

**Master-Thesis**

**Post-Quantum Kryptographie –  
Evaluation Quantencomputer-resistenter  
Public-Key-Verfahren basierend auf der  
Implementierung auf klassischen  
Computern**

Autor: Jennifer Ritz



## Aufgabenstellung

Die Entwicklung von Quantencomputern nimmt rasant an Geschwindigkeit zu. Obwohl bereits seit 20 Jahren Forschung in diesem Bereich betrieben wird, ist eine Umsetzung eines effizienten Quantencomputers in naher Zukunft jedoch noch nicht zu erwarten. Dennoch ist es notwendig, bereits in der heutigen Zeit Quantencomputer-resistente Algorithmen zu entwerfen. Die gängigen Public-Key-Verschlüsselungsverfahren beruhen auf mathematischen Problemen – der Primfaktorzerlegung und des diskreten Logarithmus. Klassische Rechner, so wie sie heute im Einsatz sind, können diese Probleme bei ausreichender Schlüssellänge nicht in annehmbarer Zeit knacken. Anders sieht es bei Quantencomputern aus. Das bedeutet, dass jegliche Kommunikation, die heute stattfindet, aufgezeichnet und später mit einem Quantencomputer entschlüsselt werden kann – gemäß dem Motto „Store now – decrypt later“. Die Verschlüsselung von heute muss demnach dem Post-Quantenzeitalter standhalten können. Das Ziel der Masterthesis ist es, eine Auswahl an bereits veröffentlichten Quantencomputer-resistenten Public-Key-Algorithmen hinsichtlich definierter kryptographisch relevanter Bewertungskriterien auf einem klassischen Computer zu implementieren und miteinander zu vergleichen. Im Rahmen der Arbeit werden zunächst Grundlagen der Kryptographie aufgezeigt. Neben einem kurzen Abriss der Geschichte der Kryptographie werden die klassischen Verschlüsselungstechniken genauer erläutert. Des Weiteren muss ein fundiertes Verständnis für die Funktionsweise eines Quantencomputers vermittelt werden, um anschließend die Bedrohungen für die zuvor beschriebenen klassischen Verschlüsselungsverfahren darzulegen. Abschließend werden Quantencomputer-resistente Verfahren aufgezeigt, die sich in die Bereiche multivariate, codebasiert, gitterbasierte, hashbasierte und isogeniebasierte Kryptographie einordnen lassen. Als Resultat der zuvor beschriebenen Grundlagen wird eine Auswahl von den dargelegten Algorithmen beispielhaft auf einem klassischen Computer implementiert und aufgrund definierter Testszenarien miteinander verglichen und bewertet. Ziel dabei ist es, den für klassische Rechner geeignetsten Quantencomputer-resistenten Algorithmus zu ermitteln.

## **Kurzbeschreibung**

Im Fokus der vorliegenden Masterthesis steht der direkte Vergleich von Quantencomputer-resistenten Public-Key-Algorithmen nach definierten Kriterien. Diese Arbeit beschreibt zunächst Grundlagen zur klassischen Kryptographie und zur Funktionsweise von Quantencomputern. Des Weiteren werden Quantenalgorithmen aufgezeigt, die als Bedrohung für die Sicherheit der klassischen Verfahren betrachtet werden. Bei der Evaluation von Public-Key-Algorithmen der Post-Quantum Kryptographie werden Verfahren aus den verschiedenen zuvor dargelegten Kryptographie-Bereichen ausgewählt. Anhand bereits veröffentlichter Quellcodes werden die Verfahren auf einem klassischen Computer so implementiert, dass die Vergleichbarkeit der durchgeführten Testszenarien gewährleistet ist. Ziel der Evaluation ist es, eine Empfehlung auf Basis der definierten Bewertungskriterien und der ermittelten Testergebnisse der durchgeführten Testszenarien auszusprechen. Mittels der modifizierten Quellcodes ist es zudem möglich, die Testszenarien auf beliebigen klassischen Rechner durchzuführen und die Algorithmen mit verschiedenen Eingangsparametern miteinander zu vergleichen.

## **Abstract**

The focus of this master thesis is a direct comparison of quantum computer resistant public key algorithms according to defined criteria. The thesis will begin by describing the basics of classic cryptography and the functionality of quantum computers. It will further highlight various quantum algorithms which are considered to be a threat to the security of classic methods. A number of methods from the previously mentioned areas of cryptography will be chosen to evaluate public key algorithms of the post-quantum cryptography. Through the use of publicly available source code, those methods will be implemented on a classic computer in such a way as to guarantee the comparability of the different test scenarios. The objective of this evaluation is to make a recommendation based on the defined evaluation criteria and the results of the executed test scenarios. Additionally, using the modified source code makes it possible to execute those test scenarios on any classic computer and compare the algorithms using different input parameters.

## Inhalt

Aufgabenstellung .....	2
Kurzbeschreibung .....	3
1 Einführung .....	6
1.1 Vorgehensweise .....	7
1.2 Definition der Forschungsfrage .....	8
1.3 Abgrenzung .....	8
2 Klassische Kryptographie .....	9
2.1 Elementare Verschlüsselungsverfahren .....	9
2.2 Symmetrische Verschlüsselungsverfahren .....	11
2.2.1 DES-Verfahren .....	12
2.2.2 AES-Verfahren .....	13
2.2.3 Vergleich der Verfahren .....	13
2.3 Asymmetrische Verschlüsselungsverfahren .....	14
2.3.1 RSA-Algorithmus .....	15
2.3.2 ElGamal-Verfahren .....	17
2.3.3 Operationen auf elliptischen Kurven .....	19
2.3.4 Vergleich der Verfahren .....	21
2.4 Hashfunktionen .....	22
2.4.1 MD5-Hashfunktion und SHA-Algorithmus .....	23
2.4.2 MAC und HMAC .....	24
2.4.3 Vergleich der Verfahren .....	25
2.5 Klassische Angriffsmöglichkeiten .....	26
3 Post-Quantum Kryptographie .....	28
3.1 Grundlagen eines Quantencomputers .....	29
3.1.1 Definition Quantenbit .....	29
3.1.2 Definition Quantenregister .....	30
3.1.3 Quanten-Fouriertransformation .....	31
3.1.4 Deutsch-Jozsa-Algorithmus .....	32
3.2 Quantenalgorithmen als Bedrohung für klassische Kryptographie .....	32
3.2.1 Grover-Algorithmus .....	33
3.2.2 Simon-Algorithmus .....	33
3.2.3 Shor-Algorithmus .....	34
3.3 Quantencomputer-resistente Public-Key-Algorithmen .....	35
3.3.1 Multivariate Kryptographie .....	35
3.3.2 Codebasierte Kryptographie .....	39
3.3.3 Gitterbasierte Kryptographie .....	42

