auf diesem Code nur Skalare sein können, durch alle skalaren Vielfachen von allen v'_i bereits vollständig beschrieben.

Aus Definition 24 ergibt sich nun, dass

$$(hG) \cdot (kv') = 0$$

gelten muss, wobei h eine beliebige Nachricht aus $\mathbb{F}_q$, G die Generatormatrix des $GRS_{n-1}(u, v)$-Codes und k ein beliebiges Element aus $\mathbb{F}_q$ ist. Da h und k Werte ungleich 0 annehmen können, gilt nach dem Nullproduktsatz bereits, dass $G \cdot v' = 0$ gelten muss. Mit Vorausgriff auf Bemerkung 11 lässt sich entsprechend das folgende Gleichungssystem konstruieren [vgl. 25, S. 304]:

$$\begin{aligned} v_0 v'_0 + v_1 v'_1 + \ldots + v_{n-1} v'_{n-1} &= 0 \\ u_0 v_0 v'_0 + u_1 v_1 v'_1 + \ldots + u_{n-1} v_{n-1} v'_{n-1} &= 0 \\ &\vdots \\ u_0^{n-2} v_0 v'_0 + u_1^{n-2} v_1 v'_1 + \ldots + u_{n-1}^{n-2} v_{n-1} v'_{n-1} &= 0 \end{aligned}$$

Dies entspricht in Matrixschreibweise

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ u_0 & u_1 & \cdots & u_{n-1} \\ \vdots & & & \vdots \\ u_0^{n-2} & u_1^{n-2} & \cdots & u_{n-1}^{n-2} \end{bmatrix} \begin{bmatrix} v_0 v_0' \\ v_1 v_1' \\ \vdots \\ v_{n-1} v_{n-1}' \end{bmatrix} = 0. \tag{3.1}$$

Falls ein $v_i' = 0$ ist, so ergibt sich dadurch, dass keine Komponente der linken *Vandermonde*-Matrix gemäß der Definition von GRS-Codes gleich 0 sein kann und, dass alle $v_i v_i' = 0$ und in der Folge alle $v_i' = 0$ sein müssten, was jedoch nicht möglich ist, zum Beispiel da die Dualisierung dieses Codes nicht wieder den $GRS_{n-1}(u, v)$-Code erzeugen kann. Damit ist gezeigt, dass $\forall i \in \{0, \ldots, n-1\}$: $v_i \neq 0$ gilt und zugleich, dass $GRS_{n-1}(u, v)^{\perp}$ die Forderungen eines GRS-Codes erfüllt. Dass sich dieses Resultat nun auch auf GRS-Codes mit beliebigem $k < n-1$ übertragen lässt, ergibt sich nun aus folgender verallgemeinerten Form der obigen Gleichungen [vgl. 25, S. 304]:

$$\sum_{i=0}^{n-1} (u_i^s v_i)(u_i^t v_i') = \sum_{i=0}^{n-1} (u_i^{s+t} v_i v_i') = 0 \tag{3.2}$$

Mit der gleichen Argumentation wie für $k = n-1$ ergibt sich unmittelbar, dass auch hier kein v_i' gleich 0 sein kann. □

Bemerkung 10
Sei $GRS_k(u, v)^{\perp} = GRS_{n-k}(u, v')$. Dann gilt für die Komponenten des Vektors v' [vgl. 23, S. 14]:

$$v_i \cdot v_i' \prod_{j=0,\, j \neq i}^{n-1} (u_j - u_i) = 1$$

Ein Beweis dieser Formel ist in [26, S. 9f.] zu finden.

3.2.3 Kontroll- und Generatormatrizen

Da lineare Codes wie gezeigt Vektorräume bilden, ist die Angabe von **Basen** möglich. Durch die Definition der Dualität von Codes zueinander, können **Generator-** und **Kontrollmatrizen** angegeben werden.

Definition 25
Sei C ein linearer $[n,k,d]_q$-Code und $C^\perp$ der duale Code zu C.
Ferner sei $\left(\langle h_{00}, h_{01}, ..., h_{0(n-1)}\rangle, \langle h_{10}, ..., h_{1(n-1)}\rangle, ..., \langle h_{(n-k-1)0}, ..., h_{(n-k-1)(n-1)}\rangle\right)$ eine Basis zu $C^\perp$. Dann ist die Matrix

$$H = \begin{pmatrix} h_{00} & h_{01} & ... & h_{0(n-1)} \\ h_{10} & h_{11} & ... & h_{1(n-1)} \\ ... & ... & ... & ... \\ h_{(n-k-1)0} & h_{(n-k-1)1} & ... & h_{(n-k-1)(n-1)} \end{pmatrix}$$

eine **Generatormatrix** von $C^\perp$ und **Kontrollmatrix** oder **Paritätsprüfmatrix** von C [vgl. 11, S. 37].

Der Zusammenhang zwischen Generator- und Kontrollmatrix ergibt sich daraus, dass über eine Generatormatrix des dualen Codes $C^\perp$ zu C überprüft werden kann, ob für ein $x \in \mathbb{F}_q$ auch $x \in C$ gilt, indem die Paarungen $x \cdot h_i, \forall i \in \{0, ..., n-k-1\}$ gebildet werden. Sollte für mindestens eines dieser Produkte $x \cdot h_i \neq 0$ gelten, so ist $x \notin C$ [vgl. 11, S. 37]. So ließe sich beispielsweise eine Identitätsabbildung $1_C(x)$ auf C erzeugen.

Generatormatrizen des verallgemeinerten Reed-Solomon-Codes

Zur Kodierung eines Informationstupels mittels des verallgemeinerten Reed-Solomon-Codes wird eine Generatormatrix des Codes benötigt.

Bemerkung 11
Gegeben sei $GRS_k(u,v)$ und der dazu duale Code $GRS_{n-k}(u,y)$. Eine Paritätsprüfmatrix zu $GRS_k(u,v)$ ist dann genau eine Generatormatrix zu $GRS_{n-k}(u,y)$ und definiert durch

$$H = (y_i u_i^j)_{0 \leq j \leq n-k-1, 0 \leq i \leq n-1} = \begin{pmatrix} y_0 & y_1 & \cdots & y_{n-1} \\ y_0 u_0 & y_1 u_1 & \cdots & y_{n-1} u_{n-1} \\ \vdots & \vdots & & \vdots \\ y_0 u_0^{n-k-1} & y_1 u_1^{n-k-1} & \cdots & y_{n-1} u_{n-1}^{n-k-1} \end{pmatrix}$$

Entsprechend ist die Generatormatrix zu $GRS_k(u, v)$ definiert durch

$$G = (v_i u_i^j)_{0 \leq j \leq k-1, 0 \leq i \leq n-1} = \begin{pmatrix} v_0 & v_1 & \cdots & v_n \\ v_0 u_0 & v_1 u_1 & \cdots & v_{n-1} u_{n-1} \\ \vdots & \vdots & & \vdots \\ v_0 u_0^{k-1} & v_1 u_1^{k-1} & \cdots & v_{n-1} u_{n-1}^{k-1} \end{pmatrix}$$

[vgl. 21, S. 277] [vgl. 25, S. 304].

Diese Matrizen sind für die Kodierung und Dekodierung – insbesondere auch für die beabsichtigte kryptographische Verwendung – von besonderer Bedeutung, da mit ihrer Hilfe Codes definiert werden können und die Frage, ob ein Wort Codewort dieses Codes ist, nun entscheidbar wird. Generatormatrizen lassen sich durch elementare Zeilen- und Spaltenoperationen verändern, sodass zwar weiterhin gültige Codeworte eines Codes C gebildet werden, jedoch wird die Projektion des Informationstupels auf ein Codewort dadurch beeinflusst, weshalb womöglich andere Codeworte entstehen. Diese Eigenschaft wird genutzt, um beispielsweise eine systematische Codierung zu ermöglichen.

3.3 Kodierung, Fehlerkorrektur und Dekodierung

In diesem Abschnitt wird mit der algorithmischen Betrachtung der Kodierung und Dekodierung samt der Fehlerkorrektur verallgemeinerter Reed-Solomon-Codes die Grundlage dafür gelegt, ein kryptographisches Verfahren auf Basis fehlerkorrigierender Codes zu konstruieren, wie es McEliece und Niederreiter in ihren Ansätzen tun.

Sämtliche Listings in diesem Abschnitt zeigen Implementierungen im auf Python basierenden Mathematiksoftwaresystem *SageMath* [27]. Für die Ausführung dieser Skripte ist ein (gegebenenfalls auch virtualisiertes) Linux-Betriebssystem erforderlich.

3.3.1 Kodierung

Bei der Codierung linearer Codes im Allgemeinen werden zwei Ansätze unterschieden:

- **nicht-systematische Codierung** – Hier ergeben sich die Codeworte direkt aus der Multiplikation von Informationstupel und einer beliebigen Generatormatrix des Codes.
- **systematische Codierung** – Hier wird eine Generatormatrix der Form $[I_k \mid C]$ verwendet, wobei I die $k \times k$-Identitätsmatrix und C die $k \times (n-k)$-Codeworterzeugungsmatrix ist. Eine solche Generatormatrix kann durch elementare Zeilen- und Spaltenoperationen erzeugt werden [vgl. 21, S. 84f.].

Nicht-Systematische Kodierung

Sei $h = \langle h_0, ..., h_{k-1} \rangle \in \mathbb{F}_q^k$ ein Informationstupel und $GRS_k(a, v)$ mit $a = \langle a_0, ..., a_{n-1} \rangle$, $\forall i, j \in \{0, ..., n-1\}: a_i, a_j \in \mathbb{F}_q \wedge (a_i = a_j \Rightarrow i = j)$ und $v = \langle v_0, ..., v_{n-1} \rangle$, $\forall i \in \{0, ..., n-1\}: v_i \in \mathbb{F}_q \setminus \{0\}$ ein verallgemeinerter Reed-Solomon-Code.

Dann ist eine Generatormatrix zu $GRS_k(a, v)$ definiert als

$$G = \begin{pmatrix} v_0 & v_1 & \cdots & v_{n-1} \\ v_0 a_0 & v_1 a_1 & \cdots & v_{n-1} a_{n-1} \\ \vdots & \vdots & & \vdots \\ v_0 a_0^{k-1} & v_1 a_1^{k-1} & \cdots & v_{n-1} a_{n-1}^{k-1} \end{pmatrix}$$

Die nicht-systematische Codierung ist nun gegeben durch die Zeilenvektor-Matrix-Multiplikation von h und G [vgl. 28, S. 37f]:

$$\begin{aligned} c = h \cdot G &= \begin{pmatrix} h_0 & \ldots & h_{k-1} \end{pmatrix} \cdot \begin{pmatrix} v_0 & v_1 & \cdots & v_{n-1} \\ v_0 a_0 & v_1 a_1 & \cdots & v_{n-1} a_{n-1} \\ \vdots & \vdots & & \vdots \\ v_0 a_0^{k-1} & v_1 a_1^{k-1} & \cdots & v_{n-1} a_{n-1}^{k-1} \end{pmatrix} \\ &= \begin{pmatrix} h_0 v_0 + h_1 v_0 a_0 + \ldots + h_{k-1} v_0 a_0^{k-1} & \cdots & h_0 v_{n-1} + h_1 v_{n-1} a_{n-1} + \ldots + h_{k-1} v_{n-1} a_{n-1}^{k-1} \end{pmatrix} \\ &= \langle c_0, \ldots, c_{n-1} \rangle \end{aligned}$$

Systematische Codierung

Wie bereits beschrieben, ist es für eine systematische Codierung lediglich erforderlich, die Generatormatrix durch Zeilenoperationen in eine systematische Form zu bringen. Dies kann durch den GAUSS'SCHEN Eliminationsalgorithmus mit Pivotierung erreicht werden [vgl. 21, S. 87f].

Nun liegt die Generatormatrix in folgender Form vor:

$$G_S = \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} \\ 0 & 1 & \cdots & 0 & p_{1,0} & p_{1,1} & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{array}\right)$$

Die systematische Codierung bewirkt nun, dass das Informationstupel unverändert die ersten k Stellen des Codewortes ergibt. Die restlichen Stellen sind $n-k$ Paritätsprüfbits, die über die rechte $k \times (n-k)$-Untermatrix der Generatormatrix erzeugt werden:

$$\begin{aligned} c = h \cdot G_S &= \begin{pmatrix} h_0 & \ldots & h_{k-1} \end{pmatrix} \cdot \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} \\ 0 & 1 & \cdots & 0 & p_{1,0} & p_{1,1} & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{array}\right) \\ &= \left(\begin{array}{cccc|ccc} h_0 & h_1 & \cdots & h_{k-1} & \sum_{i=0}^{k-1} h_0 p_{i,0} & \cdots & \sum_{i=0}^{k-1} h_{k-1} p_{i,n-k-1} \end{array}\right) \\ &= \langle h_0, h_1, \ldots, h_{k-1}, c'_k, c'_{k+1}, \ldots, c'_{n-1} \rangle \end{aligned}$$

Implementierung

Im Anhang A.1 ist die Implementierung des Kodiervorgangs eines verallgemeinerten Reed-Solomon-Codes über endlichen Körpern in *SageMath* dargestellt. Das Skript generiert aus den Stützstellen und Gewichten eine Generatormatrix für die nicht-systematische Kodierung. Die Generatormatrix für die systematische Kodierung wird dadurch erzeugt, dass die $k \times k$-Untermatrix der ersten k Spalten invertiert und mit G multipliziert wird. Referenzen und alternative Algorithmen sind dem Anhang A.2 zu entnehmen.

3.3.2 Dekodierung

Nachdem (nicht-)systematisch erzeugte Codeworte nun über einen unzuverlässigen Kanal übertragen wurden, werden Fehlerkorrektur- und Dekodieralgorithmen angewandt, um die übertragenen Informationen zu erhalten.

Im Folgenden wird der **syndrombasierte** Dekodieralgorithmus für eine maximale Fehleranzahl von $t = \lfloor \frac{n-k}{2} \rfloor$ dargestellt. Der alternative Ansatz des **listen-basierten** Dekodierens, der die Korrektur von einer größeren Anzahl an Fehlern ermöglicht, wird anschließend ohne vertiefte Darstellung erwähnt.

Syndrombasierter Dekodieralgorithmus

Gegeben sei ein verallgemeinerter Reed-Solomon-Code $GRS_k(u, v)$ mit

- Auswertungsstellen $u = \langle u_0, ..., u_{n-1} \rangle$, $\forall i, j \in \{0, ..., n-1\} \colon u_i, u_j \in \mathbb{F}_q \wedge ((u_i = u_j) \Rightarrow i = j)$
- Gewichten $v = \langle v_0, ..., v_{n-1} \rangle$, $\forall i \in \{0, ..., n-1\} \colon v_i \in \mathbb{F}_q \setminus \{0\}$

Nach einem Datenübertragungsvorgang wurde ein Tupel $r = \langle r_0, r_1, ..., r_{n-1} \rangle$ empfangen, das sich aus dem korrekten **Codeworttupel** $c = \langle c_0, c_1, ..., c_{n-1} \rangle$ und einem **Fehlertupel** $e = \langle e_0, e_1, ..., e_{n-1} \rangle$ zusammensetzt. Dabei darf die Anzahl der Komponenten von e, die nicht gleich 0 sind, nicht größer als $t = \lfloor \frac{n-k}{2} \rfloor$ sein, andernfalls ist es nicht möglich, mit diesem Dekodieralgorithmus das korrekte Codewort aus r zu ermitteln. Im Folgenden bezeichne

- $k_0, ..., k_{\epsilon-1}$ die **Fehlerstellen** im empfangenen Wort, wobei $\epsilon \leq t$ gilt
- e_{k_j} die **Fehlergröße** an der Stelle k_j.

Aus diesen Angaben lässt sich gemäß Abschnitt 3.2.3 eine Generatormatrix bestimmen. Für dieses Dekodierverfahren ist jedoch die passende Paritätsprüfmatrix erforderlich, da mit ihr ermittelt werden kann, ob ein gegebenes Wort ein gültiges Wort des betrachteten Codes ist [vgl. 28, S. 180] [vgl. 29, S. 184]:

$$H_{C_{GRS}} \cdot r^T = 0 \;\Rightarrow\; r \in C_{GRS}$$

Verfahren zur Ermittlung der Paritätsprüfmatrix Ist eine Generatormatrix in systematischer Form gegeben, so kann dazu eine Paritätsprüfmatrix wie folgt ermittelt werden [vgl. 30, S. 5]:

$$G = [I_k \mid A] \;\Rightarrow\; H = [-A^T \mid I_{n-k}]$$

Hier bezeichne I_j eine $j \times j$-Identitätsmatrix. Wie eine Generatormatrix in systematische Form gebracht werden kann, ist in Anhang A.2 dargestellt.

Eine explizite Formel zur Berechnung der Koeffizienten der Paritätsprüfmatrix (und damit der Gewichte d_i des zum betrachteten Codes dualen Codes) folgt aus diesem Zusammenhang [vgl. 23, S. 14] [vgl. 26, S. 9f]:

$$d_i = v_i^{-1} \cdot \prod_{j=0, j\neq i}^{n-1} (u_i - u_j)^{-1}$$

Die Basis des Dekodieralgorithmus bilden nun $n-k-1$ Syndrome, die die Komponenten des Produkts dieser Paritätsprüfmatrix H mit dem empfangenen Wort r ergeben:

Definition 26
Das l-te **Syndrom** eines Wortes r sei definiert als

$$S_l = \sum_{j=0}^{n-1} r_j \cdot d_j \cdot u_j^l$$

[vgl. 29, S. 184].