3.3.4	Hashbasierte Kryptographie .....	45
3.3.5	Isogeniebasierte Kryptographie .....	49
3.3.6	Vergleich der Post-Quantum Kryptographie-Kategorien .....	51
4	Evaluation ausgewählter Quantencomputer-resistenter Public-Key-Verfahren.....	56
4.1	Rahmenbedingungen der Evaluation .....	58
4.1.1	Definition der Bewertungskriterien.....	58
4.1.2	Auswahl der Quantencomputer-resistenten Algorithmen.....	59
4.1.3	Aufbau der Analyseumgebung .....	60
4.2	Referenzimplementierungen.....	61
4.2.1	Ver- und Entschlüsselungsverfahren.....	61
4.2.2	Signaturerstellungungsverfahren.....	63
4.2.3	Schlüsselaustauschverfahren.....	66
4.3	Testszzenarien.....	71
4.3.1	Definition und Vorbereitung der Testszzenarien .....	72
4.3.2	Herausforderungen.....	73
4.3.3	Testergebnisse .....	74
4.4	Bewertung der Ergebnisse.....	80
4.4.1	Ver- und Entschlüsselungsverfahren.....	80
4.4.2	Signaturerstellungungsverfahren.....	81
4.4.3	Schlüsselaustauschverfahren.....	83
5	Fazit und Ausblick .....	85
6	Literaturverzeichnis .....	87
7	Bilderverzeichnis .....	91
8	Tabellenverzeichnis.....	92
9	Anlagenverzeichnis und Anlagen.....	93
10	Verzeichnis der Abkürzungen .....	114
11	Selbstständigkeitserklärung .....	116
12	Thesen .....	117

## 1 Einführung

Vor einigen Jahrzehnten galt ein sogenannter Quantencomputer, der aufgrund der zugrundeliegenden Quantenmechanik leistungsfähiger als jeder Supercomputer sein soll, noch als Produkt der fernen Zukunft. Im Jahr 2019 war es jedoch bereits so weit, dass Quantencomputer kommerziell erworben oder die Rechenleistung eines Quantencomputers in der Cloud genutzt werden können. Vorreiter für die Forschung sind hier die namenhaften Hersteller IBM, Google, Microsoft und Amazon. IBM beispielsweise arbeitet bereits seit 1981 intensiv an der Entwicklung von kommerziellen Quantensystemen [1]. Im Jahr 2016 wurde von IBM der erste Quantencomputer weltweit in der Cloud angeboten. Dabei ist es den Nutzern mit dem sogenannten „IBM Q Experience“ möglich, mit einem Quantensystem auf Basis von 16 Quantenbits (Qubits) zu experimentieren. Laut IBM wurden von mehr als 75.000 Nutzern über 2,5 Millionen Experimente durchgeführt. Darunter waren zahlreiche Wissenschaftler, die daraus 35 wissenschaftliche Arbeiten hervorbrachten. Anfang des Jahres 2019 wurde der „IBM Q System One“ vorgestellt, welcher bereits mit 20 Qubits arbeitet. 20 Qubits galt dabei lange Zeit als Grenze für einen funktionierenden Quantencomputer [2]. Google sowie IBM können Ende 2019 bereits Quantensysteme vorweisen, die Rechenoperationen mit über 50 Qubits durchführen können und übertreffen damit den aktuell stärksten Supercomputer [3]. Bis 2021 sollen Rechner mit einer Leistung von über 100 Qubits entwickelt werden. Die Rechengeschwindigkeiten dieser Rechner sind enorm. Die zur heutigen Zeit verwendeten Verschlüsselungsalgorithmen basieren auf der Problematik, diese in annehmbarer Zeit knacken zu können. Mit der Entwicklung eines ausreichend performanten Quantencomputers ist eine Entschlüsselung innerhalb von wenigen Sekunden denkbar. Bereits in den 90er Jahren wurde der Shor-Algorithmus veröffentlicht, der eine große Bedrohung für gegenwärtige asymmetrische Kryptosysteme darstellt [4]. Durch Quantenalgorithmen sind weitere Möglichkeiten zum Brechen von symmetrischen Verschlüsselungsverfahren ein potentielles Risiko. Die Notwendigkeit Quantencomputer-resistente Verschlüsselungsverfahren zu verwenden, wird zunehmend relevanter. Jedoch werden diese Verfahren nicht erst zum Zeitpunkt des Einsatzes von geeigneten Quantencomputern benötigt, sondern sollten bereits heute in bestehende Kommunikationsverfahren implementiert werden. Der Hintergrund dabei ist, dass die heutige Kommunikation aufgezeichnet und zu einem späteren Zeitpunkt mittels Quantenalgorithmen entschlüsselt werden könnte. Demnach können

Angreifer gemäß dem Motto „Store now – decrypt later“ vorgehen [5]. Deshalb sollten Quantencomputer-resistente Algorithmen Eigenschaften aufweisen, um performant und ausreichend sicher auf klassischen Computern implementiert werden zu können.

## **1.1 Vorgehensweise**

Die Vorgehensweise dieser Masterthesis zielt darauf ab, die Stärken und Schwächen der jeweiligen Bereiche der Post-Quantum Kryptographie auf Basis der theoretischen Analyse von Quantencomputer-resistenten Public-Key-Verfahren und der praktischen Implementierung ausgewählter Algorithmen zu ermitteln. Zunächst werden in Kapitel 2 die Grundlagen der klassischen Kryptographie erläutert. Diese lässt sich in die Bereiche symmetrische und asymmetrische Verschlüsselungsverfahren sowie Hashfunktionen unterteilen. Abschließend werden klassische Angriffsmöglichkeiten aufgezeigt, um daraufhin in Kapitel 3 die Bedrohung durch Quantenalgorithmen, wie beispielsweise dem Shor-Algorithmus, zu verdeutlichen. In diesem Zuge werden zum besseren Verständnis grundlegende Begriffe und Funktionen von Quantencomputern erläutert. Anschließend werden die fünf Bereiche, in die sich die Quantencomputer-resistenten Public-Key-Verfahren einordnen lassen, detailliert anhand ihrer zeitlichen Entwicklung beschrieben. Die Bereiche multivariate, codebasierte, gitterbasierte, hashbasierte und isogeniebasierte Kryptographie unterscheiden sich durch ihre mathematischen Grundlagen, die ausschlaggebend für die Stärken und Schwächen der Verfahren sind. Am Schluss des Kapitels werden die Bereiche anhand ihrer theoretischen Eigenschaften gegenübergestellt. In Kapitel 4 werden ausgewählte Algorithmen aus den verschiedenen Bereichen in einer definierten Testumgebung implementiert. Es gilt zu prüfen, welche Algorithmen anhand einer praktischen Implementierung auf klassischen Computern empfehlenswert sind. Mithilfe von bereits veröffentlichten Referenzimplementierungen wird der Quellcode insofern angepasst, dass eine vergleichbare Evaluation der definierten Messkriterien durchgeführt werden kann. Auf Basis der festgelegten Bewertungskriterien und der Testergebnisse der durchgeführten Testszenarien wird ein Vergleich der Verfahren vorgenommen, um abschließend in Kapitel 5 ein Fazit aus den Ergebnissen zu ziehen.



## 1.2 Definition der Forschungsfrage

Das Ziel dieser Arbeit ist es, eine Antwort auf die Frage nach dem besten Quantencomputer-resistenten Public-Key-Verfahren für den Einsatz auf klassischen Computern zu geben. Mithilfe relevanter theoretischer Grundlagen und der Durchführung von praktischen Testszenarien soll in Anbetracht des aktuellen Stands der Forschung eine Empfehlung ausgesprochen werden.

## 1.3 Abgrenzung

Der Fokus dieser Arbeit liegt auf dem Vergleich der asymmetrischen Verfahren der Post-Quantum Kryptographie. Die bevorstehende Bedrohung durch Quantenalgorithmen ist besonders für Public-Key-Verfahren von Bedeutung, weshalb symmetrische Verfahren der Vollständigkeit halber in den Grundlagen zwar erläutert werden, aber für die durchgeführte Evaluation nur eine nebensächliche Rolle spielen. Aufgrund der für das Verständnis relevanten Grundlagen werden neben dem Fachgebiet Informatik auch das des Ingenieurwesens und der Elektrotechnik betrachtet. Jedoch wird der Forschungsbereich der Quantencomputer nur am Rande behandelt und nicht im Detail dargelegt. Zudem findet kein Vergleich zwischen klassischen und Post-Quantum Algorithmen statt. Die Erläuterung der klassischen Kryptographie dient lediglich der Vermittlung der fundierten mathematischen Kenntnisse. Zudem soll verdeutlicht werden, dass die ausgehende Bedrohung von Quantencomputern eine zeitnahe Entwicklung von resistenten Verfahren erfordert. Die Post-Quantum Kryptographie bietet ein breites Spektrum an Public-Key-Algorithmen. Die Evaluation beschränkt sich daher auf eine Auswahl von veröffentlichten Algorithmen und jeweils möglichen Eingangsparametern und stellt somit ein Proof of Concept (POC) dar. Ein allumfänglicher Test von möglichen Algorithmen und Parametern auf verschiedensten klassischen Computerprozessoren würde den Rahmen dieser Arbeit überschreiten. Jedoch wird durch diese Arbeit die Möglichkeit geboten, analoge Testszenarien mit weiteren Parametern auf beliebigen Rechnern durchzuführen.

## 2 Klassische Kryptographie

Zu Beginn werden grundlegende Begriffe und Definitionen erläutert, um ein einheitliches Verständnis der nachfolgenden Kapitel zu gewährleisten. Als „Kryptographie“ wird die Wissenschaft bezeichnet, die sich mit den Methoden der Verschlüsselung und Entschlüsselung von Informationen befasst [6, S. 1]. Dabei wird ein sogenannter Klartext mittels eines Schlüssels zu einem Geheimtext chiffriert. Zum Dechiffrieren wird entweder der gleiche Schlüssel oder ein Schlüsselpaar benötigt. Wird die Begriffsdefinition von Kryptographie des BSI betrachtet, so stellen „[s]ichere kryptografische Verfahren einen unverzichtbaren Grundbaustein für IT-Sicherheitsmechanismen zur Wahrung von Vertraulichkeit, Integrität und Authentizität digitaler Informationen dar.“ Hierbei wird bereits auf die Ziele der Kryptographie eingegangen. Vertraulichkeit liegt dann vor, wenn nur der legitimierte Empfänger Einsicht in den Klartext der Nachricht erhält. Unter Integrität wird die Vollständigkeit und Unveränderbarkeit einer Nachricht verstanden. Die Authentizität einer Nachricht soll ebenfalls jederzeit nachweisbar sein. Das heißt, dass der Sender einer Nachricht eindeutig identifizierbar sein muss. In diesem Zusammenhang spielt auch das Ziel Verbindlichkeit eine große Rolle. Der Sender darf nicht abstreiten können, eine Nachricht gesendet zu haben [7].

Die ersten Hinweise für die Verwendung von kryptographischen Verschlüsselungen in der Geschichte lassen bereits vor über ca. 2500 Jahren finden [8, S. 3]. Dabei wurden Nachrichten mittels eines Zylinders, der sogenannten Skytale, chiffriert. Nur wenn Sender und Empfänger die gleiche Skytale verwendeten, konnten die Nachrichten ver- und wieder entschlüsselt werden. Weitere bekannte Beispiele in der Geschichte sind die Cäsar-Verschlüsselung, die Vigenère-Verschlüsselung und das One-Time-Pad, die auf den nachfolgend erläuterten elementaren kryptographischen Verfahren basieren. Mit der Erfindung des Computers ist eine Weiterentwicklung der Kryptographie-Verfahren unabdingbar gewesen. Die Internetkommunikation erforderte im Vergleich zu damaligen Verfahren eine erheblich höhere Komplexität und benötigte Rechenleistung, um die Sicherheit der Übermittlung zu gewährleisten.

### 2.1 Elementare Verschlüsselungsverfahren

Bei den elementaren Verschlüsselungsverfahren handelt es sich um grundlegende Vorgehensweisen, die jeder Verschlüsselungsfunktion als Basis dienen. Zunächst wird

bei diesen Verfahren die monoalphabetische Substitution betrachtet [9, S. 54]. Das bedeutet, dass jeder Klartextbuchstabe durch einen Geheimtextbuchstaben ersetzt beziehungsweise substituiert wird. Wie die Substitution zu erfolgen hat, ist zwischen Sender und Empfänger festzulegen. Hierbei wird eine Substitutionstabelle verwendet. Ein anschauliches Beispiel ist die Cäsar-Verschlüsselung. Dabei wird das aus 26 Buchstaben bestehende lateinische Alphabet auf ein Alphabet abgebildet, das um den Schlüssel  $k$  verschoben wurde [10, S. 56f.]. Diese Vorgehensweise führt zu folgenden allgemeinen Formel, wobei  $c$  der Geheimtext,  $m$  der Klartext und  $E()$  die Verschlüsselungsfunktion darstellt:

$$c = E_k(m) = m + k \pmod{26} \quad (2.1)$$

Um eine Nachricht wieder zu dechiffrieren, muss die Verschiebung umgekehrt werden. Die Entschlüsselungsfunktion lautet wie folgt:

$$D_k(c) = m = c - k \pmod{26} \quad (2.2)$$

Das Knacken einer solchen Verschlüsselung gilt als trivial. Da es lediglich 25 mögliche Schlüssel gibt, können diese einfach nacheinander getestet werden, bis ein sinnvoller Klartext erscheint. Wird ein Substitutionsverfahren angewendet, das nicht auf einer Verschiebung des Alphabets, sondern auf einer zufällig generierten Buchstabenfolge basiert, kann die Verteilung der Buchstaben durch Kryptoanalyse ermittelt werden [9, S. 55]. Anhand der Häufigkeit der Verteilung der Buchstaben des deutschen Alphabets können die im Geheimtext auftretenden Häufigkeiten damit korreliert werden. Eine Abbildung von Geheimtextbuchstabe auf Klartextbuchstabe ist somit durchführbar, da die prozentualen Häufigkeiten je Buchstaben auch im Geheimtext erhalten bleiben.

Eine Erweiterung der monoalphabetischen Verschlüsselung stellt die polyalphabetische Verschlüsselung dar. Dabei wird nicht nur eine Zeile einer Substitutionstabelle verwendet, sondern mehrere Zeilen. Der Schlüssel ist somit keine einzelne Zahl, wie bei der Cäsar-Verschlüsselung, sondern kann aus mehreren Zeichen beziehungsweise einem Wort bestehen. Eines der bekanntesten Verfahren ist dabei die Vigenère-Verschlüsselung. Auch bei den polyalphabetischen Verfahren ist es durch Kryptoanalyse möglich, den Klartext ohne den Schlüssel zu ermitteln. Anhand von auftretenden Regelmäßigkeiten im Geheimtext können statistische Rückschlüsse auf den Schlüssel gezogen werden.

Die einzig sichere Methode ist das sogenannte One-Time-Pad. Die Sicherheit des Verfahrens beruht einzig auf dem Schlüssel. Dieser muss jedoch folgende Kriterien erfüllen, um statistischen Analysen standzuhalten:

- Er muss die gleiche Länge besitzen wie der Klartext.
- Er muss zufällig erstellt worden sein.
- Er darf nur einmal verwendet werden.