Da zweifelsfrei für jedes Codewort-Syndrom $\sum_{j=0}^{n-1} c_j d_j u_j^l = 0$ gilt, sind die Fehlerwortsyndrome äquivalent zu den Syndromen obiger Definition. Ferner gilt sogar $S_l = \sum_{j=0}^{\epsilon-1} e_{k_j} d_{k_j} u_{k_j}^l$, da die Fehlergröße e_j an nicht fehlerhaften Stellen gleich 0 sein muss. Die Betrachtung dieser Syndrome gibt Aufschluss darüber, an welchen Stellen Fehler während der Übertragung aufgetreten sind.

Durch die Syndrome ergibt sich folgendes Gleichungssystem:

$$\begin{aligned}
S_0 &= e_0 d_0 + e_1 d_1 + \ldots + e_{n-1} d_{n-1} \\
S_1 &= e_0 d_0 u_0 + e_1 d_1 u_1 + \ldots + e_{n-1} d_{n-1} u_{n-1} \\
S_2 &= e_0 d_0 u_0^2 + e_1 d_1 u_1^2 + \ldots + u_{n-1} d_{n-1} u_{n-1}^2 \\
&\vdots \\
S_{n-k-1} &= e_0 d_0 u_0^{n-k-1} + e_1 d_1 u_1^{n-k-1} + \ldots + e_{n-1} d_{n-1} u_{n-1}^{n-k-1}
\end{aligned}$$

Da dieses Gleichungssystem jedoch angesichts der Potenzen nicht linear ist, lässt sich ohne großen Rechenaufwand keine triviale Lösung bestimmen, die jedoch für die Berechnung der Fehlerpositionen und Fehlergrößen notwendig wäre [vgl. 21, S. 248] [vgl. 28, S. 180f] [vgl. 31, S. 148].

Stattdessen wird ein **Fehlerlokalisationspolynom** definiert.

$$\sigma(x) = \prod_{k_j} x - u_{k_j} = \sum_{i=0}^{\epsilon-1} \sigma_i \cdot x^i$$

Aus der Definition folgt unmittelbar, dass jede Wurzel dieses Polynoms eine Fehlerstelle ist [vgl. 28, S. 180f] [vgl. 29, S. 185] [vgl. 32, S. 28f.].

Ziel ist es nun, die Unbekannten σ_j zu bestimmen, um damit die Fehlerstellennummern ermitteln zu können. Dafür wird das Fehlerlokalisationspolynom mit dem durch die Syndrome erzeugten Gleichungssystem in Verbindung gesetzt [vgl. 28, S. 181f.] [vgl. 31, S. 148f.], indem für die Variable x das Syndrom S_{j+l} eingesetzt wird. So ergibt sich, dass für alle nicht-fehlerhaften Stellen des betrachteten Wortes die Fehlergröße $e_i = 0$ und für alle fehlerhaften Stellen das Fehlerlokalisationspolynom $\sigma(u_{k_i}) = 0$ ist, weshalb die Koeffizienten des Fehlerlokalisationspolynoms dieses Gleichungssystem erfüllen [vgl. 32, S. 28f.].

$$\sum_{l=0}^{\epsilon-1} \sigma_l \cdot S_{j+l} = 0 \qquad \forall j \in \{0, \ldots, \epsilon - 1\}$$

Ferner ist in [32] bewiesen, dass dieses Gleichungssystem die Koeffizienten σ_j des Fehlerlokalisationspolynoms eindeutig bestimmt [vgl. 32, S. 28f.].

Zusammengefasst wird eine nicht-triviale Lösung des durch diese folgende Matrix beschrie-

benen Gleichungssystems gesucht [vgl. 28, S. 181f.]:

$$\begin{bmatrix} S_0 & S_1 & \cdots & S_t \\ S_1 & S_2 & \cdots & S_{t+1} \\ \vdots & & & \vdots \\ S_{n-k-t-1} & S_{n-k-t} & \cdots & S_{n-k-1} \end{bmatrix} \cdot \begin{bmatrix} \sigma_0 \\ \sigma_1 \\ \vdots \\ \sigma_t \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.3}$$

Aus dieser Lösung ergibt sich durch Einsetzen der Stützstellen die Menge der Fehlerstellen $\{k_i \mid e_{k_i} \neq 0\}$. Um nun die Fehlergrößen zu diesen Fehlerstellen zu ermitteln, wird erneut die Eigenschaft ausgenutzt, dass alle Syndrome eines korrekten Codewortes gleich 0 sind und es somit genügt, die Fehlerwortsyndrome zu betrachten. Werden nun die Syndrome für die Fehlerstellen berechnet, ergibt sich die Größe des Fehlers, also den abzuziehenden Abstand des Syndroms von 0. Konkret wird also das folgende Gleichungssystem konstruiert:

$$\begin{bmatrix} d_{k_0} & d_{k_1} & \cdots & d_{k_{\epsilon-1}} \\ d_{k_0} \cdot u_{k_0} & d_{k_1} \cdot u_{k_1} & \cdots & d_{k_{\epsilon-1}} \cdot u_{k_{\epsilon-1}} \\ \vdots & & & \vdots \\ d_{k_0} \cdot u_{k_0}^{n-k-1} & d_{k_1} \cdot u_{k_1}^{n-k-1} & \cdots & d_{k_{\epsilon-1}} \cdot u_{k_{\epsilon-1}}^{n-k-1} \end{bmatrix} \cdot \begin{bmatrix} e_{k_0} \\ e_{k_1} \\ \vdots \\ e_{k_{\epsilon-1}} \end{bmatrix} = \begin{bmatrix} S_0 \\ S_1 \\ \vdots \\ S_{n-k-1} \end{bmatrix} \tag{3.4}$$

Zusammen mit $e_j = 0$ für alle fehlerfreien Stellen ergibt sich somit das Fehlergrößentupel $e = \langle e_0, e_1, \ldots, e_{n-1} \rangle$, das nun vom empfangenen Wort abgezogen wird, um das korrekte Codewort zu erhalten [vgl. 28, S. 182f.] [vgl. 31, S. 150]:

$$\langle c_0, c_1, \ldots, c_{n-1} \rangle = \langle r_0, r_1, \ldots, r_{n-1} \rangle - \langle e_0, e_1, \ldots, e_{n-1} \rangle$$

Der hier vorgestellte Algorithmus folgt den in [28] und [32] beschriebenen Schritten und wird als *Peterson-Gorenstein-Zierler*-Algorithmus bezeichnet.

Implementierung

Im Anhang A.3 ist eine Implementierung des vorgestellten Dekodieralgorithmus für GRS_k-Codes über endlichen Körpern im Mathematiksoftwaresystem *SageMath* [27] zu finden. Um das Gewichtstupel $d = \langle d_0, d_1, \ldots, d_{n-1} \rangle$ aus den gegebenen Parametern des Codes zu ermitteln, wurde die obig beschriebene Formel aus [26] bzw. [23] verwendet, da die Extraktion der Werte d_i aus einer systematischen Paritätsprüfmatrix einen wesentlich hö-

heren Berechnungsaufwand nach sich zöge.

Da in *SageMath* lineare Gleichungssysteme über endlichen Körpern keine symbolischen Variablen enthalten können, wurden die Gleichungssysteme stets in Matrixform angegeben. Um nun die triviale Lösung der Fehlerstellengleichung auszuschließen, wurde die Methode `right_kernel()` verwendet, die den Vektorraum W der Lösungen w der Gleichung $A \cdot w = 0$ zurückgibt [vgl. 27, „Linear Algebra"]. Da die aus den Syndromen bestehende Matrix in Gleichung 3.3 (hier bezeichnet als M) nach [28, S. 181f.] nicht-singulär und damit invertierbar ist, folgt, dass $\det M \neq 0$ gilt und damit der triviale Kern kein Kern der Matrix M ist. Somit erscheint die Verwendung dieser Funktion anstelle der Funktion `solve_right()` probat zur Lösung der Problemstellung.

Weitere Dekodieralgorithmen

Der dargestellte Dekodieralgorithmus, der ein Codewort eines $GRS_k(\alpha, v)$-Codes mit bis zu $t = \lfloor \frac{n-k}{2} \rfloor$ Fehlern über die Berechnung von Syndromen eindeutig dekodieren kann, wurde insbesondere durch die Arbeiten von BERLEKAMP und MASSEY weiterentwickelt. So konnte die Bestimmung der Fehlerstellen durch die Gleichungen der *Newton-Identitäten* performanter realisiert werden [vgl. 28, S. 186f.]. Eine weitere hier ansetzende Alternative wurde von SUGIYAMA publiziert, die zur Bestimmung des Fehlerlokalisationspolynoms den *Euklidischen Algorithmus* verwendet [vgl. 28, S. 190f.].

Abweichend zum Ansatz dieser syndrombasierten Dekodieralgorithmen entwickelten SUDAN und GURUSWAMI **listenbasierte** Dekodierverfahren, die im Allgemeinen kein eindeutiges korrektes Codewort, sondern eine Anzahl möglicher Codeworte zurückgeben. Diese Verfahren können empfangene Worte mit einer größeren Fehleranzahl als t dekodieren [vgl. 33, S. 96f.], indem das GRS_k-Codes zugrundeliegende Interpolationsproblem durch Parameterschätzung gelöst wird [vgl. 33, S. 100f.].

4 Kryptographische Verfahren auf Basis fehlerkorrigierender Codes

Dass fehlerkorrigierende Codes wie GRS_k-Codes für kryptographische Zwecke genutzt werden können, ergibt sich durch die Definition *code-basierter Kryptographie* von SENDRIER wie folgt:

> „Code-based cryptography includes all cryptosystems, symetric or asymetric, whose security relies, partially or totally, on the hardness of decoding in a linear error correcting code, possibly chosen with some particular structure or in a specific family (for instance, quasi-cyclic codes, or Goppa codes)." [34, S. 215]

Die Sicherheit dieser Verfahren liegt folglich darin begründet, dass es unter bestimmten Bedingungen bzw. bei der Wahl entsprechender Parameter hinreichend schwierig ist, das Dekodierproblem linearer fehlerkorrigierender Codes zu lösen.

In diesem Kapitel werden die Ansätze dieser Verfahren zunächst allgemein dargestellt, bevor die Arbeit von NIEDERREITER genutzt wird, um ein Kryptosystem auf Basis von GRS_k-Codes zu konstruieren.

4.1 Grundlagen und Hintergründe

In Abbildung 4.1 sind einige für den Kontext dieser Arbeit relevante Publikationen zu code-basierten Kryptosystemen in zeitlicher Abfolge, basierend auf dem Publikationsjahr, dargestellt. Die Auswahl orientiert sich einerseits an der Themeneingrenzung dieser Arbeit und andererseits am thematischen Überblick in [38]. Ebenda wird das durch MCELIECE vorgeschlagene Kryptosystem [2] als das erste code-basierte Kryptosystem bezeichnet [vgl. 38, S. 96]. Dieses Verfahren basiert nicht wie zuvor publizierte Verfahren primär auf dem *Rucksackproblem*, sondern auf algebraischen Codes [vgl. 2, S. 114] und deren Dekodierproblem [vgl. 38, S. 98].

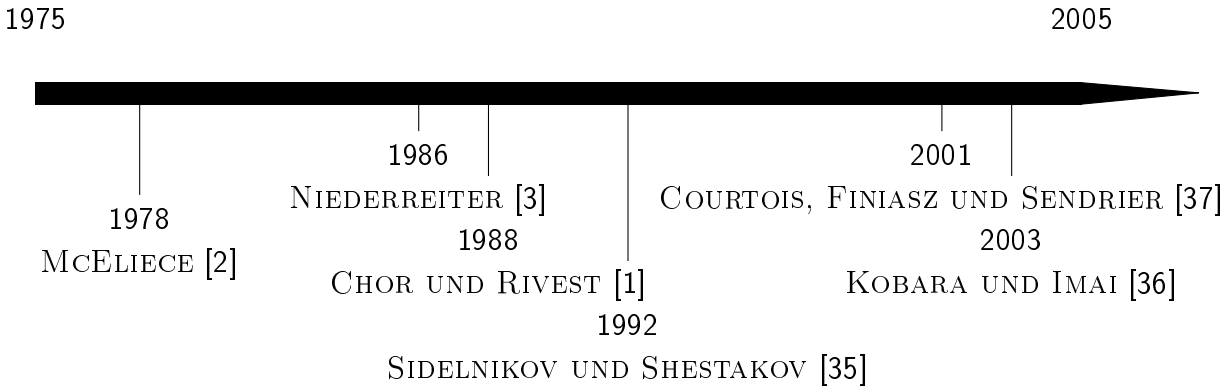

Abbildung 4.1: Chronologische Einordnung ausgewählter Beiträge der code-basierten Kryptographie

4.1.1 PKC

Grundlage aller im Folgenden betrachteten Verfahren sind *Public-Key-Cryptosystems*, wie sie von Diffie und Hellman vorgestellt wurden:

Definition 27
Sei M eine endliche Menge an Nachrichten, C eine endliche Menge dazugehöriger Chiffrate und K die Bildmenge eines Schlüsselpaarerzeugungsalgorithmus. Dann ist ein **Public-Key-Kryptosystem** ein Paar zweier invertierbarer Algorithmen

$$E_k \colon M \to C$$
$$D_k \colon C \to M,$$

für das die folgenden Bedingungen gelten muss:

- Für jedes $k \in K$ gilt: $(E_k)^{-1} = D_k$.
- Für jedes $k \in K$ und jede Nachricht $m \in M$ respektive jedes Chiffrat $c \in C$ gilt: E_k und D_k sind einfach zu berechnen.
- Für nahezu jedes $k \in K$ muss es (unter verhältnismäßigem Ressourceneinsatz) unmöglich sein, D_k aus einem gegebenen E_k zu berechnen.
- Für ein gegebenes $k \in K$ muss es möglich sein, das inverse Paar E_k und D_k zu bestimmen.

[vgl. 39, S. 647f.]

Im Vergleich zu **symmetrischen Verfahren**, bei denen zur Ver- und Entschlüsselung jeweils derselbe Schlüssel verwendet werden muss, liegt der große Vorteil von *PKCs* darin, dass der Verschlüsselungsschlüssel E_k veröffentlicht werden kann, ohne die Sicherheit des Kryptosystems zu gefährden. Zu beachten ist jedoch, dass die Integrität öffentlicher Schlüssel gewährleistet werden muss [vgl. 39, S. 648].

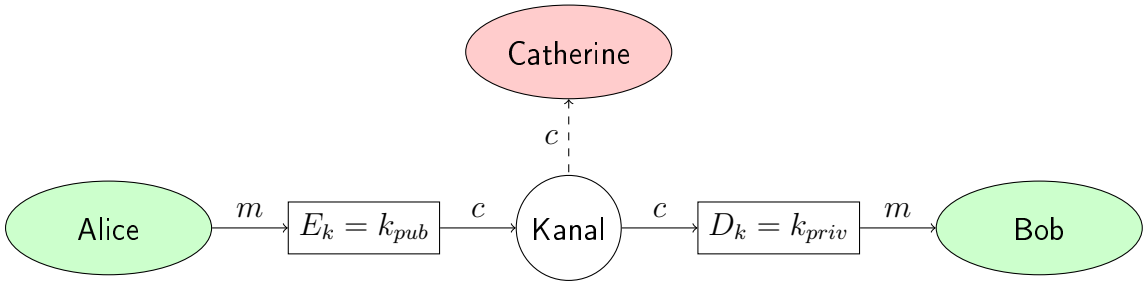

Abbildung 4.2: Schematische Darstellung eines *PKC* (nach [39, S. 647])

Public-Key-Cryptosystems lassen sich nach ihrem zugrundeliegenden Problem klassifizieren, wobei Chor und Rivest in [1] folgende Klassen unterscheiden:

- *PKCs* auf Basis schwerer zahlentheoretischer Probleme (beispielsweise das RSA-Verfahren, das auf der Primfaktorzerlegung basiert)
- *PKCs*, deren Sicherheit auf dem *Rucksackproblem* basiert

Das McEliece-Kryptosystem lässt sich keiner der beiden Kategorien zuordnen, da es auf dem Dekodierproblem fehlerkorrigierender Codes basiert [vgl. 1, S. 901].

4.1.2 Rucksackproblem

Das Rucksackproblem (im Englischen *knapsack problem*) beschreibt ein $\mathcal{NP}$-hartes (siehe Bemerkung 13) Entscheidungsproblem, das darauf basiert, zu einer endlichen Menge von Elementen die größtmögliche Summe von (binär) gewichteten Elementen dieser Menge zu finden, die nicht größer als ein vorher definierter Wert ist [vgl. 1, S. 902] [vgl. 40, S. 127]. Anwendungen dieses Problems ergeben sich beispielsweise bei der Beladung eines Containers: Das Volumen des Containers ist der nicht zu überschreitende Wert und gesucht wird eine Auswahl an (auch mehrfach wählbaren) Gütern, die den Stauraum des Containers maximal ausnutzt [vgl. 40, S. 127].

Im Kontext kryptographischer Anwendungen wird folgende Definition genutzt:

Definition 28
Sei $A = \{a_0, a_1, \ldots, a_{n-1}\}$ eine Menge mit $\forall j \in \{0, \ldots, n-1\}: a_j \in \mathbb{N}$ und $S \in \mathbb{N}$. Das **0-1-Rucksackproblem** bezeichnet die Frage, ob eine ganzzahlige Lösung der Form

$$\sum_{i=0}^{n-1} x_i a_i = S \qquad (x_i \in \{0,1\})$$

existiert [vgl. 1, S. 902].

Eine verallgemeinerte Variante dieses Problems ergibt sich, wenn anstelle der Restriktion $x_i \in \{0,1\}$ ein **Gewicht** h definiert wird, sodass $\sum x_i \leq h$ gelten muss, damit das Problem als gelöst gilt [vgl. 1, S. 902].

CHOR und RIVEST konstruieren darauf nun ein kryptographisches Verfahren, wie in Algorithmus 1. Die Komponenten a_i werden je nach Kryptosystem so ausgewählt, dass in

Algorithmus 1 Verschlüsselungsalgorithmus eines Rucksack-Typ-*PKCs* (nach [1, S. 902])

Require: $a = \langle a_0, a_1, \ldots, a_{n-1} \rangle,\ h,\ m = \langle m_0, m_1, \ldots, m_{n-1} \rangle$ mit $\sum_{i=0}^{n-1} m_i \leq h$
  $c \leftarrow 0$
  for $j \in \{0, \ldots, n-1\}$ **do**
    $c \leftarrow c + (m[j] * a[j])$
  end for
  return c

Kenntnis gewisser geheimer Zusatzinformationen (auch genannt *Hintertür* bzw. *trapdoor*) die Gleichung leicht gelöst, respektive das Codewort c dekodiert, werden kann [vgl. 1, S. 902].

Die Arbeit von CHOR und RIVEST bildet die Grundlage für das NIEDERREITER-Schema[1] [vgl. 3, S. 159]. Das von ihnen vorgeschlagene Kryptosystem ist definiert über einem endlichen Erweiterungskörper $\mathbb{F}_{p^f}$, in dem Berechnungen zum **diskreten Logarithmus** möglich sind [vgl. 1, S. 903].

Bemerkung 12
Sei $p \in \mathbb{P}$ und $f \in \mathbb{N}$ dergestalt, dass $\mathbb{F}_{p^f}$ einen endlichen Körper der Charakteristik p

[1] Dies ist möglich, da die Arbeit [1] bereits vier Jahre vor ihrem Erscheinen im Jahr 1988 im Rahmen der *CRYPTO '84* präsentiert wurde. NIEDERREITER referenziert 1986 in [3] eine Preprint-Fassung.

bezeichne. Ferner sei α ein primitives Element dieses Körpers. Dann sind die Funktionen

$$y = \alpha^x \tag{4.1}$$

$$x = \log_\alpha(y) \tag{4.2}$$

in $\mathbb{F}_{p^f}$ invers zueinander. (4.1) bezeichnet die Exponentialfunktion und (4.2) die Logarithmusfunktion zur Basis α. Während für die Exponentiation maximal $2 \cdot \lceil \log_2 p \rceil$ Multiplikationen erforderlich sind [vgl. 41, S. 106],[2] sind es für das Logarithmieren wesentlich mehr. Dadurch ergibt sich das **diskrete Logarithmusproblem** über endlichen Körpern, das die Schwierigkeit beschreibt, Logarithmen in endlichen Körpern großer Charakteristik zu lösen [vgl. 41, S. 106f.].

Dieses Problem wird in vielen kryptographischen Verfahren wie dem *Diffie-Hellman-Schlüsselaustausch* oder dem *RSA-Kryptosystem* ausgenutzt, da nach wie vor keine effizienten Algorithmen für dieses Problem in Körpern großer Charakteristik implementierbar sind [vgl. 1, S. 904]. Solange die Parameter der Kryptosysteme an die gegenwärtig zur Verfügung stehende Rechenleistung angepasst werden, ist die Sicherheit der Verfahren gewährleistet [vgl. 43].

4.2 McEliece-Kryptosystem

Grundlage des McEliece-Kryptosystems sind *Goppa*-Codes, benannt nach Valery Denisovich Goppa.

Definition 29
Sei $p \in \mathbb{P}$ und $f \in \mathbb{N}$ so, dass $\mathbb{F}_{p^f}$ einen endlichen Körper der Charakteristik p bezeichne. Ferner sei $g(z)$ ein Polynom mit Koeffizienten aus $\mathbb{F}_{p^f}$ und L eine (meist unechte) Teilmenge aller Elemente aus $\mathbb{F}_{p^f}$, die nicht Wurzel von $g(z)$ sind. Dann existiert ein **Goppa-Code** mit Eingabemenge $\mathbb{F}_q$, Wertemenge $\mathbb{F}_{q^f}$, Goppapolynom $g(z)$ und Länge $|L|$. Der Code ist nun definiert durch die Menge aller Vektoren C, deren Komponenten – indiziert durch die Elemente der Menge L – folgende Bedingung erfüllen [vgl. 44, S. 590]:

$$\sum_{\gamma \in L} \frac{C_\gamma}{z - \gamma} \equiv 0 \mod g(z)$$

[2] Die Aussage in [41] bezieht sich auf einen Körper $\mathbb{F}_p$, wobei für Erweiterungskörper $\mathbb{F}_{p^h}$ von einer ähnlichen Komplexitätsklasse ausgegangen werden kann [vgl. 42].

Verwendet werden im Folgenden binäre *Goppa*-Codes mit Körper $\mathbb{F}_{2^f}$, $L = \{\epsilon_0, \epsilon_1, \ldots, \epsilon_{N-1}\}$, Länge $|L| = 2^f$ und über $\mathbb{F}_{2^f}$ irreduziblem Goppapolynom $g(z)$ mit Grad t, wobei t die maximale Fehlerkorrekturkapazität angibt. Die Dimension des Codes ist definiert durch $k \geq n - t \cdot f$ [vgl. 2, S. 114].

Wurden passende Parameter n, t gewählt und ein irreduzibles Polynom gefunden – die Wahrscheinlichkeit dafür, dass ein zufällig gewähltes Polynom irreduzibel ist, beträgt $\frac{1}{t}$ [vgl. 2, S. 114] – kann eine $k \times n$-Generatormatrix G erzeugt werden. Hierfür wird zunächst eine Paritätsprüfmatrix

$$H = \begin{bmatrix} \frac{1}{g(\epsilon_0)} & \frac{1}{g(\epsilon_1)} & \cdots & \frac{1}{g(\epsilon_{N-1})} \\ \frac{\epsilon_0}{g(\epsilon_0)} & \frac{\epsilon_1}{g(\epsilon_1)} & \cdots & \frac{\epsilon_{N-1}}{g(\epsilon_{N-1})} \\ \frac{\epsilon_0^{t-1}}{g(\epsilon_0)} & \frac{\epsilon_1^{t-1}}{g(\epsilon_1)} & \cdots & \frac{\epsilon_{N-1}^{t-1}}{g(\epsilon_{N-1})} \end{bmatrix}$$

gebildet [vgl. 45, S. 3] und anschließend invers zum in 3.3.2 genannten Vorgehen eine Generatormatrix ermittelt.

Schlüsselerzeugung Zur Erzeugung des öffentlichen Schlüssels des McEliece werden folgende Schritte ausgeführt [vgl. 2, S. 114]:

1. Wahl einer nicht-singulären und dichten (das heißt, für die Mehrzahl ihrer Komponenten s_{ij} gilt: $s_{ij} \neq 0$) $k \times k$-Matrix S
2. Wahl einer $n \times n$-Permutationsmatrix P, wobei eine Matrix dann eine Permutationsmatrix ist, wenn in jeder Zeile und jeder Spalte nur genau eine Komponente p_{ij} mit $p_{ij} = 1$ vorhanden ist und alle übrigen Komponenten 0 sind.
3. Der öffentliche Schlüssel k_{pub} ist nun gegeben durch $k_{pub} = G' = S \times G \times P$.

Der Verschlüsselungsalgorithmus folgt damit nun unmittelbar:

Algorithmus 2 Verschlüsselungsalgorithmus des McEliece-Kryptosystems (nach [2])

Require: G', Eingabedaten m mit l Komponenten aus $\mathbb{F}_{2^f}$
1: Teile m in Blöcke der Länge k (ggf. mit Nullen auffüllen)
2: **for** u in $m[]$ **do**
3: Wähle zufälligen Störvektor z mit Länge n und Gewicht t
4: $x_u \leftarrow u \cdot G' + z$
5: **end for**
6: **return** x

Für ein übertragenes Wort x erfolgt die Dechiffrierung nun wie folgt:

Algorithmus 3 Entschlüsselungsalgorithmus des McEliece-Kryptosystems (nach [2])

Require: Übertragenes Wort x, *Goppa*-Code-Parameter, P, S
1: $(x' \leftarrow x \cdot P^{-1}) \Rightarrow x' \in C$
2: $u' \leftarrow$ Ergebnis des Dekodieralgorithmus von Patterson auf x'
3: $u = u' \cdot S^{-1}$
4: **return** u

Die Sicherheit dieses Verfahrens basiert nun einerseits auf der Schwierigkeit, aus dem öffentlichen Schlüssel die Generatormatrix G des *Goppa*-Codes abzuleiten, da die Anzahl der Möglichkeiten bei entsprechend geeigneten Werten für n und k nicht zuletzt durch die Anzahl möglicher P und S schnell zu groß wird, um die korrekte Matrix unter realistischem Ressourceneinsatz enumerativ abzuleiten [vgl. 2, S. 115]. Ein weiterer Angriff, den McEliece selbst beschreibt, läge darin, ohne Kenntnis von G zu versuchen, u aus x abzuleiten. Dieser Ansatz lässt sich auf folgendes Problem zurückführen:

Bemerkung 13
Sei x ein empfangenes Wort mit $\epsilon \leq t$ fehlerbehafteten Stellen und H die $n \times (n-k)$-Paritätsprüfmatrix des zugrundeliegenden Codes. Dann ist $s = yH$ das Syndrom von y zu C mit H. Da y bekanntlich fehlerhaft sein kann, wird das Codewort x mit der geringsten *Hamming*-Distanz zu y gesucht, das das Syndrom s bildet, um anschließend das Differenztupel z_0 abziehen zu können [vgl. 46, S. 384]. Folglich soll gelten:

$$s = xH = (y - z_0)H$$

Das Problem, genau dieses z_0 zu finden, wird als **generelles Dekodierproblem** bezeichnet. Berlekamp, McEliece und Tilborg weisen in [46] nach, dass dieses Problem auf $\mathcal{NP}$-harte Probleme zurückführbar ist und damit ebenfalls $\mathcal{NP}$-hart sein muss. Die Klasse der $\mathcal{NP}$-harten Probleme beschreibt Probleme, die über Backtracking-Algorithmen gelöst werden können, deren Suchtiefe durch ein durch die Eingabelänge parametrisiertes Polynom begrenzt ist [vgl. 46, S. 384].

Daraus folgt, dass auch dieser zweite Ansatz bei geeigneten Parametern unausführbar ist [vgl. 2, S. 115]. In Kapitel 5.3 dieser Arbeit wird die daraus begründete aktuelle Relevanz des dargestellten Kryptosystems – insbesondere hinsichtlich der *Post-Quanten-Kryptographie* – diskutiert.

4.2.1 Implementierung

Anhang A.4 zeigt eine beispielhafte Implementierung des Schlüsselerzeugungsalgorithmus und der Verschlüsselung des McEliece-Kryptosystems nach [2]. Auf eine Implementierung des Dechiffrieralgorithmus wurde verzichtet.

Jeweils für das – in der Originalversion [2] irreduzible – *Goppa*-Polynom g, die Störmatrix S und die Permutationsmatrix P kann sowohl eine eigene Wahl angegeben werden als auch eine zufällige Wahl des Programms genutzt werden. Für das Generatorpolynom wird hierfür die Methode `irreducible_element()` einer Instanz der Klasse `PolynomialRing` verwendet. Ein irreduzibles Polynom hat in den hier betrachteten Körpern keine Wurzeln [vgl. 47, S. 3], weshalb die nachfolgend implementierte Mengenbildung der Nicht-Wurzelelemente L obsolet ist, jedoch vorentlastend für den Fall anderer reduzibler Polynome dennoch ausgeführt wird.

Für die Störmatrix S wird, sofern keine Eingabe erfolgt, mittels `random_matrix()`eine zufällige Matrix aus Elementen von $\mathbb{F}_q$ bestimmt. Die Parameterwahl `density=1.0` bewirkt, dass keine der Matrixkomponenten gleich null ist. Diese Wahl ist hier nicht erforderlich, es würde ein Wert zwischen $0,5$ und $1,0$ genügen, um die erforderte Dichte der Matrix zu erreichen. Inwiefern eine Randomisierung dieses Wertes der Sicherheit des Verfahrens zuträglich ist, kann an dieser Stelle nicht beantwortet werden, wobei davon auszugehen ist, dass der Grad der Dichte dieser Matrix mit dem Grad der Verwürfelung der Generatormatrix korreliert.

Auch für die zufällige Wahl der Permutationsmatrix P kann eine *SageMath*-Klasse gewählt werden: `Permutations(n)` bildet die Menge aller möglichen Permutationen der Werte $\{1, \ldots, n\}$ [vgl. 27, „Permutations“], aus der anschließend ein zufälliges Element ausgewählt werden kann. Dieses Vorgehen wird für die Wahl des Störvektors z bei der Verschlüsselung wiederholt: Zunächst wird ein Vektor bestehend aus t Einsen und $n - t$ Nullen gebildet, der anschließend randomisiert permutiert wird, um die Störung nichtdeterministisch werden zu lassen. Auch die Frage, inwiefern die Zufälligkeit der *SageMath*-Methoden aktuellen Anforderungen an Zufallszahlengeneratoren, beispielsweise in Hinblick auf Entropie, entspricht, wird als offene Frage belassen. Für eine Implementierung fernab von Bildungszwecken ist idealerweise eine Sprache mit formaler Verifizierbarkeit der Implementierungen zu wählen.

Die Speicherung des öffentlichen und privaten Schlüssels im JavaScript Object Notation (JSON)-Dateiformat entspricht nicht dem üblicherweise gewählten Privacy Enhanced Mail (PEM)-Dateiformat, erscheint im Kontext dieser Arbeit jedoch probat, um die Verständlichkeit der Dateien zu gewährleisten. Da sich *SageMath*-Matrizen nicht nativ als JSON-Objekte serialisieren lassen, musste hier eine Stringrepräsentation der Liste der Matrixzeilen gewählt werden. Diese Lösung hat jedoch zur Folge, dass, um die Matrizen wieder zu deserialisieren, die Python-Funktion `eval()` aufgerufen werden muss. Die Verwendung dieser Funktion ist aus Codesicherheits-Erwägungen bedenklich, da sie die Ausführung beliebiger eingegebener Befehle ermöglicht. An den gefährdetsten Stellen der Implementierung wird daher vor dem Funktionsaufruf eine Validierung der Eingabe durchgeführt, die sicherstellt, dass die Eingabe nur zugelassene Zeichen enthält.

4.3 Niederreiter-Schema

Während McEliece sein Kryptosystem verbindlich auf *Goppa*-Codes definiert, ist das Kryptosystem von Niederreiter abstrakter, indem für dieses System geeignete Codes diskutiert werden [vgl. 3, S. 161f.].

Die zentrale Aussage, die für dieses Kryptosystem benötigt wird, ist folgende:

Theorem 8
Sei C ein linearer $[n, k, d]$-Code mit Fehlerkorrekturkapazität t über einem endlichen Körper $\mathbb{F}_q$. Dann bezeichne H eine $(n-k) \times n$-Paritätsprüfmatrix dieses Codes. So ist die Menge C beschrieben durch alle Vektoren $c \in \mathbb{F}_q^n$ mit $H \cdot c^T = 0$.

Ferner bezeichne das **Gewicht** $w(x)$ die Anzahl der Komponenten eines Vektors, die ungleich null sind. Für die *Hamming*-Distanz gilt dann $d(x, y) = w(x-y)$. Für zwei Vektoren $y, z \in \mathbb{F}_q^n$ mit $w(y) \leq t$ und $w(z) \leq t$ gilt nun [vgl. 3, S. 160f.]:

$$(H \cdot y^T \; = \; H \cdot z^T) \; \Rightarrow \; y = z$$

Beweis. Sei $c \in C$. Dann ist $H \cdot c^T = 0$. Da $H \cdot y^T - H \cdot z^T = 0$, gilt

$$H \cdot y^T - H \cdot z^T = H \cdot c^T \; \Leftrightarrow \; y - z = c$$

Die Dekodierbarkeit eines linearen Codes wird dadurch sichergestellt, dass zu jedem Vektor $x \in \mathbb{F}_q^n$ maximal ein gültiges Codewort c existiert, sodass $d(x, c) \leq t$. Würde diese Aussage nicht gelten, könnte ein fehlerhaftes Codewort nicht mehr eindeutig dekodiert werden. Da $d(y, \langle 0, 0, \ldots, 0 \rangle) = w(y) \leq t$ und $d(y, c) = w(y - c) = w(z) \leq t$, folgt, dass $c = \langle 0, 0, \ldots, 0 \rangle$ und $y = z$ gelten müssen, da andernfalls die Dekodierbarkeit des Codes nicht gegeben wäre [vgl. 3, S. 161]. □

Dieses Theorem ist die Grundlage dafür, ein kryptographisches System auf linearen Codes aufzubauen: Zu einem Klartext $m \in \mathbb{F}_q^n$ und zu einer wie in obigem Theorem erzeugten Paritätsprüfmatrix H kann der Geheimtext $y = H \cdot m^T$ gebildet und versendet werden. Aufgrund des Rucksackproblems können – bei ausreichender Parameterwahl und unter Annahme realistischer Ressourcen – lediglich jene Empfänger die Nachricht dechiffrieren, die sich in Kenntnis der gewählten Paritätsprüfmatrix H befinden. Dieses skizzierte Kryptosystem ist kein *PKC*, da die Matrix H der gemeinsame Schlüssel zum Ver- und Entschlüsseln ist.