Die Probleme sind hier schnell erkennbar. Bei sehr langen Nachrichten hat der Schlüssel eine enorme Länge, der dazu noch geheim an den Empfänger übertragen werden muss – bei jeder Nachricht erneut, um die Einmaligkeit zu gewährleisten. Wird der Schlüssel bei der Übertragung abgefangen, ist die Verschlüsselung hinfällig. Zudem ist es nahezu unmöglich, einen rein zufälligen Schlüssel dieser Länge zu erzeugen, da beispielsweise nur Würfeln oder der Münzwurf als zufällig gelten [10, S. 61f.].

Abschließend werden die Transpositionschiffren erläutert. Dabei werden Klartextbuchstaben oder auch Bits mittels einer zweidimensionalen Matrix auf einen Geheimtext abgebildet. Es wird die Methode des Vertauschens, die sogenannte Permutation, angewendet. Das bedeutet, die Buchstaben bleiben zwar die gleichen, sind jedoch nicht mehr an derselben Stelle. Hierbei ist die zuvor erwähnte Skytale als eine Spaltentransposition zu nennen. Dadurch, dass der Klartext um einen Zylinder gewickelt wird, wird der Text in Spalten abgebildet, die dann vertikal gelesen einen Geheimtext ergeben. Um sich vor Angreifern zu schützen, sollten häufig unterschiedliche Größen der Skytale gewählt werden. Jedoch können diese Chiffren ebenfalls durch Kryptanalyse, wie beispielsweise durch Anagramming, entschlüsselt werden. Es werden demnach die Buchstaben durch Tafeln für die Häufigkeit von Digrammen, wie das Vorkommen von „ch“, wieder an die korrekten Stellen gebracht [6, S. 15ff.].

## **2.2 Symmetrische Verschlüsselungsverfahren**

Die im vorherigen Kapitel erläuterten Verfahren können ausschließlich den symmetrischen Verschlüsselungsverfahren zugeordnet werden. Das Kernprinzip dieser Algorithmen liegt an der Verwendung eines identischen Schlüssels. Sender und Empfänger müssen zur Ver- und Entschlüsselung von Nachrichten den gleichen Schlüssel besitzen (vgl. Bild 1).



**Bild 1: Ablauf einer symmetrischen Verschlüsselung**

Problematisch ist hierbei die Übermittlung des Schlüssels. Denn dieser muss, wie die Nachricht selbst, unverändert, geheim und nachweisbar vom richtigen Empfänger übertragen werden. Zudem muss der Schlüssel bei ständiger Kommunikation regelmäßig ausgetauscht werden, um statistischen Analysen standzuhalten. Ebenfalls ist die Schlüsselgenerierung und Schlüsselverwaltung bei mehreren Kommunikationspartnern nicht zu vernachlässigen. Dabei gilt bei  $n$  Kommunikationspartnern eine Schlüsselmenge von:

$$n = \frac{(n-1)}{2} \quad (2.3)$$

Dennoch kann durch Produktverschlüsselung eine hinreichende Sicherheit gewährleistet werden [9, S. 75].

### 2.2.1 DES-Verfahren

Eines der bekanntesten symmetrischen Verfahren stellt der „Data Encryption Standard“ (DES) dar. Dabei handelt es sich um eine Block-Produktverschlüsselung. DES wurde in den 1970ern von einem IBM-Forscherteam entwickelt und beruht auf einer nichtlinearen Substitution und einer Permutation. Bei der Verschlüsselung werden 16 Iterationschleifen durchgeführt, die mittels eines Schlüssels den Klartext blockweise chiffrieren. Der Vorteil dabei ist, dass die kleinsten Veränderungen am Schlüssel oder am Klartext einen vollkommen anderen Geheimtext erzeugen. Zu Beginn wurde das Verfahren mit einer Schlüssellänge von 56 Bit angewendet. Jedoch konnten die Schlüssel dadurch leicht durch Brute-Force-Attacken, also Ausprobieren aller möglichen Schlüssel, geknackt werden. Da die Implementierbarkeit von DES in Hardwaremodulen gut

geeignet war, wurde der DES stets weiterentwickelt. Die letzte Entwicklung ist das Triple-DES-Verfahren, das mit einer Schlüssellänge von 112 Bit arbeitet. Heutzutage wird dieses Verfahren nicht mehr empfohlen, da die verwendete Schlüssellänge mittlerweile ein hohes Sicherheitsrisiko darstellt [9, S. 61f.].

### **2.2.2 AES-Verfahren**

Das National Institute of Standards (NIST) startete im Jahr 1997 einen Auswahlprozess für einen neuen Verschlüsselungsstandard. Ergebnis des Prozesses war der Advanced Encryption Standard (AES), dessen Funktionsweise dem des DES ähnelt. Der Klartext wird dabei zunächst in Blöcke fester Längen aufgeteilt. Diese können eine Länge von 128 Bit, 192 Bit oder 256 Bit besitzen. Anschließend werden, analog zum DES, Transformationsrunden durchgeführt, die je nach Länge des Blocks oder Schlüssels, zehn, zwölf oder vierzehn Runden betragen können. Der erste Schritt einer Runde stellt eine monoalphabetische Substitution auf Basis eines festgelegten Arrays dar (ByteSub-Transformation). Der zweite Schritt ist eine Permutation, bei der die Zeilen des Arrays um maximal vier Spalten nach links verschoben werden (ShiftRow-Transformation). Die erste Zeile wird dabei vernachlässigt und alle übergelaufenen Zeilen werden von rechts fortgesetzt. Im dritten Schritt werden alle Spalten jeweils mit einem festen Polynom multipliziert (MixColumn-Transformation). Abschließend wird die AddRoundKey-Transformation durchgeführt, wobei der aus dem Schlüssel ermittelte Rundenschlüssel mit dem Array bitweise XOR verknüpft wird. Da der AES-Algorithmus auf elementaren Verschlüsselungsverfahren basiert, die lediglich aufgrund der angewendeten Reihenfolge eine hinreichende Sicherheit bieten, ist eine Verschlüsselung schnell und flexibel durchführbar [9, S. 62ff.]. Wird die Empfehlung zu den Schlüssellängen des BSI betrachtet, so wird zum Einsatz von AES mit den Längen 128 Bit, 192 Bit und 256 Bit geraten. Aktuell gelten als besondere Schwachstelle des AES die sogenannten Related-Key-Angriffe. Dabei liegen dem Angreifer Informationen über Ver- und Entschlüsselungen bekannter oder gewählter Klartexte vor, die mit unterschiedlichen Schlüsseln verschlüsselt wurden, aber zueinander in Beziehung stehen [11, S. 22f.].

### **2.2.3 Vergleich der Verfahren**

Werden die zuvor beschriebenen Verfahren verglichen, kann zusammenfassend folgende Tabelle 1 erstellt werden. Dabei wird der Aufwand zum Brechen in der Anzahl der benötigten Rechenschritte aufgezeigt [12].

Tabelle 1: Vergleich von symmetrischen Verfahren

Symmetrische Verfahren	DES	3DES	AES
Schlüssellänge (Bit)	56	168	128 / 192 / 256
Aufwand zum Brechen (in Rechenschritten)	$2^{39}$	$2^{112}$	$2^{126,1} / 2^{169,7} / 2^{254,4}$
Sicherheit	niedrig	mittel	hoch / sehr hoch / maximal

Anhand dessen ist erkennbar, dass für eine ausreichende Sicherheit, der AES-Algorithmus verwendet werden sollte. Dennoch ist nicht zu vernachlässigen, dass der DES aufgrund seiner einfachen Implementierbarkeit in Hardwaremodulen in Kombination mit weiteren Sicherheitsvorkehrungen weiterhin empfehlenswert ist.