Um nun ein *PKC* auf diesem Verfahren aufzubauen, werden ähnlich zum in [2] dargestellten Vorgehen zusätzliche Störparameter in die Matrix H eingebracht, die das Publizieren eines öffentlichen Schlüssels ermöglichen. Die Schlüsselerzeugung verläuft wie folgt [vgl. 3, S. 161]:

1. Wahl einer nicht-singulären $(n - k) \times (n - k)$-Matrix S
2. Wahl einer $n \times n$-Matrix P, die aus der Permutation einer nicht-singulären Diagonalmatrix entsteht
3. Der öffentliche Schlüssel ist nun $k_{pub} = S \cdot H \cdot P$

Wichtig zu bemerken ist, dass die hier gewählte Matrix P im Gegensatz zur Matrix P des McEliece-Verfahrens auch Komponenten $x_{ij} \notin \{0, 1\}$ enthalten kann, über die sich so eine Gewichtung einzelner Spalten ergeben kann.

Die Ver- und Entschlüsselung folgt nun wie in den Algorithmen 4 und 5 dargestellt.

Algorithmus 4 Verschlüsselungsalgorithmus des Niederreiter-Schemas (nach [3])

Require: k_{pub}, Klartextvektor $y \in \mathbb{F}_q^n$ mit $w(y) \leq t$
1: $x \leftarrow k_{pub} \cdot y^T$
2: **return** x

Algorithmus 5 Entschlüsselungsalgorithmus des NIEDERREITER-Schemas (nach [3])

Require: S, H, P, Geheimtextvektor $x = k_{pub} \cdot y^T$
1: $s \leftarrow S^{-1} \cdot x$
2: $t \leftarrow$ Ergebnis eines zum Code C passenden Dekodierverfahrens auf s
3: $y \leftarrow t \cdot (P^T)^{-1}$
4: **return** y

4.3.1 Korrektheit des Verfahrens

Dass das NIEDERREITER-Verfahren in Ver- und Entschlüsselung korrekt ist, kann anhand der folgenden Aussagen bewiesen werden.

Theorem 9
Ist y ein gültiges Codewort eines GRS_k-Codes, so ist es das Produkt dieses Codewortes und einer Permutationsmatrix P ebenfalls.

Beweis. Nach HILL erzeugen die folgenden Operationen auf einer Generatormatrix G eines linearen $[n, k, d]$-Codes äquivalente Codes [vgl. 48, S. 50]:

- Zeilenvertauschungen
- Multiplikation einer Zeile mit einem Skalar ungleich 0
- Addition eines skalaren Vielfachen einer Zeile auf eine andere
- Vertauschung von Spalten
- Multiplikation einer Spalte mit einem Skalar ungleich 0

Da die Matrix P eine (gewichtete) Permutationsmatrix ist, besitzt sie pro Zeile und Spalte genau in einer Komponente einen Wert ungleich 0. Da nach Definition 25 die Zeilen einer Generatormatrix eine Basis des durch den Code beschriebenen Vektorraums bilden und die Zeilen einer Paritätsprüfmatrix eine Basis des Kerns der Generatormatrix [vgl. 29, S. 29f.], besteht hinsichtlich der Zeilenvertauschungen und der Multiplikation einer Zeile mit einem Skalar bereits Äquivalenz, da dies aus der Definition einer Basis unmittelbar folgt und die lineare Unabhängigkeit erhalten bleibt. Bei der Multiplikation der Matrix H mit der Matrix P werden aufgrund der obig beschriebenen Beschaffenheit der Matrix P die Spalten der Matrix H mit Skalaren multipliziert und vertauscht. Hierbei bleibt die lineare Unabhängigkeit nicht zwangsläufig erhalten, jedoch erzeugen diese Operationen **äquivalente Codes**,

die sich zwar in der Anordnung der Symbole unterscheiden, aber den gleichen Vektorraum aufspannen [vgl. 30, S. 24] [vgl. 49, S. 1193f.].

$$(G \cdot y^T) \in GRS_k \Rightarrow (G \cdot P \cdot y^T) \in GRS_k \tag{4.3}$$

Daraus, dass permutierte Generatormatrizen zu permutierten Codeworten führen, folgt durch Assoziativität, dass auch eine Permutation des Codewortes Teil des betrachteten Codes ist. □

Für die Korrektheit des Verfahrens ist nun noch zu zeigen, dass ein durch die Verschlüsselungsoperation des NIEDERREITER-Verfahrens erzeugtes Chiffrat mittels des korrespondierenden Entschlüsselungsverfahrens dechiffriert werden kann.

Theorem 10
Sei y ein Klartextvektor mit $w(y) \leq t$, H eine Paritätsprüfmatrix eines linearen Codes, S eine nicht-singuläre Matrix und P eine Permutationsmatrix. Dann lässt sich das Produkt $ySHP$ mittels des Entschlüsselungsalgorithmus des NIEDERREITER-Verfahrens dechiffrieren.

Beweis. Das Chiffrat bzw. das Geheimwort ist entsprechend des NIEDERREITER-Verschlüsselungsalgorithmus definiert durch

$$\begin{aligned} x &= ySHP \\ &\Leftrightarrow SH \cdot (y^T P). \end{aligned}$$

Für die Entschlüsselung wird nun zunächst die Störmatrix durch Multiplikation mit ihrem Inversen ‚neutralisiert':

$$\begin{aligned} s &= x \cdot S^{-1} \\ &\Leftrightarrow SHy^T PS^{-1} \\ &\Leftrightarrow Hy^T P \end{aligned}$$

Im Folgenden bezeichne $y_P = yP$. Dann ist Hy_P^T nach Definition 26 genau ein Syndrom-

vektor

$$S = \begin{pmatrix} S_0 \\ S_1 \\ \vdots \\ S_{n-k-1} \end{pmatrix}$$

von y bezüglich des GRS_k-Codes. Die Syndrome bilden nun gemeinsam mit der Paritätsprüfmatrix H die Eingabewerte eines syndrombasierten Dekodieralgorithmus (siehe beispielsweise Kapitel 3.3.2). Da jedoch das Geheimwort nicht etwa aus der Kodierung von y dieses Codes entstanden ist, liefert der syndrombasierte Dekodieralgorithmus als Resultat einen Null-Zeilenvektor $\mathbf{0}$. Der berechnete Fehlerterm entspricht dem Klartextwort y, da

$$\begin{aligned} x &= (G \cdot \mathbf{0} + y) \cdot SHP \\ &\Leftrightarrow y \cdot SHP \end{aligned}$$

gilt, wobei G eine Generatormatrix des Codes GRS_k bezeichnet. Damit ist gezeigt, dass das NIEDERREITER-Verfahren korrekt ist. □

4.3.2 Wahl geeigneter Codes

Die Sicherheit eines solchen Kryptosystems ist stark an die Wahl eines geeigneten Codes gebunden. NIEDERREITER formuliert daher folgende Anforderungen an Codes [vgl. 3, S. 161f.]:

- Die Fehlerkorrekturkapazität des Codes sollte relativ hoch sein, sodass eine möglichst große Anzahl an Klartextvektoren verwendet werden kann.
- Für C sollte es einen effizienten Dekodieralgorithmus geben, damit die Entschlüsselung in kurzen Rechenzeiten ausgeführt werden kann.
- Die Dimension k sollte im Verhältnis zu n weder zu klein noch zu groß gewählt werden, da ein zu kleiner Wert die Anzahl geeigneter Codes einschränkt und ein zu großer Wert die Größe der Codeworte (im Verhältnis zum Nachrichtenwort) minimiert. In beiden Fällen verringert sich die Sicherheit des Kryptosystems.

Als geeignete Codes werden *Goppa*-Codes als Teil der *Alterant Codes*, algebraisch-geometrische Codes und *Reed-Solomon-Codes* vorgestellt, da sie die *Gilbert-Varshamov-Schranke* einhalten [vgl. 3, S. 162]:

Aus Definition 6 geht eine Interpretation der Minimaldistanz als Kugel hervor. Das Volumen dieser Kugel ist nun wie folgt definiert:

Definition 30
Sei $\mathbb{F}_q$ ein endlicher Körper und $\mathbb{F}_q^n$ ein n-dimensionaler Vektorraum über diesem Körper. Dann ist eine Kugel mit Radius t definiert als die Menge aller Wörter mit *Hamming-Distanz* kleiner gleich t. Das **Volumen** dieser Kugel ist definiert als die Anzahl der Elemente der Kugel und wird berechnet durch [vgl. 29, S. 95]

$$V_q(n,t) = \sum_{i=0}^{t} \binom{n}{i} (q-1)^i. \tag{4.4}$$

Ferner sei die **Maximalkardinalität** definiert als die größtmögliche Anzahl an Codeworten eines $[n,k,d]$-Codes über $\mathbb{F}_q$ [vgl. 48, S. 11]:

$$\mathcal{A}_q(n,d) = \max\{|C| \mid C \subseteq \mathbb{F}_q^n \ \wedge \ d(C) = d\} \tag{4.5}$$

Damit lässt sich die *Gilbert-Varshamov-Schranke* wie folgt formulieren:

Theorem 11
Seien $\mathbb{F}_q$ ein endlicher Körper und $n,k,d \in \mathbb{N}$ so gewählt, dass gilt

$$V_q(n-1,d-2) < q^{n-k}. \tag{4.6}$$

Dann existiert ein linearer $[n,k]$-Code mit einer Minimaldistanz von mindestens d [vgl. 29, S. 97].

Eine alternative Formulierung liefert VAN LINT [vgl. 8, S. 66]:

$$\mathcal{A}_q(n,d) \geq \frac{q^n}{V_q(n,d-1)} \tag{4.7}$$

Die *Gilbert-Varshamov-Schranke* liefert folglich eine Untergrenze für die maximal mögliche Codewortanzahl eines Codes, der die Bedingung erfüllt. In Bezug zur Kryptographie sind solche Codes von besonderer Bedeutung, da es nicht möglich sein soll, ein Kryptosystem

durch bloßes Erraten möglicher Code- oder Klartextworte zu brechen. Dass sowohl *Goppa*-Codes als auch verallgemeinerte *Reed-Solomon-Codes* diese Schranke ab $q \geq 49$ erfüllen, stellen VAN LINT und SPRINGER dar [vgl. 50].

4.4 Vergleich zum McEliece-Kryptosystem

Die Frage, inwieweit sich beide dargestellten Vorschläge von MCELIECE und NIEDERREITER und insbesondere auch ihre Sicherheit unterscheiden, ist Gegenstand dieses Abschnittes.

Theorem 12
Sei C ein linearer $[n, k, 2t + 1]$-Code (wobei t die Fehlerkorrekturschranke bezeichne). Dann sind die Kryptosysteme von MCELIECE und NIEDERREITER bezüglich dieses Codes äquivalent zueinander und weisen dieselbe Sicherheit auf.

Beweis. Die Aussage wird durch einen direkten Beweis gezeigt. Sei G' der öffentliche Schlüssel des MCELIECE-Kryptosystems, bestehend aus dem Matrixprodukt SGP, wobei G eine Generatormatrix des betrachteten Codes ist. Gemäß Abschnitt 3.3.2 kann aus G' eine Paritätsprüfmatrix H' abgeleitet werden (hierbei ist unerheblich, dass es sich bei G' nicht um eine Generatormatrix des betrachteten Codes handelt). Die Verschlüsselungsoperation des MCELIECE-Kryptosystems ist definiert durch

$$x = u \cdot G' + z. \tag{4.8}$$

u bezeichnet einen Klartextblock und z einen Fehlervektor mit Länge n und Gewicht $w(z) \leq t$. Wird diese Gleichung nun mit H'^T multipliziert, ergibt sich durch $G'H'^T = 0$:

$$v = x \cdot H'^T = u \cdot G'H'^T + zH'^T \;\Leftrightarrow\; zH'^T \tag{4.9}$$

Die Verschlüsselung im NIEDERREITER-Verfahren ist gegeben durch

$$x = y \cdot H'^T, \tag{4.10}$$

wobei $H' = k_{pub} = SHP$. Da z und y strukturidentische Fehlervektoren der Länge n mit Gewicht $w \leq t$ sind, ist durch 4.9 auch die Verschlüsselung des NIEDERREITER-

Kryptosystems beschrieben [vgl. 51, S. 272]. Sollte es einer angreifenden Person bei bekanntem x und H' aus Gleichung 4.10 möglich sein, y zu finden, gilt das NIEDERREITER-Verfahren als gebrochen. Gleiches trifft jedoch nach 4.9 auch auf das McELIECE-Verfahren zu, da auch hier das Finden von z bei bekanntem u und H' zum Bruch des Systems führt.

Auch ausgehend von der NIEDERREITER-Verschlüsselungsgleichung 4.10 kann die Äquivalenz zur McELIECE-Verschlüsselungsoperation gezeigt werden. Sei c ein n-dimensionaler Vektor mit Gewicht $w \geq t$,[3] der die Gleichung $x = cH'^T$ erfüllt. Dann kann c analog zum obigen Vorgehen aufgefasst werden als

$$c = mG' + y \tag{4.11}$$

für einen k-dimensionalen Vektor m und y mit Gewicht y, denn

$$\begin{aligned} x = c \cdot H'^T &= (mG' + y) \cdot H'^T \\ &\Leftrightarrow mG'H'^T + yH'^T \\ &\Leftrightarrow yH'^T, \end{aligned}$$

was wiederum genau Gleichung 4.10 entspricht. Kann also das McELIECE-Kryptosystem gebrochen werden, so kann es das NIEDERREITER-Kryptosystem ebenfalls [vgl. 51, S. 272]. □

[3] Bei einem Gewicht größer als t ist c nicht mehr durch das NIEDERREITER-Verfahren dechiffrierbar. Hier ist die Dechiffrierung jedoch gar nicht das Ziel.

5 Bedeutung in Gegenwart und Zukunft

Ziel dieses Kapitels ist es, die Bedeutung der vorgestellten Verfahren nach McEliece und Niederreiter und auch der codebasierten Kryptographie im Allgemeinen anhand den Dimensionen Sicherheit, Performanz und Anwendbarkeit zu diskutieren, jeweils auch in Hinblick auf den erwarteten Zeitpunkt, an dem ein funktionsfähiger Quantencomputer in der Lage ist, herkömmliche kryptographische Verfahren zu brechen. Dass die hier aufgeführten Argumente lediglich für die Gegenwart Aussagekraft besitzen, liegt in der Natur der Sache, schließlich ist weder bekannt, *wann* jener Quantencomputer existieren wird, noch *mit welchen Möglichkeiten*.

5.1 Sicherheit

Da bereits gezeigt wurde, dass die Sicherheit der beiden Verfahren nach Niederreiter und McEliece die gleiche Sicherheitsklasse aufweisen, sofern dieselbe Klasse an Codes verwendet wird, liegt es nahe, die Sicherheit einzelner Codeklassen hinsichtlich ihrer Eignung für die Kryptographie zu beurteilen. Darüber hinaus werden exemplarisch weitere Sicherheitseigenschaften beider Verfahren dargestellt.

5.1.1 Strukturangriffe auf GRS-Codes

Sidelnikov und Shestakov zeigen in ihrem 1992 erschienenen Artikel „On insecurity of cryptosystems based on generalized Reed-Solomon codes," dass die insbesondere für das Niederreiter-Schema vorgesehenen GRS-Codes eine strukturelle Schwäche aufweisen: Sei $k_{pub} = H \cdot U$, wobei $U = S^T \cdot P$ und H, S, P entsprechend der Definition des Niederreiter-Schemas eine Paritätsprüf-, Stör- und Permutationsmatrix darstellen. Die Autoren stellen nun in [35] einen Algorithmus vor, mit dem in einer polynomiellen Komplexität von $O((n-k-1)^4+(n-k-1)\cdot n)$ die geheimen Matrizen H und U ermitteln werden können. Dies wird dadurch möglich, dass jeder Eintrag in einer Zeile k_{pub_i} als Polynom mit einem

Grad $\leq (n-k-1)$ aufgefasst werden kann. Es ergibt sich ein System polynomieller Gleichungen, dessen Lösung dem privaten Schlüssel $k_{priv} = \langle S, P \rangle$ entspricht [vgl. 38, S. 120f.]. Dieser Angriff begründet den Ausschluss von verallgemeinerten *Reed-Solomon*-Codes aus der Liste geeigneter Codeklassen. Eine algorithmische Darstellung dieses Algorithmus ist in [52, S. 18] zu finden.

5.1.2 *Adaptive chosen ciphertext*-Angriffe

Definition 31 ([53])
Sei C ein **Public-Key-Kryptosystem** bestehend aus

- dem Schlüsselerzeugungsalgorithmus K,
- dem Verschlüsselungsalgorithmus E und
- dem Entschlüsselungsalgorithmus D.

Ferner sei A ein probabilistischer Angriff auf C, wobei A aus zwei aufeinander folgenden Abläufen A_1 und A_2 mit folgenden Eigenschaften besteht:

1. A_1 erhält als Eingabe das öffentliche Ergebnis von E, also k_{pub}, und berechnet zwei Klartexte m_0, m_1 derselben Länge.
2. Nun wird für den Angriff verborgen m_b mittels k_{pub} verschlüsselt, wobei $b \in \{0, 1\}$ zufällig gewählt wird. Das entstandene Chiffrat wird als c^* bezeichnet.
3. A_2 erhält dieses Chiffrat und versucht, b zu ermitteln.

Ein solcher Angriff A wird als ***Chosen plaintext*-Angriff (IND-CPA)** bezeichnet. Ein Kryptosystem ist dann **ununterscheidbar (IND)**, wenn der richtige Wert für b mit einer Wahrscheinlichkeit von maximal $\frac{1}{2}$ erraten werden kann [vgl. 54, S. 7].

Steht A_1 nur während des ersten Schritts ein Entschlüsselungsorakel zur Verfügung, das eine beliebige Anzahl an Chiffraten entschlüsselt, ohne dabei Informationen über den privaten Schlüssel preiszugeben, handelt es sich um einen **Nicht-adaptiven *Chosen ciphertext*-Angriff (IND-CCA1)**. Steht zusätzlich auch in Schritt 3 ein solches Orakel zur Verfügung, das beliebige Chiffrate mit Ausnahme von c^* entschlüsseln kann, so wird der Angriff A als **Adaptiver *Chosen ciphertext*-Angriff (IND-CCA2)** bezeichnet, da Angreifende durch die Orakel ihre Wahl der Klartexte m_0 und m_1 adaptieren können [vgl. 53, S. 153f.].

Die obig definierte IND-CCA2-Eigenschaft ist eine der stärksten bekannten Vorstellungen von Sicherheit von PKCs [vgl. 55, S. 6672] und fungiert damit natürlicherweise als Basis für die Parameterwahl. Unter anderem die Arbeiten von DOTTLING, DOWSLEY, MULLER-QUADE ET AL. [55] und PERSICHETTI [56] zeigen, durch welche Parameter die Ununterscheidbarkeit von Codeworten selbst bei adaptiven *chosen ciphertext*-Angriffen im McEliece-Kryptosystem erreicht werden kann. Die ursprünglich publizierte Form des McEliece-Kryptosystems ist nicht IND-CCA2-sicher [vgl. 57, S. 34].

Insbesondere durch das Aufgeben von Restriktionen in der Wahl des zugrundeliegenden endlichen Körpers $\mathbb{F}_q$ und auch des Schlüsselerzeugungsverfahrens lässt sich die Sicherheit des McEliece-Kryptosystems signifikant zu der ursprünglich publizierten Variante erhöhen. So werden durch die Erzeugung des öffentlichen Schlüssels $k_{pub} = S \cdot G \cdot P$ beispielsweise Seitenkanalangriffe ermöglicht, weshalb es sogar sicherer sein kann, statt dieser Multiplikationen eine systematische Form G' der Generatormatrix G als Schlüssel zu veröffentlichen [vgl. 56, S. 169], sofern durch Verwürfelung der Eingaben die Ununterscheidbarkeit gewährleistet wird [vgl. 57, S. 34].

5.1.3 *Information Set Decoding*-Angriffe

Eine weitere Klasse von Angriffen auf die Kryptosysteme von NIEDERREITER und McEliece sind *Information Set Decoding*-Angriffe, die auf der bereits 1962 veröffentlichten Arbeit „The use of information sets in decoding cyclic codes“ von PRANGE basieren und damit bereits McEliece bei der Wahl der Parameter des durch ihn vorgeschlagenen Kryptosystems beeinflusst haben. Veränderungen und Optimierungen dieses Angriffs, wie unter anderem beschrieben in [57] führen dazu, dass ausreichend große Parameter für n, t und nach Möglichkeit auch endliche Körper $\mathbb{F}_q$ mit anderen Werten für q als Zweierpotenzen gewählt werden müssen, um diesen Angriffen zu widerstehen [vgl. 57, S. 44ff.] [vgl. 59, S. 11].

Diese Angriffe basieren darauf, über die Berechnung einer systematischen Form einer Generatormatrix eines Codes C und einen gezielt erzeugten Vektor mit einer definierten Maximaldistanz zu C zu erwirken, dass die Fehlerstellen einer Nachricht durch die systematische Form ersichtlich werden [vgl. 60]. Diesen Angriffen kann durch die beschriebenen Maßnahmen begegnet werden [vgl. 57, S. 44ff.].

5.2 Anwendungen in der Gegenwart

Während sich die Sicherheitsbewertung der Verfahren auch Dekaden nach ihrer Veröffentlichung nur geringfügig verändert hat und nie in einem Ausmaß, dem sich nicht durch Anpassungen der Parameter begegnen ließe, sind kaum Implementierungen und Anwendungen der Kryptosysteme von McEliece und Niederreiter bekannt [vgl. 61, S. 44:2].

Auch eine Webrecherche mit den Suchtermen „Applications of McEliece“ und „McEliece in practice“ am 05.04.2023 in den Suchmaschinen *Google*, *Google Scholar* und *Cryptology ePrint Archive (IACR)* lieferte neben akademischen Beiträgen kaum Informationen über praktische Anwendungsszenarien und Implementierungen, die über die wissenschaftliche Betrachtung des Kryptosystems hinausgehen. Die Gründe dafür liegen womöglich in der in Relation zu anderen Verfahren großen Schlüssellänge von mehreren Kilo- bis Megabyte [vgl. 62, S. 79]. Einige Versuche, die Schlüssellängen des Verfahrens zu reduzieren, auch durch die Wahl anderer Codes, verringerten die Sicherheit des Verfahrens und erwiesen sich dadurch als ungeeignet [vgl. 61, S. 44:2]. Andere Beiträge mit diesem Ziel wie [62] reduzierten die Schlüssellänge zwar, jedoch nicht auf ein mit anderen PKCs vergleichbares Maß.

5.3 Quantensicherheit

Dass code-basierte Kryptosysteme geeignet sein könnten, um die Schutzziele der Vertraulichkeit und Integrität auch unter der Verfügbarkeit von Quantencomputern zu gewährleisten, zeigen diverse Arbeiten, Forschungsgruppen und nicht zuletzt auch die Tatsache, dass entsprechende Kryptosysteme Teil von Standardisierungsrunden des National Institute of Standards and Technology (NIST) sind.

In diesem Abschnitt wird begründet, warum das McEliece-Kryptosystem als Kandidat für *Post-Quantum Cryptography* gilt. Anschließend werden drei Kryptosysteme aus dem Standardisierungsprozess des NIST vorgestellt, die auf code-basierter Kryptographie und insbesondere auch auf den Kryptosystemen von Niederreiter und McEliece basieren.

5.3.1 Argumentation

Etablierte Verfahren wie *RSA* oder *ElGamal* basieren auf zahlentheoretischen Problemen wie dem diskreten Logarithmus oder der Primfaktorzerlegung [63]. Während die Sicherheit auf konventionellen Computern durch die Abwesenheit effizienter Algorithmen zur Lösung dieser Probleme gewährleistet wird, zeigt sich durch Algorithmen wie jenem von SHOR in [64], der auf *Quantum Fourier Sampling* basiert [vgl. 63, S. 763] [vgl. 65, S. 330], dass diese Verfahren mit Quantencomputern gebrochen werden können.

DINH, MOORE und RUSSELL zeigen in [63], dass *Quantum Fourier Sampling*-Angriffe jedoch nicht die Sicherheit der Verfahren von MCELIECE und NIEDERREITER kompromittieren. Da diese Angriffe in einem Großteil der Quantenalgorithmen verwendet werden [vgl. 63, S. 763], entsteht die Einordnung dieser Kryptosysteme als potenziell quantensicher. Dass die beiden Kryptosysteme nicht resistent gegenüber anderen, möglicherweise noch nicht bekannten Quantenalgorithmen sind, bleibt jedoch möglich. Daher ist ein Ziel des NIST-Standardisierungsprozesses, die Sicherheit der Kryptosysteme gründlich und langfristig zu erforschen [vgl. 66].

5.3.2 Aktuelle Kandidaten

Im NIST-Standardisierungsprozess für Post-Quanten-Kryptographie werden in mehreren Runden Verfahren analysiert, hinsichtlich Design, Sicherheit und Performanz bewertet und bei entsprechender Eignung standardisiert [vgl. 66, S. 27ff.]. Eingereicht wurden und analysiert werden Verfahren, die auf fehlerkorrigierenden Codes, multivariaten Polynomen oder Gittern basieren. Ein ebenfalls eingereichtes Verfahren, das auf Isogenien aufgebaut ist, konnte total gebrochen werden [vgl. 67] und ist daher nicht mehr Teil des Standardisierungsprozesses. Hingegen sind code-basierte Verfahren zwar noch nicht standardisiert, jedoch weiterhin Teil des Auswahlprozesses, zum Zeitpunkt der Abgabe dieser Arbeit in Runde vier. Im Folgenden werden die drei Verfahren *BIKE*, *Classic McEliece* und *HQC* kurz beschrieben.

BIKE

BIKE ist ein Akronym für *Bit Flipping Key Encapsulation* und bezeichnet einen code-basierten Key Encapsulation Mechanism (KEM) auf Grundlage von quasi-zyklischen Moderate

Density Parity-Check (MDPC)-Codes [vgl. 66, S. 29]. Die Verschlüsselung während des Public Key Exchange (PKE) folgt dabei dem NIEDERREITER-Schema.

Definition 32
Sei C ein linearer $[n, k]$-Code. Falls eine Zahl $n_0 \in \mathbb{N}$ existiert, sodass für alle Codeworte $c \in C$ gilt, dass die Verschiebung von c um n_0 ebenfalls ein gültiges Codewort aus C ist, so ist C **quasi-zyklisch** [vgl. 68, S. 2070].

Definition 33
Ein $[n, k, d]$-MDPC-Code ist ein linearer Code der Länge n, Dimension k, dessen Paritätsprüfmatrizen in ihren Zeilen das konstante Gewicht d aufweisen [vgl. 68, S. 2070].

BIKE sei seinen Autoren zufolge IND-CCA-sicher [vgl. 66, S. 31]. Die NIST beabsichtigt, das Verfahren aufgrund seiner günstigen Performanzeigenschaften zum Ende der vierten Runde ihres Verfahrens zu standardisieren [vgl. 66, S. 31].

Classic McEliece

Wie der Name andeutet, bezeichnet *Classic McEliece* ein code-basiertes Kryptosystem, das in seinem Design nah an den klassischen Ansätzen von NIEDERREITER und MCELIECE bleibt. So wird ein binärer *Goppa*-Code als Grundlage gewählt und die Verschlüsselung erfolgt wie im NIEDERREITER-Schema, wenn auch mit Modifikationen, um die Eigenschaft, IND-CCA-sicher zu sein, zu erreichen [vgl. 66, S. 31f.].

Im Gegensatz zu *BIKE* ist die Schlüssellänge und der daher benötigte Rechenaufwand nach Auffassung der NIST zu hoch für eine Vielzahl von Anwendungen [vgl. 66, S. 33]. Dennoch wird die Einreichung in der vierten Runde weiter betrachtet, insbesondere, falls sich konkrete Anwendungsfälle ergeben, in denen konkurrierende Verfahren weniger gut geeignet scheinen als *Classic McEliece* [vgl. 66, S. 33].

HQC (Hamming Quasi-Cyclic)

Das Kryptosystem *HQC* basiert wie *BIKE* auf quasi-zyklischen MDPC-Codes, deren Generatormatrizen jedoch in systematischer Form verwendet werden. Dafür ist die Definition zyklischer Matrizen erforderlich.

Definition 34
Sei $x = \langle x_0, \ldots, x_{n-1} \rangle \in \mathbb{F}_q^n$. Dann wird die aus x abgeleitete Matrix

$$\mathrm{rot}(x) = \begin{pmatrix} x_0 & x_n & \cdots & x_1 \\ x_1 & x_0 & \cdots & x_2 \\ \vdots & \vdots & & \vdots \\ x_{n-1} & x_{n-2} & \cdots & x_0 \end{pmatrix} \tag{5.1}$$

als **zyklisch** bezeichnet [vgl. 69, S. 3929].

Mithilfe dieser zyklischen Matrizen können *systematische quasi-zyklische* Codes angegeben werden [vgl. 69, S. 3930f.].

Das Verfahren weist nach Auffassung der NIST eine akzeptable, wenn auch nicht optimale Performanz auf, wird aber insbesondere aufgrund der starken Sicherheitseigenschaften weiter im Standardisierungsprozess betrachtet [vgl. 66, S. 34].

6 Zusammenfassung und Diskussion

6.1 Zusammenfassung

In dieser Arbeit wurden kryptographische Verfahren, die auf fehlerkorrigierenden Codes wie *Goppa-* oder verallgemeinerten *Reed-Solomon-*Codes basieren, hinsichtlich ihres mathematischen Fundaments, ihrer Konfiguration und Bedeutung für die *Post-Quanten-Kryptographie* dargestellt und diskutiert.

Fehlerkorrigierende Codes Fehlerkorrigierende Codes sind ein Teilgebiet der Codierungstheorie und der Kanalcodierung zuzuordnen, da ihr Zweck darin besteht, Informationen vor Übertragungsfehlern zu schützen, indem redundante Bits hinzugefügt werden. Je nach verwendetem Code kann so eine Anzahl an Fehlern *erkannt* beziehungsweise *korrigiert* werden. Die typischerweise zu diesem Zweck verwendeten linearen Codes basieren auf der Interpolation von Polynomen und werden zumeist auf endlichen Körpern der Form $\mathbb{F}_q$ definiert, wobei q eine Primzahlpotenz ist. Sehr weit verbreitet, beispielsweise auch in *Quick Response (QR)*-Codes, sind *Reed-Solomon-Codes*. Kryptographisch bedeutsam ist die verallgemeinerte Form dieser Codes, GRS-Codes, bei der die gewählten Auswertungsstellen des zu interpolierenden Polynoms durch ein Gewichtstupel augmentiert werden.

Zu linearen Codes können *duale* Codes gebildet werden. Für die Codierung verwendet werden nun *Generator-* und *Paritätsprüfmatrizen* eines Codes, wobei eine *Generatormatrix* eines Codes die *Paritätsprüfmatrix* des zu ihm dualen Codes ist. Die Kodierung eines Informationstupels kann *systematisch* oder *nicht-systematisch* erfolgen. Dafür wird eine Generatormatrix des Codes durch elementare Matrixoperationen so verändert, dass eine Submatrix der Generatormatrix eine Identitätsmatrix bildet. Die Dekodierung linearer Codes kann nun über *Syndrome*, also Produkte aus empfangenem Wort und Paritätsprüfmatrix, erfolgen. In der Folge ensteht ein Gleichungssystem, über das fehlerhafte Stellen erkannt und korrigiert werden können, sofern die Fehlerkorrekturschranke des Codes eingehalten wurde.

Kryptographische Verfahren Auf Basis dieser Codes wurden durch McEliece und Niederreiter jeweils *Public-Key*-Kryptosysteme vorgeschlagen, wobei für Systeme diesen Typs charakteristisch ist, dass für Ver- und Entschlüsselung jeweils verschiedene Schlüssel verwendet werden und somit der Schlüssel zur Verschlüsselung veröffentlicht werden kann. In beiden Verfahren wird zunächst ein zugrundeliegender fehlerkorrigierender Code festgelegt. Dessen Generatormatrix im McEliece-Verfahren beziehungsweise dessen Paritätsprüfmatrix im Niederreiter-Schema wird nun permutiert und verwürfelt, um es Angreifenden unmöglich zu machen, die eigentliche Generatormatrix zu bestimmen. Die daraus entstandene Matrix stellt den öffentlichen Schlüssel dar. Die zur Permutation und Verwürfelung genutzten Matrizen fungieren nun als für die Entschlüsselung verwendeter privater Schlüssel und müssen geheim gehalten werden. Im McEliece-Verfahren wird die zu verschlüsselnde Nachricht nun mit dem öffentlichen Schlüssel multipliziert und zusätzlich wird ein zufälliger Störvektor hinzuaddiert. Im Niederreiter-Schema wird der öffentliche Schlüssel mit einem Klartextvektor verschlüsselt, der jedoch im Kontext der Dekodierung einem Fehlervektor entspricht. Die Entschlüsselung beider Verfahren verläuft nun durch die inverse Anwendung der Permutations- und Verwürfelungsmatrizen und der Anwendung eines Dekodieralgorithmus, wodurch der Klartext entsteht. Sofern beide Verfahren denselben Code verwenden, sind beide Verfahren äquivalent zueinander und haben identische Sicherheitseigenschaften.