### 2.3 Asymmetrische Verschlüsselungsverfahren

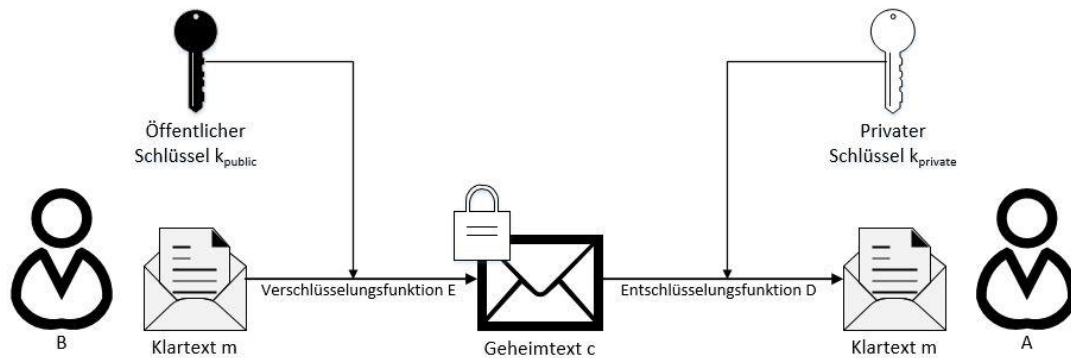
Wie bereits erläutert, stellt das größte Problem symmetrischer Verschlüsselungsverfahren der Schlüsselaustausch zwischen Sender und Empfänger dar. Aus diesem Grund wurden asymmetrische Verfahren entwickelt, sogenannte Public-Key-Kryptosysteme, die erstmals 1976 von Whitfield Diffie und Martin Hellman eingeführt wurden. Diese Kryptosysteme verwenden nicht nur einen Schlüssel, sondern ein Schlüsselpaar zur Ver- und Entschlüsselung [6, S. 73]. Die beiden Schlüssel stehen miteinander in Verbindung, wie nachfolgend anhand von Beispielen genauer erläutert wird. Damit das Kryptosystem funktioniert, ist es notwendig sicherzustellen, dass der private Schlüssel nicht aus dem öffentlichen abgeleitet werden kann [9, S. 77f.]. Deshalb beruhen die Algorithmen auf mathematischen Problemen, für die es aktuell keine effizienten Lösungen gibt. Dazu zählen beispielsweise die Primfaktorzerlegung und der diskrete Logarithmus. Die Funktionsweise der asymmetrischen Algorithmen lässt sich grundlegend wie folgt beschreiben (vgl. Bild 2):

Zur Verschlüsselung nutzt der Sender den öffentlich zugänglichen Schlüssel (public key)  $k_{public}$ , um seine Nachricht  $m$  zu chiffrieren.

$$c = E_{k_{public}}(m) \quad (2.4)$$

Zum Dechiffrieren nutzt nun der Empfänger seinen geheimen privaten Schlüssel (private key)  $k_{private}$ .

$$m = D_{k_{private}}(c) \quad (2.5)$$



**Bild 2: Ablauf einer asymmetrischen Verschlüsselung**

Asymmetrische Verfahren können dadurch auch zur digitalen Signatur verwendet werden. Dazu wird eine Nachricht mit dem privaten Schlüssel verschlüsselt und erzeugt damit die Signatur, die der Original-Nachricht vor der Übertragung angehängt wird. Der Empfänger kann daraufhin mittels des öffentlichen Schlüssels, diese beigefügte Signatur entschlüsseln. Anschließend wird diese mit der empfangenen Nachricht verglichen. Stimmen die beiden Nachrichten überein, ist die Authentizität des Senders bestätigt [9, S. 79].

Nachfolgend werden die bekanntesten Public-Key-Algorithmen dargestellt, um die Vor- und Nachteile der heutigen Verschlüsselung zu erläutern.

### 2.3.1 RSA-Algorithmus

Das bekannteste Verfahren ist der sogenannte Rivest-Shamir-Adleman-Algorithmus (RSA-Algorithmus). Dieser wurde 1977 von den Namensgebern Ron Rivest, Adi Shamir und Leonard Adleman veröffentlicht und zählt heute zu den am häufigsten genutzten Public-Key-Verfahren [6, S. 77ff.]. Der Algorithmus basiert auf dem Problem der Primfaktorzerlegung und wird im Detail wie folgt durchgeführt:

1. Schlüsselgenerierung:



- Zunächst muss das Schlüsselpaar durch den Empfänger erzeugt werden. Dabei werden zwei große Primzahlen  $p$  und  $q$  von etwa gleicher Länge festgelegt.
  - Anschließend wird das Produkt  $n = p \cdot q$  berechnet und eine teilerfremde Zahl  $e$  zu  $\varphi(n) = (p - 1)(q - 1)$  gewählt.
  - Der öffentliche Schlüssel setzt sich somit aus dem Produkt  $n$  und der Zahl  $e$  zusammen:  $k_{public} = (n, e)$ .
  - Daraufhin kann der private Schlüssel  $k_{private}$  errechnet werden:  $d = e^{-1} \text{ mod } \varphi(n)$ .
2. Verschlüsselung:
- Die Verschlüsselung einer Nachricht mit dem öffentlichen Schlüssel ist mit folgender Funktion durchzuführen:  $c = m^e \text{ mod } n$ .
3. Entschlüsselung:
- Mittels des privaten Schlüssels lässt sich die Nachricht vom Empfänger dechiffrieren:  $m = c^d \text{ mod } n$ .

Um zu beweisen, dass die Ver- und Entschlüsselung nach dieser Vorgehensweise funktioniert, müssen die mathematischen Grundlagen genauer betrachtet werden. Zunächst müssen Prämissen festgelegt werden [6, S. 78]:

- $n = p \cdot q \in \mathbb{N}$  mit den Primzahlen  $p$  und  $q$ , wobei  $p \neq q$
- $e$  und  $d \in \mathbb{N}$ , wobei  $e \cdot d \text{ mod } \varphi(n) = 1$
- Nachricht  $m \in \mathbb{Z}_n$
- Daraus folgt:  $(m^e \text{ mod } n)^d \text{ mod } n = m$

Zusätzlich muss bewiesen werden, dass  $m^{e \cdot d} \text{ mod } n = m$  für  $m \in \mathbb{Z}_n$ , wobei der größte gemeinsame Teiler  $ggT(m, n) = p$  oder  $ggT(m, n) = q$  ist.

- Wenn  $ggT(m, n) = p$ , dann folgt daraus  $ggT(m, q) = 1$
- Da  $e \cdot d \text{ mod } \varphi(n) = 1$ , gibt es ein  $k \in \mathbb{N}$ ,  
wobei  $e \cdot d = \varphi(n) \cdot k + 1 = (p - 1)(q - 1) \cdot k + 1$
- Daraus folgt für alle  $x \in \mathbb{Z}_n$ :  

$$x^{(p-1)(q-1) \cdot k + 1} \text{ mod } n = x \iff \begin{aligned} x^{(p-1)(q-1) \cdot k + 1} \text{ mod } p &= x \text{ mod } p \text{ und} \\ x^{(p-1)(q-1) \cdot k + 1} \text{ mod } q &= x \text{ mod } q \end{aligned}$$
- Die erste Gleichung der rechten Seite für  $m = x$  ist erfüllt, da  $ggT(m, q) = 1$  ist. Die zweite Gleichung ist durch  $m^{q-1} \text{ mod } q = 1$  gültig.