Sicherheit und Bedeutung Die von Niederreiter für die Verwendung seines Kryptosystems vorgeschlagenen GRS-Codes sind für kryptographische Zwecke ungeeignet, da durch Sidelnikov und Shestakov ein Angriff publiziert wurde, der Struktureigenschaften dieser Codes ausnutzt und somit Fehlerstellen ohne Kenntnis des privaten Schlüssels wiederherstellen kann. Für die für das McEliece-Kryptosystem vorgesehenen *Goppa*-Codes sind bei geeigneter Wahl der Parameter keine erfolgreichen Angriffe bekannt, weshalb das Verfahren als sicher gilt. Über entsprechende Parameter kann erreicht werden, dass das Verfahren sicher in Bezug auf *adaptive chosen ciphertext*-Angriffe wird. Da beide Verfahren ferner auf mathematischen Problemen basieren, die als $\mathcal{NP}$-hart gelten, und Quantenalgorithmen wie *Quantum Fourier Sampling* keine signifikanten Einschränkungen der Sicherheit erwarten lassen, gelten code-basierte Verfahren wie das McEliece-Kryptosystem als geeignete Kandidaten für die *Post-Quanten-Kryptographie* und sind entsprechend Teil eines Standardisierungsprozesses des NIST. In diesem Prozess sind auch Verfahren enthalten, die nicht auf *Goppa*-Codes, sondern auf quasi-zyklischen MDPC-Codes basieren und dadurch Vorteile in Bezug auf Rechenleistung und Schlüssellänge aufweisen.

6.2 Reflexion

Diese Arbeit stellt die theoretischen Hintergründe code-basierter Kryptographie dar und betrachtet insbesondere die Kryptosysteme von NIEDERREITER und MCELIECE. Relevante Algorithmen wie Kodierung, Dekodierung, Verschlüsselung und Entschlüsselung wurden zusätzlich zu einer Beschreibung in Text und mathematischen Ausdrücken im Mathematiksoftwaresystem *SageMath* implementiert, um die einzelnen Algorithmen und ihre Korrektheit anschaulich werden zu lassen. Die Implementierungen sind jedoch nicht für andere als akademische Zwecke geeignet, da sie weder resistent gegenüber Seitenkanalangriffen sind, noch formal verifiziert wurden und zudem in der Standardausgabe eine Vielzahl von Informationen über die Algorithmen und ihre Parameter preisgegeben werden, was insbesondere für kryptographische Zwecke problematisch ist.

Der Fokus dieser Arbeit liegt weniger auf der Verifikation der Sicherheit und der Ausarbeitung möglicher Angriffe, sondern auf der algorithmischen Darstellung der Kryptosysteme und ihrer Hintergründe. In folgenden Arbeiten sollte daher ein stärkerer Fokus auf Versuche, die Sicherheit der Algorithmen und Kryptosysteme zu beweisen, gelegt werden und Angriffe wie jener von SIDELNIKOV und SHESTAKOV implementiert werden. Weitere Verfahren der code-basierten Kryptographie, die vielleicht weniger populär, aber dennoch für eine tiefergehende Betrachtung geeignet sind, bedürfen ebenfalls weiterer Untersuchung, da sie hier lediglich in geringem Umfang Erwähnung finden.

6.3 Ausblick

Kryptographie, die auf fehlerkorrigierenden Codes basiert, erfährt durch das Forschungsfeld der *Post-Quanten-Kryptographie* wesentlich mehr Aufmerksamkeit als zum Zeitpunkt der Veröffentlichung der entsprechenden Verfahren. Neben Verfahren, die auf *multivariaten Polynomen* oder *Gittern* basieren, ergibt sich durch code-basierte Verfahren ein in der Praxis neuartiger, wenn auch in der Theorie fundiert erforschter, Typus kryptographischer Verfahren, der für die Informationssicherheit der Zukunft mit hoher Wahrscheinlichkeit bedeutend sein wird. Insbesondere jene Verfahren, die auf quasi-zyklischen Codes basieren, erscheinen erfolgsversprechend, da sie die Probleme der Originalpublikationen wie die großen Schlüssellängen oder Performanznachteile adressieren. Offene Forschungsfragen bestehen unter anderem darin, sichere und praktikable Implementierungen der Systeme zu

finden. Ebenso wird die Verifizierbarkeit der Sicherheit und jene einzelner Implementierungen einen Forschungsgegenstand darstellen. Es ist davon auszugehen, dass die Forschung in diesen Bereichen mit Abschluss des NIST-Standardisierungsprozesses und der voranschreitenden Entwicklung von Quantencomputern beschleunigt wird, da dann die Notwendigkeit besteht, etablierte, jedoch nicht quantensichere, kryptographische Primitiven durch neue Verfahren der *Post-Quanten-Kryptographie* zu ersetzen. Mit Quantencomputern wird mit hoher Wahrscheinlichkeit ebenfalls die Forschung zur Kryptoanalyse der neuen Verfahren verstärkt werden.

A Anhang

A.1 Implementierung der GRS-Kodierung

```
#!/usr/bin/env sage
print("### GENERALIZED REED-SOLOMON ENCODING")
print("### This sage script allows you to perform a (non-)
    systematic Generalized Reed-Solomon encoding over a finite
    field F_q.")
print("### (C) Roman Wetenkamp, 2023.\n")

# Basic Code properties
q = input("Please type in q, e.g. 2^2: \t\t")
n = int(input("Please type in length n: \t\t"))
k = int(input("Please type in dimension k: \t\t"))

# Generate Galois field
F_q = None
if len(q.split("^")) < 2:
  F_q.<a> = GF(int(q))
elif len(q.split("^")) == 2:
  F_q.<a> = GF(pow(int(q.split("^")[0]), int(q.split("^")[1])))
else:
  print(f"ERROR:\tSorry, unable to work with this q={q}!")
  exit(1)

# GRS Code properties
a = input(f"\nPlease type in evaluation tupel \na = (a_0, a_1,
    ..., a_{n-1}): \t\t")
a = [F_q(x) for x in a[1:-1].split(",")]

if len(a) != n:
  print(f"ERROR:\tTupel length {len(a)} differs from code length
     {n}!")
  exit(1)
```

```
28
29 for element in a:
30   if a.count(element) > 1:
31     print("ERROR:\tElements␣of␣a␣must␣be␣pairwise␣distinct!")
32     exit(1)
33   else:
34     pass
35
36 v = input(f"Please␣type␣in␣weight␣tupel␣\nv␣=␣(v_0,␣v_1,␣...,␣v_
    {n-1}):␣\t\t")
37 v = [F_q(x) for x in v[1:-1].split(",")]
38
39 if len(v) != n:
40   print(f"ERROR:\tTupel␣length␣{len(v)}␣differs␣from␣code␣length
       ␣{n}!")
41   exit(1)
42
43 for element in v:
44   if v == 0:
45     print("ERROR:\tElements␣of␣v␣must␣not␣be␣zero!")
46     exit(1)
47   else:
48     pass
49
50 # Generate Generator Matrix G of GRS_k(a, v)
51 rows = []
52 i = 0
53 while i < k:
54   row = []
55   j = 0
56   while j < n:
57     row.append(v[j]*(a[j]**i))
58     j += 1
59   rows.append(row)
60   i += 1
61
62 G = matrix(F_q, rows)
```

```
63
64 print(f"\nGenerator matrix: \n{G}")
65
66 # Generate systematic Generator Matrix G_S of GRS_k(a, v)
67 A = G.submatrix(0,0,k,k)
68 G_S = A.inverse() * G
69 print(f"\nSystematic form matrix: \n{G_S}")
70
71 # Perform Encoding
72 h = input(f"\nPlease type in message tupel \nh = (h_0, ..., h_{k
    -1}): \t\t\t")
73 h = [F_q(x) for x in h[1:-1].split(",")]
74
75 if len(h) != k:
76   print(f"ERROR:\tTupel length {len(h)} differs from input word
       length {k}!")
77   exit(1)
78
79 h_v = matrix(F_q, h)
80
81 C = h_v * G
82 C_S = h_v * G_S
83
84 print(f"\nCodeword non-systematic approach:\t{C.rows()[0]}")
85 print(f"Codeword systematic approach:\t\t{C_S.rows()[0]}")
```

Quelltext A.1: *SageMath*-Implementierung des Kodieralgorithmus für GRS-Codes

A.2 Transformieren der Generatormatrix in die systematische Form

Die folgende Ausarbeitung stellt die Ergebnisse einer Recherche zu Algorithmen dar, die es ermöglichen, aus nicht-systematischen Generatormatrizen systematische zu konstruieren. Sie entstand als Nebenprodukt zu dieser Studienarbeit und ist hier in Gänze dargestellt, da sie weder publiziert noch zitierfähig ist.

On transforming Generator Matrices of GRS_k Codes for systematic Encoding

A Literature Review

Roman Wetenkamp*

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

January 29, 2023

Abstract

Linear codes like GRS_k are frequently used for error-correction and even cryptographic purposes. Since the encoding is often required to be *systematic*, the goal of this paper is to find algorithms that transform *non-systematic* generator matrices into *systematic* ones. The conducted literature review shows that most frequently the inverse of the $k \times k$-submatrix is computed and multiplied to the generator matrix to achieve the *standard form*.

1 Introduction

The encoding of information tupels with linear codes is done by matrix multiplication. Since several generator matrices for the same linear code exist, approaches like *systematic encoding* make use of this in order to produce codewords with certain properties. This article deals with the question how existing generator matrices of a linear code can be transformed into *systematic* ones algorithmically.

2 Problem

Definition 1 ([1]). *A code C with length n over a finite field $\mathbb{F}_q$ with $q \in \mathbb{P}$ or $q = p^m, p \in \mathbb{P} \wedge m \in \mathbb{N}$ is said to be **linear** if C forms a linear subspace of $\mathbb{F}_q^n$.*

* ✉ s200376@student.dhbw-mannheim.de

Linear Codes can be represented by a generator matrix and a parity-check matrix which is the generator matrix of the dual code.

An example for linear codes is the class of *Generalized Reed Solomon* codes:

Definition 2 ([2]). *Let $a = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ be a tupel of pairwise distinct components from $\mathbb{F}_q$ and $v = \langle v_0, v_1, \ldots, v_{n-1} \rangle$ be a tupel of components from $\mathbb{F}_q \backslash \{0\}$. The **generalized Reed-Solomon code** $GRS_k(a, v)$ is now defined by all codewords*

$$c = \langle v_0 f(a_0), v_1 f(a_1), \ldots, v_{n-1} f(a_{n-1}) \rangle$$

for all polynomials $f \in \mathbb{F}_q[x]/(x^n - 1)$ with $\deg f < k$.

The generator matrix for this class of codes as in [2] is given by

$$G_{GRS} = \begin{pmatrix} v_0 & v_1 & \cdots & v_{n-1} \\ v_0 a_0 & v_1 a_1 & \cdots & v_{n-1} a_{n-1} \\ \vdots & \vdots & & \vdots \\ v_0 a_0^{k-1} & v_1 a_1^{k-1} & \cdots & v_{n-1} a_{n-1}^{k-1} \end{pmatrix}$$

Definition 3 ([1]). *A generator matrix G is called to be **systematic** if it has the form $G = [I_k \mid P_{n-k}]$ with I_k being the $k \times k$-identity matrix and P_{n-k} being the $k \times (n-k)$-matrix for generating parity-check bits.*

Since the matrix G_{GRS} is in *non-systematic* form, an algorithm for transforming this matrix into a *systematic* form is required.

3 Method

A literature review is conducted.

- *Question of Research:* Which algorithms generate *systematic* generator matrices for linear codes like Generalized Reed Solomon?
- *Derived search terms:* "Generalized Reed Solomon" $\wedge$ ("Systematic Encoding" $\vee$ "Generator Matrix"); "Generator Matrix" $\wedge$ ("Standard Form" $\vee$ "Systematic Form");
- *Databases:* IEEE Xplore (1); Google Scholar (2); JSTOR (3); KIT KVK (4);
- *Scope:* 1+3: Metadata; 2+4: Full-texts
- *Criteria for Selection:* Implementability; Efficiency; Applicability;

Since a variety of different shaped and constructed generator matrices is found among linear codes in general, it seems ineffective to search for terms like "Linear codes *systematic* encoding" containing no information on specific code classes. To find general algorithms that work for several types of linear codes, the second search term was introduced.

4 Results

Using the first search term "Generalized Reed Solomon *systematic* encoding", the following results were gathered:

- Brauchle points out that matrices of Generalized Reed Solomon codes are *Vandermonde* matrices. He proposes a new algorithm for the recovery of erased symbols [3]. The search term "Vandermonde matrix *systematic* conversion" will be added to this papers scope. Since [3] references [4], it will be added to the review.
- In [4] an algorithm for the *systematic* encoding of Reed-Solomon codes with arbitrary parity positions (neither pre- nor postponed to the information bits) is proposed. Brauchle and Koetter show that for a given parity-check *Vandermonde* matrix $H =$

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha^0 & \alpha^1 & \cdots & \alpha^{n-1} \\ \vdots & \vdots & & \vdots \\ \alpha^{0\cdot(2t-1)} & \alpha^{1\cdot(2t-1)} & \cdots & \alpha^{(n-1)\cdot(2t-1)} \end{pmatrix}$$

a *systematic* form can be derived by multiplying the original matrix with the inverse of the $2t \times 2t$-submatrix B_{2t}:

$$H = [A \mid B_{2t}] \Rightarrow H_{sys} = B_{2t}^{-1} H = [P \mid I_{2t}]$$

- Mattoussi, Roca, and Sayadi compare in [5] the construction of *systematic* generator matrices using *Vandermonde* and *Hankel* matrices. Their *Hankel*-based approach has no need for matrix inversions and is therefore considered to be faster and simpler [5].

The second search term "Generalized Reed Solomon Generator Matrix" delivers one additional result:

- Roth and Seroussi proved in [6] that a $GRS_k(\alpha, v)$ code with length n has a *systematic* generator matrix $G = [I \mid A]$ where A is a $k \times (n-k)$ – Generalized Cauchy Matrix. Their proof is constructive, hence a *systematic* generator matrix can be derived from a given *non-systematic* generator matrix using the formulas in [6].

The term "Generator Matrix Standard Form" gives related procedures:

- Nakkiran, Rashmi, and Ramchandran describe in [7] the "systematic remapping" operation in order to generate *systematic* codewords from *non-systematic* generator matrices: Let G be a $k \times n$-generator matrix. G_k denotes the $k \times k$-submatrix consisting of the first k columns of G. Since encoding is done by $c = G \cdot m$ for an information tuple m of length k, the *remapping step* and *systematic* encoding is defined by

$$\overline{m} = G_k^{-1} \cdot m \Rightarrow c_{sys} = G \cdot \overline{m}$$

Several papers described here reference [8]. Hill proposes an algorithm for transforming a generator matrix G to *standard form* using elementar operations like row/column permutation, row multiplications with scalars or adding multiples of a row to another.

Algorithm 1 Algorithm for generating a standard form matrix after [8]

Require: $G = (g_{ij})_{1 \leq i \leq k,\, 1 \leq j \leq n}$
  for $j \in \{1, \ldots, k\}$ **do**
    if $g_{jj} = 0$ **then**
      if $i \in \{j+1, \ldots, k\} \colon g_{ji} \neq 0$ **then**
        $\mathrm{row}(g_{jj}) \leftarrow \mathrm{row}(g_{ij}))$
      else
        $\mathrm{col}(g_{jj}) \leftarrow \mathrm{col}(g_{jh} \neq 0)$
      end if
    end if
    $\mathrm{row}(g_{jj}) \leftarrow \mathrm{row}(g_{jj}) \cdot g_{jj}^{-1}$
    for $i \in \{1, 2, \ldots, k\} \colon i \neq j$ **do**
      $\mathrm{row}(g_{i0}) \leftarrow \mathrm{row}(g_{i1}) - g_{ij} \cdot \mathrm{row}(g_{1j})$
    end for
  end for

5 Conclusion

The research has shown that the default approach for *systematic* encoding of linear codes is given by *Vandermonde* matrices. For a given *non-systematic* generator matrix, the $k \times k$-submatrix is inverted and multiplied with G to achieve the *systematic* form $G_{sys} = [I_k \mid P]$. Since matrix inversion can be an exhaustive task for large matrices, alternative approaches using *Hankel* or *Cauchy* matrices could be taken into consideration. Even row and column operations could be used algorithmically to achieve a *systematic* generator matrix, but it should be pointed out that this algorithm does not seem to be more efficient than matrix inversion since matrix inversion is performance engineered in many programming libraries.

Specific solutions for GRS_k codes were not discovered during the conducted literature review. Since this literature review was not done systematically and a comparison of results did not take place, further investigation is required to find and verify the most efficient algorithm for the underlying question.

References

[1] W. C. Huffman and V. Pless, "Basic concepts of linear codes," in *Fundamentals of Error-Correcting Codes*, Cambridge, UK: Cambridge University Press, 2010, ch. 1, pp. 1–52, ISBN: 9780521131704. [Online]. Available: `https://archive.org/details/fundamentalsofer0000huff/` (visited on 01/25/2023).