- Dadurch lässt sich beweisen:  $m^{\varphi(n) \cdot k + 1} \bmod n = m^{e \cdot d} \bmod n = m$

Um die Sicherheit des Verfahrens zu gewährleisten, müssen  $p$  und  $q$  so gewählt werden, dass es nicht möglich ist, diese aus  $n$  zu berechnen. Deshalb muss insbesondere die Geheimhaltung der beiden Primzahlen sichergestellt werden [11, S. 38]. Das BSI gibt an, dass eine Schlüssellänge für  $n$  von mindestens 2000 Bit zu verwenden ist. Daraus ergeben sich folgende Nebenbedingungen für die Wahl der Zahl  $e$ :

- $\text{ggT}(e, (p-1) \cdot (q-1)) = 1$
- $2^{16} + 1 \leq e \leq 2^{1824} - 1$

Ab dem Jahr 2023 erachtet das BSI eine Schlüssellänge von mindestens 3000 Bit für ausreichend sicher.

Für die Erzeugung von zufälligen Primzahlen sind gemäß dem BSI drei Verfahren zulässig. Eine der Grundlagen ist die sogenannte Verwerfungsmethode, wobei ein Intervall, in dem die gesuchte Primzahl gefunden werden soll, zunächst wie folgt definiert wird:

$$I := [a, b] \cap \mathbb{N} \quad (2.6)$$

Anschließend wird eine ungerade Zahl  $p$  entsprechend der Gleichverteilung auf  $I$  ausgewählt. Falls diese keine Primzahl darstellen sollte, wird erneut bei 1 begonnen, bis die ausgegebene Zahl  $p$  eine Primzahl ist. Besonders bei RSA ist darauf zu achten, dass  $I$  nicht zu klein gewählt wird, um die Anzahl der möglichen Primzahlen zu erhöhen. Abschließend ist zu erwähnen, dass der RSA-Algorithmus auch zur Signatur von Nachrichten verwendet werden kann [11, S. 75ff.].

### 2.3.2 ElGamal-Verfahren

Als Basis des ElGamal-Verfahrens gilt der Diffie-Hellman-Schlüsselaustausch aus dem Jahr 1976. Dieser wird zwar nicht zum Ver- und Entschlüsseln von Nachrichten verwendet, bietet aber eine ausreichend sichere Methode, um geheime Schlüssel über einen unsicheren Kanal auszutauschen. Dabei beruht das Austauschverfahren auf dem Problem des diskreten Logarithmus und stellt somit eine Einwegfunktion dar. Bei diesem Verfahren sind die Primzahl  $p$  und die primitive Wurzel  $g \bmod p$  öffentlich bekannt. Der Schlüsselaustausch läuft anschließend wie folgt ab:

- Alice (A) wählt Zufallszahl  $x_A$ , wobei  $x_A \in \{1, \dots, p-2\}$

- A berechnet:  $y_A = g^{x_A} \bmod p$
- Bob (B) wählt Zufallszahl  $x_B$ , wobei  $x_B \in \{1, \dots, p - 2\}$
- B berechnet  $y_B = g^{x_B} \bmod p$
- $y_A$  wird an B und  $y_B$  wird an A übergeben
- A kann nun den Schlüssel  $k$  berechnen:

$$k = y_B^{x_A} \bmod p$$

- B errechnet den identischen Schlüssel  $k$ :

$$k = y_A^{x_B} \bmod p$$

Demzufolge, dass A und B den identischen Schlüssel berechnen können, kann dieser nun für eine symmetrische Verschlüsselung verwendet werden. Der bedeutende Vorteil dabei ist, dass damit das Schlüsselaustauschproblem von symmetrischen Verfahren gelöst werden kann [6, S. 141f.].

Das Diffie-Hellman-Verfahren dient dem ElGamal-Verschlüsselungs- und Signaturverfahren als Grundlage. Bei der Verschlüsselungsfunktion muss zunächst wie beim RSA-Algorithmus eine Schlüsselerzeugung stattfinden [6, S. 122f.].

- A wählt eine Primzahl  $p$  und eine Basis  $g$
- A wählt den privaten Schlüssel  $d$ , wobei  $d \in \{1, \dots, p - 2\}$
- A berechnet  $e = g^d \bmod p$
- Daraus ergeben sich:  $k_{private} = d$  und  $k_{public} = (p, g, e)$

Mithilfe des öffentlichen Schlüssels ist B nun in der Lage, eine Nachricht  $m$  für A zu verschlüsseln, wobei  $m \in \{1, \dots, p - 1\}$ .

- B wählt eine Zufallszahl  $k \in \{1, \dots, p - 2\}$
- B errechnet  $a = g^k \bmod p$  und  $b = m \cdot y^k \bmod p$
- Anschließend übermittelt B die beiden Zahlen  $a$  und  $b$  an A

A kann anschließend mittels des privaten Schlüssels und den beiden übermittelten Werten  $a$  und  $b$  die Nachricht  $m$  berechnen:

- A errechnet:  $z = (a^d)^{-1} \bmod p = a^{p-1-d} \bmod p$
- Dadurch erhält A:  $m = z \cdot b \bmod p$

Wie beim RSA-Algorithmus spricht auch hier das BSI bei der Wahl der Schlüssellänge eine Empfehlung von 2000 Bit und ab dem Jahr 2023 von 3000 Bit aus [11, S. 29].

### 2.3.3 Operationen auf elliptischen Kurven

Bei der Elliptic Curve Cryptography (ECC) und den darauf anwendbaren asymmetrischen kryptographischen Operationen handelt es sich um eine weitere Alternative zum RSA-Algorithmus oder dem ElGamal-Verfahren. Basis dieses Kryptosystems sind Operationen auf elliptischen Kurven über endliche Körper. Dabei werden Punkte auf der elliptischen Kurve miteinander addiert oder multipliziert [9, S. 83].

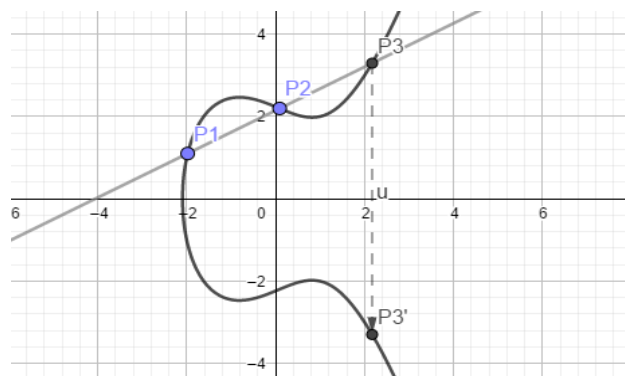
Im Allgemeinen lässt sich eine elliptische Kurve  $E$  über einen Körper  $K$  mit der Gleichung wie folgt darstellen [6, S. 267]:

$$y^2 = x^3 + ax + b \text{ mit } a, b \in K \quad (2.7)$$

Daraus ergibt sich für die Menge aller Punkte auf der Kurve, wobei  $\mathcal{O}$  der Punkt im Unendlichen darstellt:

$$E(K) = \{(x, y) \in K \times K \mid y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\} \quad (2.8)$$