[2] F. MacWilliams and N. Sloane, "10 reed-solomon and justesen codes," in *The Theory of Error-Correcting Codes*, ser. North-Holland Mathematical Library, vol. 16, Elsevier, 1977, pp. 294–316. DOI: `https://doi.org/10.1016/S0924-6509(08)70535-X`.

[3] J. Brauchle, "On efficient recovery of erased symbols in generalized reed-solomon codes," in *2011 IEEE International Conference on Communications (ICC)*, 2011, pp. 1–5. DOI: `10.1109/icc.2011.5962475`.

[4] J. Brauchle and R. Koetter, "A systematic reed-solomon encoder with arbitrary parity positions," in *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference*, 2009, pp. 1–4. DOI: `10.1109/GLOCOM.2009.5426304`.

[5] F. Mattoussi, V. Roca, and B. Sayadi, "Complexity comparison of the use of vandermonde versus hankel matrices to build systematic mds reed-solomon codes," in *2012 IEEE 13th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2012, pp. 344–348. DOI: `10.1109/SPAWC.2012.6292924`.

[6] R. M. Roth and G. Seroussi, "On generator matrices of mds codes (corresp.)," *IEEE Transactions on Information Theory*, vol. 31, no. 6, pp. 826–830, 1985. DOI: `10.1109/TIT.1985.1057113`.

[7] P. Nakkiran, K. V. Rashmi, and K. Ramchandran, "Optimal systematic distributed storage codes with fast encoding," in *2016 IEEE International Symposium on Information Theory (ISIT)*, 2016, pp. 430–434. DOI: `10.1109/ISIT.2016.7541335`.

[8] R. Hill, "Introduction to linear codes," in *A First Course in Coding Theory*, ser. Oxford Applied Mathematics and Computing Science Series, Oxford University Press, 1986, ISBN: 0-19-853804-9.

A.3 Implementierung eines GRS-Dekodieralgorithmus

```
#!/usr/bin/env sage
print("### GENERALIZED REED-SOLOMON DECODING")
print("### This sage script allows you to perform a GRS decoding over a finite field F_q.")
print("### (C) Roman Wetenkamp, 2023.\n")

# Basic Code properties
q = input("Please type in q, e.g. 2^2: \t")
n = int(input("Please type in length n: \t"))
k = int(input("Please type in dimension k: \t"))

# Generate Galois field
F_q = None
if len(q.split("^")) < 2:
  F_q.<a> = GF(int(q))
elif len(q.split("^")) == 2:
  F_q.<a> = GF(pow(int(q.split("^")[0]), int(q.split("^")[1])))
else:
  print(f"Sorry, unable to work with this q={q}!")

# Generate GRS-Code
ae = input(f"\nPlease type in evaluation tupel \na = (a_0, a_1, ..., a_{n-1}): \t")
ae = [F_q(x) for x in ae[1:-1].split(",")]

if len(ae) != n:
  print(f"ERROR:\tTupel length {len(ae)} differs from code length {n}!")
  exit(1)

for element in ae:
  if ae.count(element) > 1:
    print("ERROR:\tElements of a must be pairwise distinct!")
    exit(1)
```

```
32    else:
33      pass
34
35  v = input(f"Please␣type␣in␣weight␣tupel␣\nv␣=␣(v_0,␣v_1,␣...,␣v_
      {n-1}):␣\t")
36  v = [F_q(x) for x in v[1:-1].split(",")]
37
38  if len(v) != n:
39    print(f"ERROR:\tTupel␣length␣{len(v)}␣differs␣from␣code␣length
        ␣{n}!")
40    exit(1)
41
42  for element in v:
43    if v == 0:
44      print("ERROR:\tElements␣of␣v␣must␣not␣be␣zero!")
45      exit(1)
46    else:
47      pass
48
49  y = input(f"\nPlease␣type␣in␣word␣tupel\n(r0,r1,...,r{n-1})␣of␣
      F_({q})^{n}:␣\t")
50  y = [F_q(x) for x in y[1:-1].split(",")]
51
52  if len(y) != n:
53    print("\nERROR:␣The␣provided␣codeword␣has␣not␣exactly␣n␣digits
        !")
54    exit()
55
56  # Generate Generator matrix G
57  rows = []
58  i = 0
59  while i < k:
60    row = []
61    j = 0
62    while j < n:
63      row.append(v[j]*(ae[j]**i))
64      j += 1
```

```
65    rows.append(row)
66    i += 1
67
68  G = matrix(F_q, rows)
69
70  # Compute Coefficients of the Parity-check matrix H
71  d = []
72  for i in range(len(v)):
73    d_c = ~v[i] * prod(~(ae[i] - ae[j]) for j in range(0, n) if i
          != j)
74    d.append(d_c)
75
76  print(f"\nParity-check␣Coefficients␣d:␣\t{d}")
77
78  # Syndrome computation
79  S = []
80  for i in range(0, n-k):
81    S_j = 0
82    for j in range (0, n):
83      S_j += y[j] * d[j] * ae[j]**i
84    S.append(S_j)
85
86  print(f"\nSyndromes:␣\t\t\t{S}")
87
88  if S[1] == 0 and S[1:].count(S[1]) == len(S[1:]):
89    print("\n\nINFO:␣The␣provided␣word␣is␣a␣correct␣codeword␣and␣
          was␣received␣error-free!")
90    exit(0)
91
92  R.<x> = PolynomialRing(F_q)
93
94  # Construct Syndrom-matrix M
95  mu = (n-k)//2
96  M = []
97  for i in range(0, n-k-mu):
98    row = []
99    for j in range(i, mu+i+1):
```

```
100       row.append(S[j])
101     M.append(row)
102 print(f"Matrix␣M_{mu}:␣\t\t\t{M}")
103 M = Matrix(R, M)
104 
105 print(f"\nMaximal␣number␣of␣Errors:␣\t{mu}")
106 
107 # Key Equation
108 # S_0 \sigma_e-1 + S_1 \sigma_e-2 + ... + S_e-1 \sigma_0 = 0
109 SigCoeff = M.right_kernel().basis()[0]
110 print(f"\nCoefficients␣of␣ELP␣\sigma:␣\t{list(SigCoeff)}")
111 
112 G.<z> = PolynomialRing(F_q)
113 sigma = sum(G(SigCoeff[i]) * z^(i) for i in range(0, mu+1))
114 
115 print(f"\nError-locator␣Polynomial:␣\t{sigma}")
116 
117 err_loc = []
118 for i in range(0, n):
119     if sigma(ae[i]) == 0:
120         print(f"Found␣an␣error␣at␣\t\t{i}")
121         err_loc.append(i)
122 
123 # Compute Error Magnitudes E_1, ..., E_v
124 S_m = []
125 X_m = []
126 for i in range(0, n-k):
127     S_m.append(S[i])
128     X_row = []
129     for j in range(0, len(err_loc)):
130         X_row.append(d[err_loc[j]] * ae[err_loc[j]]**i)
131     X_m.append(X_row)
132 
133 S_m = Matrix(R, S_m)
134 X_m = Matrix(R, X_m)
135 
136 E_m = X_m.solve_right(S_m.T)
```

```
137 E_m = E_m.list()
138
139 print(f"\nError magnitudes: \t\t{E_m}")
140
141 # Construct Error-Correction Codeword
142 e = []
143 for i in range(n):
144   if i in err_loc:
145     e.append(E_m[err_loc.index(i)])
146   else:
147     e.append(0)
148
149 print(f"Error-Correction Codeword e: \t{e}")
150
151 # Correct Errors
152 c = []
153 for i in range(n):
154   c.append(y[i] - e[i])
155
156 print(f"\nCorrected Codeword c: \t\t{c}")
```

Quelltext A.2: *SageMath*-Implementierung eines Dekodieralgorithmus für GRS-Codes

A.4 Implementierung des McEliece-Kryptosystems

```
1 #!/usr/bin/env sage
2 import json
3
4 # Key generation
5 def create_keys():
6   # Basic Goppa Code Properties
7   q = input("Please type in q, e.g. 2^2: \t\t")
8   n = int(input("Please type in length n: \t\t"))
9   k = int(input("Please type in dimension k: \t\t"))
10  t = int(input("Please type in error capacity t: \t"))
11
12  # Generate Galois field
```

```
13  F_q = None
14  if len(q.split("^")) < 2:
15          F_q.<a> = GF(int(q))
16  elif len(q.split("^")) == 2:
17    F_q.<a> = GF(pow(int(q.split("^")[0]), int(q.split("^")[1]))
        )
18  else:
19          print(f"ERROR:\tSorry, unable to work with this q={q}!
              ")
20
21  # Find a Goppa Polynom
22  R.<x> = PolynomialRing(F_q)
23  g = R.irreducible_element(t)
24
25  if input("\nShould a specific Goppa polynomial \nof degree t
      be used? [Y/n] \t\t") in ["Y", "y", ""]:
26    g = input("Please enter the Goppa polynomial \nwith x as
        variable: \t\t\t")
27    g = R(g)
28    if not g.is_irreducible():
29      print(f"\nERROR:\tGiven polynomial is reducible over F_{q}
          ")
30      exit()
31
32  print(f"\nGoppa polynomial: \t\t\t{g}")
33
34  # Specifiy set L of all non-roots of g
35  # In original McEliece approach, L denotes all elements of F_q
36  elements = Set()
37  for i in enumerate(F_q):
38    _, e = i
39    elements = elements.union(Set([e]))
40
41  roots = Set()
42  for i in g.roots():
43    r, _ = i
44    roots = roots.union(Set([r]))
```

```
45
46  L = elements.difference(roots)
47
48  print(f"\nSet␣of␣all␣elements:␣\t\t\t{elements}")
49  print(f"Set␣of␣all␣non-root␣elements:␣\t\t{L}")
50
51  # Construction of the Parity-check Matrix H
52  H_rarr = []
53  for i in range(0, k):
54    row = []
55    for j in L:
56      row.append((j**i)/g(j))
57    H_rarr.append(row)
58  H = Matrix(R, H_rarr)
59
60  print(f"\nParity-Check␣matrix␣H:␣\n{H}")
61
62  A = H.submatrix(0,0,k,k)
63  H_S = A.inverse() * H
64
65  print(f"\nSystematic␣Parity-check␣matrix␣H:␣\n{H_S}")
66
67  P = H_S.submatrix(0,k)
68  I_n_k = identity_matrix(F_q, n-k)
69  G = (-(P.T)).augment(I_n_k)
70
71  print(f"\nGenerator␣matrix␣G:␣\n{G}")
72
73  # Construction of G'
74  if input("\nShould␣a␣specific␣scrambling␣\nk␣x␣k-matrix␣S␣be␣
       used?␣[Y/n]\t\t") in ["Y", "y", ""]:
75    S = input("Please␣enter␣the␣S␣as␣a␣list␣of␣lists␣[[...],␣
         [...]]:\t\t")
76    print(S.split())
77    for i in list(S):
78      if i not in (['[', ']', ',', '␣', '^'] + [str(e) for e in
           elements]):
```

```
79          print("ERROR:\tUnable to process given S!")
80          exit()
81
82      S = Matrix(F_q, eval(S))
83
84  else:
85      S = random_matrix(F_q, k, density=1.0)
86
87  print(f"\nScrambling Matrix S: \n{S}")
88
89  if input("\nShould a specific permutation \nn x n-matrix P be
        used? [Y/n]\t\t") in ["Y", "y", ""]:
90      P = input(f"Please enter P as a column-index-list, e.g. [1,
            ..., {n}]:\t\t")
91      for i in list(P):
92          if i not in (['[', ']', ',', ' '] + [str(e) for e in range
                (n+1)]):
93          print("\nERROR:\tUnable to process given P!")
94          exit()
95      P = Permutation(eval(P)).to_matrix()
96
97  else:
98      P = Permutations(n).random_element().to_matrix()
99
100 print(f"\nPermutation Matrix P: \n{P}")
101
102
103 # Key generation
104 k_pub = S * G * P
105 print(f"\nPublic key: \n{k_pub}")
106
107 # Save Key files
108 if input(f"\nShould the PUBLIC key be saved as 'public_key.
        json'? [Y/n]\t\t") in ["Y", "y", ""]:
109     try:
110         f = open("public_key.json", "w")
111         f.write(json.dumps({"q": int(a.multiplicative_order()+1),
```

```
            "n": n, "t": t, "k": k, "key": str(k_pub.rows())}))
112         f.close()
113       except Exception as e:
114         print(e)
115         print("\nERROR:\tUnable to write keyfile! Check
              permissions and try again please.")
116
117     if input(f"\nShould the PRIVATE key be saved as 'private_key.
          json'? [Y/n]\t\t") in ["Y", "y", ""]:
118       try:
119         f = open("private_key.json", "w")
120         f.write(json.dumps({"q": int(a.multiplicative_order()+1),
              "n": n, "t": t, "k": k, "g": str(g), "P": str(P.rows())
              , "S": str(S.rows()), "G": str(G.rows())}))
121         f.close()
122
123       except Exception as e:
124         print(e)
125         print("\nERROR:\tUnable to write keyfile! Check
              permissions and try again please.")
126
127
128     return {"q": int(a.multiplicative_order()+1), "n": n, "k": k,
          "t": t, "g": g, "P": P, "S": S, "G": G, "G'": k_pub}
129
130
131 # Encryption
132 def encrypt(keys):
133     message = input("\nPlease type in the plaintext \nmessage as a
          list: \t\t\t")
134     message = eval(message)
135
136     F_q.<a> = GF(keys["q"])
137
138     k_pub = keys["G'"]
139     t = keys["t"]
140     k = keys["k"]
```

```
141
142     n = len(k_pub.columns())
143     while len(message) % k != 0:
144       message.append(0)
145
146     message_blocks = []
147     ciphertext_blocks = []
148
149     for i in range(0, len(message)//k):
150       message_blocks.append(vector(F_q, message[k*i:k+(k*i)]))
151
152     for block in message_blocks:
153       z = []
154       for i in range(n):
155         if i < t:
156           z.append(1)
157         else:
158           z.append(0)
159       z = vector(F_q, z) * Permutations(n).random_element().
             to_matrix()
160       bk = block * k_pub
161
162       encrypted_block = bk + z
163       ciphertext_blocks.append(encrypted_block)
164
165     return ciphertext_blocks
166
167
168 # Decryption (Placeholder)
169 def decrypt(keys):
170     print("\nERROR:\tDecryption␣is␣currently␣not␣implemented.")
171     exit()
172
173
174 if __name__ == "__main__":
175     print("###␣McEliece␣CRYPTPOSYSTEM␣ENCRYPTION")
176     print("###␣This␣sage␣script␣allows␣you␣to␣perform␣a␣McEliece␣
```

```
        cryptosystem␣encryption.")
177 print("###␣(C)␣Roman␣Wetenkamp,␣2023.\n")
178
179 # Search for valid key files
180 keys = {"q": None, "n": None, "t": None, "k": None, "g": None,
        "P": None, "S": None, "G": None, "G'": None}
181 mode = 0
182
183 try:
184   f = open("public_key.json", "r")
185   pubcon = json.loads(f.read())
186   keys["q"] = pubcon["q"]
187   F_q.<a> = GF(keys["q"])
188   R.<x> = PolynomialRing(F_q)
189
190   keys["n"] = pubcon["n"]
191   keys["t"] = pubcon["t"]
192   keys["k"] = pubcon["k"]
193
194   keys["G'"] = pubcon["key"]
195   keys["G'"] = keys["G'"].replace("^", "**")
196   keys["G'"] = Matrix(F_q, eval(keys["G'"], {"a": F_q.gen()}))
197
198   try:
199     h = open("private_key.json", "r")
200     content = json.loads(h.read())
201
202     content["g"] = content["g"].replace("^", "**")
203     content["S"] = content["S"].replace("^", "**")
204     content["G"] = content["G"].replace("^", "**")
205
206     keys["g"] = R(eval(content["g"], {"a": F_q.gen(), "x": R.
          gen()}))
207     keys["P"] = Matrix(F_q, eval(content["P"]))
208     keys["S"] = Matrix(F_q, eval(content["S"], {"a": F_q.gen()
          }))
209     keys["G"] = Matrix(F_q, eval(content["G"], {"a": F_q.gen()
```

```
            }))
210
211       except Exception as e:
212         print(e)
213         print("\nWARNING:\tNo␣private␣keyfile␣found␣-␣thus␣only␣
              encryption␣is␣possible")
214         mode = 1
215
216     except Exception as e:
217       print(e)
218       print("\nNo␣valid␣keyfile␣found␣-␣Key␣generation␣is␣starting
            ␣...\n")
219       keys = create_keys()
220
221     if mode == 0:
222       mode = int(input("Please␣choose␣a␣mode␣\n(1␣-␣Encryption,␣2␣
            -␣Decryption):\t"))
223
224     if mode == 1:
225       print(f"\nCiphertext␣blocks:\t\t\t{encrypt(keys)}")
226
227     if mode == 2:
228       decrypt(keys)
```

Quelltext A.3: *SageMath*-Implementierung des McEliece-Kryptosystems

Literaturverzeichnis

[1] B. Chor und R. Rivest, „A knapsack-type public key cryptosystem based on arithmetic in finite fields," *IEEE Transactions on Information Theory*, Jg. 34, Nr. 5, S. 901–909, 1988, Konferenzbeitrag CRYPTO 84. DOI: 10.1109/18.21214.

[2] R. J. McEliece, „A Public-Key Cryptosystem Based On Algebraic Coding Theory," *The Deep Space Network Progress Report*, Jg. 42-44, 1978, NASA Code 310-10-67-11. Adresse: https://ntrs.nasa.gov/api/citations/19780016269/downloads/19780016269.pdf (besucht am 15. 02. 2023).

[3] H. Niederreiter, „Knapsack-type Cryptosystems and Algebraic Coding Theory," *Problems of Control an Information Theory*, Jg. 15, Nr. 2, N. Krasovskii und G. Bognár, Hrsg., S. 159–166, 1986, ISSN: 0370-2529. Adresse: http://real-j.mtak.hu/7997/1/MTA_ProblemsOfControl_15.pdf (besucht am 14. 02. 2023).

[4] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*, 20th anniversary edition. Indianapolis: John Wiley und Sons, 2015, ISBN: 978-1-119-09672-6. Adresse: https://www.schneier.com/books/applied-cryptography-2preface/ (besucht am 06. 11. 2022).

[5] D. J. Bernstein, J. Buchmann und E. Dahmen, Hrsg., *Post-Quantum Cryptography*. Heidelberg: Springer, 2009, ISBN: 978-3-540-88701-0.

[6] V. Cerf, „ASCII format for Network Interchange," RFC Editor, RFC 20, Okt. 1969, S. 1–56. DOI: 10.17487/RFC0020.

[7] M. Borda, *Fundamentals in Information Theory and Coding*, 1. Aufl. Berlin: Springer, 2011. DOI: 10.1007/978-3-642-20347-3.

[8] J. H. van Lint, *Coding Theory*, Lecture Notes in Mathematics, A. Dold und B. Eckmann, Hrsg. Berlin: Springer, 1973, ISBN: 3-540-06363-3.

[9] N. L. Biggs, *Codes: An Introduction to Information Communication and Cryptography* (Springer Undergraduate Mathematics Series). London: Springer, 2008, DHBW-Bestand. DOI: 10.1007/978-1-84800-273-9.

[10] W. Willems, *Codierungstheorie und Kryptographie*. Basel: Birkhäuser, 2008. DOI: 10.1007/978-3-7643-8612-2.

[11] O. Manz, *Fehlerkorrigierende Codes*. Wiesbaden: Springer Vieweg, 2017. DOI: https://doi.org/10.1007/978-3-658-14652-8.

[12] J. L. Massey, „Some applications of source coding in cryptography," *European Transactions on Telecommunications*, Jg. 5, Nr. 4, S. 421–430, 1994. DOI: https://doi.org/10.1002/ett.4460050405.

[13] R. W. Hamming, „Error detecting and error correcting codes," *The Bell system technical journal*, Jg. 29, Nr. 2, S. 147–160, 1950. DOI: 10.1002/j.1538-7305.1950.tb00463.x.

[14] A. Viterbi, „Convolutional Codes and Their Performance in Communication Systems," *IEEE Transactions on Communication Technology*, Jg. 19, Nr. 5, S. 751–772, 1971. DOI: 10.1109/TCOM.1971.1090700.

[15] S. Roman, *Coding and Information Theory* (Graduate Texts in Mathematics). New York: Springer, 1992, Bd. 134, ISBN: 3-540-97812-7.

[16] R. Lidl und H. Niederreiter, „Algebraic Foundations," in *Finite Fields* (Encyclopedia of Mathematics and its Applications), 2. Aufl., Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1996, S. 1–46, UB-Bestand. DOI: 10.1017/CBO9780511525926.003.

[17] A. Myasnikov, A. Ushakov und V. Shpilrain, *Group-based Cryptography* (Advanced Courses in Mathematics - CRM Barcelona). Basel: Birkhäuser, 2008, DHBW-Bestand. DOI: 10.1007/978-3-7643-8827-0.

[18] J. M. Howie, *Fields and Galois Theory* (Springer Undergraduate Mathematics Series). London: Springer, 2006, UB-Bestand. DOI: 10.1007/978-1-84628-181-5.

[19] G. Fischer und B. Springborn, *Lineare Algebra, Eine Einführung für Studienanfänger* (Grundkurs Mathematik), 19. Aufl. Berlin: Springer, 2020. DOI: 10.1007/978-3-662-61645-1.

[20] G. Teschl und S. Teschl, *Mathematik für Informatiker, Band 1: Diskrete Mathematik und Lineare Algebra* (eXamen.press), 4. Aufl. Heidelberg: Springer, 2013. DOI: 10.1007/978-3-642-37972-7.

[21] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. New York: Wiley-Interscience, 2005, ISBN: 978-0-471-64800-0.

[22] S. B. Wicker und V. K. Bhargava, „An Introduction to Reed-Solomon Codes," in *Reed-Solomon Codes and Their Applications*. 1994, S. 1–16. DOI: 10.1109/9780470546345.ch1.

[23] C. Mahr, *Zur Codierung von RS-, BCH- und Goppa-Codes* (Fortschritt-Berichte 95). Düsseldorf: VDI-Verlag, 1988, Bd. 10, UB-Bestand (Magazin).

[24] N. Zivic, *Coding and Cryptography, Synergy for a Robust Communication*. München: Oldenbourg Wissenschaftsverlag, 2013, DHBW-Bestand. DOI: 10.1524/9783486781267.

[25] „10 Reed-Solomon and Justesen codes," in *The Theory of Error-Correcting Codes*, Ser. North-Holland Mathematical Library, F. MacWilliams und N. Sloane, Hrsg., Bd. 16, Elsevier, 1977, S. 294–316. DOI: https://doi.org/10.1016/S0924-6509(08)70535-X. Adresse: https://www.sciencedirect.com/science/article/pii/S092465090870535X.

[26] V. Weger, „A Code-Based Cryptosystem using GRS codes," Masterarbeit, Universität Zürich, Zürich, CH, 2017. Adresse: https://user.math.uzh.ch/weger/mathesis.pdf (besucht am 10. 02. 2023).

[27] „SageMath - Open-Source Mathematical Software System," SageMath Open Source Collective. (2022), Adresse: https://www.sagemath.org/ (besucht am 17. 01. 2023).

[28] W. C. Huffman und V. Pless, *Fundamentals of Error-Correcting Codes*. Cambridge, UK: Cambridge University Press, 2010, ISBN: 9780521131704. Adresse: https://archive.org/details/fundamentalsofer0000huff/ (besucht am 25. 01. 2023).

[29] R. Roth, *Introduction to Coding Theory*. Cambridge University Press, 2006. DOI: 10.1017/CBO9780511808968.

[30] „1 Linear codes," in *The Theory of Error-Correcting Codes*, Ser. North-Holland Mathematical Library, F. MacWilliams und N. Sloane, Hrsg., Bd. 16, Elsevier, 1977, S. 1–37. DOI: https://doi.org/10.1016/S0924-6509(08)70526-9. Adresse: https://www.sciencedirect.com/science/article/pii/S0924650908705269.

[31] D. G. Hoffman, D. A. Leonard, C. C. Lindner, K. T. Phelps, C. Rodger und J. R. Wall, *Coding Theory: The Essentials*. New York: Marcel Dekker, Inc., 1991, KIT-Bestand, ISBN: 0-8247-8611-4.

[32] B. H. Matzat. „Codierungstheorie." Vorlesungsskript, Universität Heidelberg. (2007), Adresse: https://www.math.uni-bielefeld.de/~baumeist/Codierungstheorie/matzat-Codierungstheorie.pdf (besucht am 10. 02. 2023).

[33] V. Guruswami, *List Decoding of Error-Correcting Codes, Winning Thesis of the 2002 ACM Doctoral Dissertation Competition* (Lecture Notes in Computer Science), 1. Aufl. Springer Berlin, Heidelberg, 2005. DOI: https://doi.org/10.1007/b104335.

[34] N. Sendrier, „Code-Based Cryptography," in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg und S. Jajodia, Hrsg. Boston, MA: Springer US, 2011, S. 215–216, ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_378. Adresse: https://doi.org/10.1007/978-1-4419-5906-5_378.

[35] V. M. Sidelnikov und S. O. Shestakov, „On insecurity of cryptosystems based on generalized Reed-Solomon codes," *Discrete Mathematics and Applications*, Jg. 2, Nr. 4, S. 439–444, 1992. DOI: doi:10.1515/dma.1992.2.4.439. Adresse: https://doi.org/10.1515/dma.1992.2.4.439.

[36] K. Kobara und H. Imai, „On the one-wayness against chosen-plaintext attacks of the Loidreau's modified McEliece PKC," *IEEE Transactions on Information Theory*, Jg. 49, Nr. 12, S. 3160–3168, 2003. DOI: 10.1109/TIT.2003.820016.

[37] N. T. Courtois, M. Finiasz und N. Sendrier, „How to Achieve a McEliece-Based Digital Signature Scheme," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, S. 157–174.

[38] R. Overbeck und N. Sendrier, „Code-based cryptography," in *Post-Quantum Cryptography*, D. J. Bernstein, J. Buchmann und E. Dahmen, Hrsg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, S. 95–145, ISBN: 978-3-540-88702-7. DOI: 10.1007/978-3-540-88702-7_4. Adresse: https://doi.org/10.1007/978-3-540-88702-7_4.

[39] W. Diffie und M. E. Hellman, „New Directions in Cryptography," *IEEE Transactions on Information Theory*, Jg. 22, Nr. 6, S. 644–654, 1976. Adresse: https://caislab.kaist.ac.kr/lecture/2010/spring/cs548/basic/B08.pdf (besucht am 15. 02. 2023).

[40] H. M. Salkin und C. A. De Kluyver, „The Knapsack Problem: A Survey," *Naval Research Logistics Quarterly*, Jg. 22, Nr. 1, S. 127–144, 1975. DOI: https://doi.org/10.1002/nav.3800220110. Adresse: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800220110.

[41] S. Pohlig und M. Hellman, „An improved algorithm for computing logarithms overGF(p)and its cryptographic significance (Corresp.)," *IEEE Transactions on Information Theory*, Jg. 24, Nr. 1, S. 106–110, 1978. DOI: 10.1109/TIT.1978.1055817.

[42] S. Gao, J. von zur Gathen, D. Panario und V. Shoup, „Algorithms for Exponentiation in Finite Fields," *Journal of Symbolic Computation*, Jg. 29, Nr. 6, S. 879–889, 2000, ISSN: 0747-7171. DOI: https://doi.org/10.1006/jsco.1999.0309. Adresse: https://www.sciencedirect.com/science/article/pii/S0747717199903097.

[43] N. Gura, A. Patel, A. Wander, H. Eberle und S. C. Shantz, „Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs," in *Cryptographic Hardware and Embedded Systems - CHES 2004*, M. Joye und J.-J. Quisquater, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, S. 119–132.

[44] E. Berlekamp, „Goppa codes," *IEEE Transactions on Information Theory*, Jg. 19, Nr. 5, S. 590–592, 1973. DOI: 10.1109/TIT.1973.1055088.

[45] F. D. Volta, M. Giorgetti und M. Sala, „Permutation equivalent maximal irreducible Goppa codes," 2008. DOI: 10.48550/ARXIV.0806.1763. Adresse: https://arxiv.org/abs/0806.1763.

[46] E. Berlekamp, R. McEliece und H. van Tilborg, „On the inherent intractability of certain coding problems (Corresp.)," *IEEE Transactions on Information Theory*, Jg. 24, Nr. 3, S. 384–386, 1978. DOI: 10.1109/TIT.1978.1055873.

[47] T. M. Qaradaghi und N. N. Abdulrazaq, *Comparison Between Irreducible and Separable Goppa Code in McEliece Cryptosystem*, Cryptology ePrint Archive, Paper 2015/1050, https://eprint.iacr.org/2015/1050, 2015. Adresse: https://eprint.iacr.org/2015/1050.

[48] R. Hill, „Introduction to linear codes," in *A First Course in Coding Theory*, Ser. Oxford Applied Mathematics and Computing Science Series, Oxford University Press, 1986, ISBN: 0-19-853804-9.

[49] N. Sendrier, „Finding the permutation between equivalent linear codes: the support splitting algorithm," *IEEE Transactions on Information Theory*, Jg. 46, Nr. 4, S. 1193–1203, 2000. DOI: 10.1109/18.850662.

[50] J. van Lint und T. Springer, „Generalized Reed - Solomon codes from algebraic geometry," *IEEE Transactions on Information Theory*, Jg. 33, Nr. 3, S. 305–309, 1987. DOI: 10.1109/TIT.1987.1057320.

[51] Y. X. Li, R. Deng und X. M. Wang, „On the equivalence of McEliece's and Niederreiter's public-key cryptosystems," *IEEE Transactions on Information Theory*, Jg. 40, Nr. 1, S. 271–273, 1994. DOI: 10.1109/18.272496.

[52] D. Engelbert, R. Overbeck und A. Schmidt, *A Summary of McEliece-Type Cryptosystems and their Security*, Cryptology ePrint Archive, Paper 2006/162, https://eprint.iacr.org/2006/162, 2006. Adresse: https://eprint.iacr.org/2006/162.

[53] E. Kiltz und J. Malone-Lee, „A General Construction of IND-CCA2 Secure Public Key Encryption," in *Cryptography and Coding*, K. G. Paterson, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, S. 152–166, ISBN: 978-3-540-40974-8.

[54] D. Bogdanov. „IND-CCA2 secure cryptosystems," University of Tartu. (2005), Adresse: https://kodu.ut.ee/~lipmaa/teaching/MTAT.07.006/2005/slides/S5.Bogdanov.indcca2.pdf (besucht am 02.04.2023).

[55] N. Dottling, R. Dowsley, J. Muller-Quade und A. C. A. Nascimento, „A CCA2 Secure Variant of the McEliece Cryptosystem," *IEEE Transactions on Information Theory*, Jg. 58, Nr. 10, S. 6672–6680, 2012. DOI: 10.1109/TIT.2012.2203582.

[56] E. Persichetti, „On the CCA2 Security of McEliece in the Standard Model," in *Provable Security*, J. Baek, W. Susilo und J. Kim, Hrsg., Cham: Springer International Publishing, 2018, S. 165–181.

[57] D. J. Bernstein, T. Lange und C. Peters, „Attacking and Defending the McEliece Cryptosystem," in *Post-Quantum Cryptography*, J. Buchmann und J. Ding, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, S. 31–46, ISBN: 978-3-540-88403-3.

[58] E. Prange, „The use of information sets in decoding cyclic codes," *IRE Transactions on Information Theory*, Jg. 8, Nr. 5, S. 5–9, 1962. DOI: 10.1109/TIT.1962.1057777.

[59] T. Lange. „Code-based cryptography V: Information-set decoding." Vorlesungsfolien. (2021), Adresse: `https://hyperelliptic.org/tanja/teaching/pqcrypto21/slides/cbc-mm-5.pdf` (besucht am 04.04.2023).

[60] C. Peters, „Information-Set Decoding for Linear Codes over Fq," in *Post-Quantum Cryptography*, N. Sendrier, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 81–94, ISBN: 978-3-642-12929-2.

[61] I. V. Maurich, T. Oder und T. Güneysu, „Implementing QC-MDPC McEliece Encryption," *ACM Trans. Embed. Comput. Syst.*, Jg. 14, Nr. 3, Apr. 2015, ISSN: 1539-9087. DOI: 10.1145/2700102. Adresse: `https://doi.org/10.1145/2700102`.

[62] T. P. Berger, P.-L. Cayrel, P. Gaborit und A. Otmani, „Reducing Key Length of the McEliece Cryptosystem," in *Progress in Cryptology – AFRICACRYPT 2009*, B. Preneel, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, S. 77–97, ISBN: 978-3-642-02384-2.

[63] H. Dinh, C. Moore und A. Russell, „McEliece and Niederreiter Cryptosystems That Resist Quantum Fourier Sampling Attacks," in *Advances in Cryptology – CRYPTO 2011*, P. Rogaway, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 761–779, ISBN: 978-3-642-22792-9.

[64] P. W. Shor, „Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Review*, Jg. 41, Nr. 2, S. 303–332, 1999. DOI: 10.1137/S0036144598347011.

[65] L. Hales und S. Hallgren, „Quantum Fourier sampling simplified," English (US), *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, S. 330–338, 1999, ISSN: 0734-9025. DOI: 10.1145/301250.301336.

[66] G. Alagic, D. Apon, D. Cooper et al., „Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process," National Institute of Standards und Technology, Bericht, 2022. DOI: 10.6028/NIST.IR.8413-upd1.

[67] W. Castryck und T. Decru, *An efficient key recovery attack on SIDH*, Cryptology ePrint Archive, Paper 2022/975, https://eprint.iacr.org/2022/975, 2022. Adresse: https://eprint.iacr.org/2022/975.

[68] R. Misoczki, J.-P. Tillich, N. Sendrier und P. S. L. M. Barreto, „MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes," in *2013 IEEE International Symposium on Information Theory*, 2013, S. 2069–2073. DOI: 10.1109/ISIT.2013.6620590.

[69] C. Aguilar-Melchor, O. Blazy, J.-C. Deneuville, P. Gaborit und G. Zémor, „Efficient Encryption From Random Quasi-Cyclic Codes," *IEEE Transactions on Information Theory*, Jg. 64, Nr. 5, S. 3927–3943, 2018. DOI: 10.1109/TIT.2018.2804444.