Werden beispielsweise zwei Punkte  $P_1$  und  $P_2$  addiert, ergibt sich ein an der x-Achse gespiegelter Punkt  $P_3'$ . Das Vorgehen einer Addition von  $P_1$  und  $P_2$  auf der elliptischen Kurve  $E: y^2 = x^3 - 2x + 5$  wird nachfolgend in Bild 3 verdeutlicht:



**Bild 3: Beispiel einer elliptischen Kurve**

Liegt lediglich der Punkt  $P_3'$  als Information vor, ist es unmöglich, die beiden ursprünglichen Punkte  $P_1$  und  $P_2$  zu ermitteln. Um aufzuzeigen wie elliptischen Kurven aktuell eingesetzt werden, soll nachfolgend die Kombination aus ElGamal-Verfahren und

ECC genauer erläutert werden. Die Funktionsweise ist dabei wie folgt, wobei jegliche Addition und Multiplikation auf der elliptischen Kurve stattfindet [6, S. 273]:

- Eine elliptische Kurve  $E$  wird festgelegt
- A wählt eine Basis  $g$ , wobei diese nun einen Punkt auf  $E$  darstellt:  $g = (x_g, y_g)$
- A wählt den privaten Schlüssel  $d$  und berechnet:  $e = d \cdot g$
- Somit setzt sich der öffentliche Schlüssel aus der Kurve  $E$  und  $(g, e) = ((x_g, y_g), d \cdot (x_g, y_g)) = ((x_g, y_g), (x_e, y_e))$  zusammen

Möchte B nun eine Nachricht verschlüsseln, wird  $m$  ebenfalls als Punkt auf der Kurve dargestellt:  $m = (x_m, y_m)$ . Zudem wählt B die Zufallszahl  $k$ , die innerhalb der Anzahl der möglichen Punkte liegt. Daraufhin werden  $a$  und  $b$  wie folgt berechnet, um diese anschließend an A zu übermitteln:

- $a = k \cdot g = k \cdot (x_g, y_g) = (x_a, y_a)$
- $b = m + k \cdot e = (x_m, y_m) + k \cdot (x_e, y_e) = (x_b, y_b)$

A kann nun  $m$  aus  $a$  und  $b$  berechnen:

- $z = -d \cdot a = -d \cdot (x_a, y_a) = (x_z, y_z)$
- $m = z + b = (x_z, y_z) + (x_b, y_b) = (x_m, y_m)$

BSI empfiehlt für asymmetrische Verfahren, die auf elliptischen Kurven und dem Diffie-Hellman-Verfahren beruhen, eine Schlüssellänge von mindestens 250 Bit [11, S. 15]. Diese Verfahren werden auch als „Elliptic Curve Diffie-Hellman“ (ECDH) bezeichnet.

Eine weitere Einsatzmöglichkeit der ECC ist der sogenannte „Elliptic Curve Digital Signature Algorithm“ (ECDSA). Dabei werden elliptische Kurven mit dem DSA-Verfahren kombiniert. DSA beruht auf einer Variante des ElGamal-Signaturverfahrens und einer Hashfunktion [6, S. 133]. Beim ECDSA werden zunächst ein privater und ein öffentlicher Schlüssel mittels einer gewählten elliptischen Kurve berechnet. Bei der Erzeugung der Signatur mit dem privaten Schlüssel sowie bei der Prüfung der Signatur, kommt die Hashfunktion zum Einsatz. Somit werden bei dem Verfahren nicht nur der diskrete Logarithmus und elliptische Kurven, sondern als zusätzliche Sicherheit eine Hashfunktion angewendet [6, S. 275f.]. Das BSI geht auch hierbei davon aus, dass eine Schlüssellänge von 250 Bit ausreichend sicher ist [11, S. 15].

Demzufolge bieten die Verfahren basierend auf der ECC einen besonderen Vorteil gegenüber den bereits erwähnten Verschlüsselungsverfahren. Sie benötigen eine

deutlich kleinere Schlüssellänge und erzielen dadurch eine schnellere Verarbeitung und Kommunikation, sowie einen geringeren Speicherbedarf. Deshalb werden elliptische Kurven nicht nur zur Ver- und Entschlüsselung, sondern auch zum Schlüsselaustausch durch Kombination mit dem Diffie-Hellman-Verfahren oder zur Signaturerstellung verwendet.

### 2.3.4 Vergleich der Verfahren

Wie in den vorangegangenen Kapiteln erläutert, weisen die unterschiedlichen Verfahren jeweils spezifische Vor- und Nachteile auf. Dabei unterscheiden sie sich nicht nur in ihrer Komplexität, sondern auch in den verwendeten Schlüssellängen, die einerseits für Sicherheit aber auch zu Performanceeinbußen bei der Berechnung führen können. Um abschließend ausgewählte Verfahren miteinander zu vergleichen, wird wie bei den symmetrischen Algorithmen die Tabelle 2 erstellt, die um das Kriterium „Performance“ erweitert wurde [12]:

Tabelle 2: Vergleich von asymmetrischen Verfahren

Asymmetrische Verfahren	DSA (Signatur)	RSA	ECC (nach NIST)	ECDH
Schlüssellänge (Bit)	1024	1024 / 2048 / 4096 / 8192	192 / 244 / 256 / 384 / 521	256
Aufwand zum Brechen (in Rechenschritten)	$2^{61}$	$2^{70,4} / 2^{94,6} / 2^{126,3} / 2^{167,8}$	$2^{96} / 2^{112} / 2^{128} / 2^{192} / 2^{260,5}$	$2^{128}$
Sicherheit	niedrig	niedrig / mittel / hoch / sehr hoch	mittel / hoch / sehr hoch / maximal	hoch
Performance	mittel	niedrig	hoch	hoch

Anhand dessen ist erkennbar, dass DSA nach Möglichkeit nicht mehr zur Signaturerstellung verwendet werden sollte. Sichere Alternativen, um Nachrichten zu signieren, werden im folgenden Kapitel detaillierter anhand von Hashfunktionen aufgezeigt. Am geeignetsten erscheinen beim Einsatz eines Public-Key-Verfahrens die

Algorithmen basierend auf elliptischen Kurven. Dennoch ist es zum aktuellen Zeitpunkt laut dem BSI ausreichend sicher, weiterhin das RSA-Verfahren zu verwenden. Hierbei sind besonders die hybriden Verschlüsselungsverfahren zu nennen. Wird lediglich der Schlüssel asymmetrisch mit langsamer Performance beispielsweise mit RSA ausgetauscht, kann damit eine performante symmetrische Verschlüsselung für den Nachrichtenaustausch ausgeführt werden. Dadurch kann das Schlüsselaustauschproblem der symmetrischen Verschlüsselung als auch das Performanceproblem der asymmetrischen Verschlüsselung gelöst werden [9, S. 84].

Die Kombination aus dem ElGamal-Verfahren und dem symmetrischen Verfahren AES wird heutzutage in Versionen des Pretty-Good-Privacy-Programms (PGP-Programm) verwendet. Dabei wird das ElGamal-Verfahren nur zum sicheren Austausch des Sitzungsschlüssels genutzt, um anschließend mit diesem Schlüssel eine Kommunikation gesichert durch AES durchzuführen. PGP wird häufig im Bereich der E-Mail-Kommunikation verwendet und kann sowohl zur Verschlüsselung als auch zur Signaturerstellung dienen. Zu Beginn wurde statt dem ElGamal-Verfahren, das RSA-Verfahren als asymmetrisches Kryptosystem eingesetzt. In den aktuellen Versionen ist es möglich, eigene Kombinationen zu erstellen. So kann der Benutzer für die asymmetrische Verschlüsselung zwischen RSA oder ElGamal wählen, bei der symmetrischen unter anderem zwischen AES oder 3DES [9, S. 331f.].

## 2.4 Hashfunktionen

Wie bereits in den vorherigen Kapiteln erläutert, können Public-Key-Verfahren auch zur Signaturerzeugung verwendet werden. Wird jedoch beispielsweise die Erzeugung einer Signatur mittels des RSA-Algorithmus genauer betrachtet, weist die Signatur die gleiche Länge wie der Klartext auf. Diese kann unter Umständen eine enorme Größe erreichen und beansprucht dementsprechend zur Erstellung eine hohe Laufzeit. Ein weiterer Nachteil stellt bei RSA die Verschlüsselung in Blöcken dar, da bei Entfernen eines Blocks der Signatur dennoch ein logischer Klartext entstehen kann. Aus diesem Grund sollen nachfolgend Hashfunktionen und deren Vorteile bei der Signaturerstellung aufgezeigt werden [6, S. 93].

Eine Hashfunktion  $H$  erzeugt aus einer beliebig langen Nachricht  $m$  einen Hashwert  $h$  fester Größe und somit eine eindeutige Prüfsumme für diese Nachricht. Dabei müssen

die Hashfunktionen folgende Anforderungen erfüllen, um als kryptographisch stark zu gelten [9, S. 87f.]:

- Die Hashfunktion muss eine öffentlich bekannte Einwegfunktion sein, um beispielweise von allen Systemen genutzt werden zu können und damit Experten die Sicherheit prüfen können.
- Wird ein Hashwert für eine Nachricht gebildet, muss dieser bei jeder weiteren Berechnung eindeutig dieser Nachricht angehören. Das bedeutet, dass  $H(m) = h_m$  ein Fingerabdruck von  $m$  darstellen muss.
- Es muss sichergestellt werden, dass die Nachricht  $m$  nicht aus dem Hashwert  $h_m$  ermittelt werden kann. Demnach gilt dies als Einwegfunktion und es existiert keine Umkehrfunktion  $H^{-1}(h_m)$ , um den Inhalt von  $m$  zu erhalten.
- Die wichtigste Eigenschaft ist die Kollisionsresistenz. Das bedeutet, dass zu zwei verschiedenen Nachrichten kein identischer Hashwert  $h$  existieren darf. Dabei spielt die feste Länge des Hashwerts eine bedeutende Rolle. Bei zu kurz gewählten Bitlängen ist es einfacher, Kollisionen zu entdecken. Die Empfehlung des BSI lautet daher, eine Mindestlänge von 256 Bit zu verwenden [11, S. 15].
- Eine weitere Eigenschaft nennt das BSI: die 2nd-Preimage-Eigenschaft. Dabei muss es unmöglich sein, aus einer gegebenen Nachricht  $m$  ein  $m'$  zu ermitteln, das den gleichen Hashwert  $h$  besitzt [11, S. 39].

#### 2.4.1 MD5-Hashfunktion und SHA-Algorithmus

Eine der ersten bekannten Hashfunktionen ist die MD5-Hashfunktion aus dem Jahr 1991 von Ronald Rivest. Dabei handelt es sich um eine Hashfunktion, die Werte mit einer Länge von 128 Bit erzeugt. Heutzutage können Kollisionen bereits in unter einer Minute mit  $2^{18}$  Rechenschritten gefunden werden [6, S. 103].

Eine weitere Hashfunktion ist der sogenannte „Secure Hash Algorithm“ (SHA). Die erste Form ist die Funktion SHA-1, die sicherer gilt als das MD5-Verfahren. Jedoch ist es auch hier bereits möglich, in kurzer Zeit Kollisionen zu ermitteln, da mit einem Hashwert der geringen Bitlänge von 160 Bit gearbeitet wird. Das bereits erläuterte DSA-Verfahren beruht bei der Signaturfunktion auf dieser Hashfunktion, weshalb SHA-1 auch als Schwachstelle des Verfahrens gilt. Als sicherer eingestuft werden die Nachfolger SHA-2 und SHA-3 [6, S. 103]. Aus den SHA-2- und SHA-3-Familien können laut BSI folgende Funktionen bedenkenlos eingesetzt werden, um eine ausreichende Sicherheit zu erhalten [11, S. 39]:



- SHA-2: SHA-256, SHA-512/256, SHA-384 und SHA-512
- SHA-3: SHA3-256, SHA3-384, SHA3-512

Die jeweiligen Suffixe der Funktionen geben die Länge des Hashwerts an.

#### 2.4.2 MAC und HMAC

Trotz des geringen Sicherheitsniveaus spielen die Funktionen MD5 und SHA-1 noch eine bedeutende Rolle im HMAC-Verfahren (Hash-based Message Authentication Code). Die Grundlage dazu ist der sogenannte MAC (Message Authentication Code), der mit einer Einweg-Hashfunktion und einem Schlüssel arbeitet. Die Hashfunktion kann nur mittels des ausgetauschten Schlüssels  $k$  korrekt verwendet werden [9, S. 88f.]. Die durch das BSI vorgegebene Schlüssellänge sollte mindestens 128 Bit betragen [11, S. 15]. Ein MAC wird generell wie folgt dargestellt, wobei  $K$  der Menge aller möglichen Schlüssel entspricht [6, S. 109f.]:

$$\{h_k \mid k \in K\} \quad (2.9)$$

Es müssen zudem weitere Eigenschaften vorliegen:

- Für einen beliebigen Schlüssel  $k$  und beliebiger Nachricht  $m$  muss  $H_k(m)$  und somit der MAC leicht zu berechnen sein.
- Es sollte die Kompressionseigenschaft erfüllt sein, sodass eine beliebig lange Nachricht  $m$  auf die definierte Bitlänge von  $h_k(m)$  abgebildet werden kann.
- Für den beliebigen, öffentlich unbekanntem Schlüssel muss zudem gelten, dass dieser fälschungsresistent ist. Obwohl MAC-Paare ( $m$  und  $h_k(m)$ ) öffentlich bekannt sind, muss es unmöglich sein, weitere Paare dadurch zu ermitteln und somit beispielsweise auf den Schlüssel  $k$  zu schließen.

Die Vorgehensweise bei MAC ist wie folgt:

- Ein Schlüssel  $k$  wird zwischen A und B ausgetauscht.
- A sendet an B die Nachricht  $m$  und den zugehörigen MAC  $h_k(m)$
- B berechnet ebenfalls den MAC und vergleicht diesen mit dem übermittelten MAC. Stimmen die Werte überein, kann B sicher sein, dass die Nachricht von A stammt (Authentizität).

Das daraus entstandene HMAC-Verfahren wird durch die Komplexität von XOR-Operationen auf Blöcken erweitert. Der Vorteil dabei ist, dass die Kollisionsresistenz von

der verwendeten Hashfunktion keine große Rolle spielt, weshalb auch weiterhin HMAC-MD5 und HMAC-SHA1 Verwendung finden.

### 2.4.3 Vergleich der Verfahren

Im Jahr 2014 wurde ein Vergleich der Funktionen vorgenommen, der sich wie in der nachfolgenden Tabelle 3 zusammenfassend darstellen lässt und die zuvor genannten Vor- und Nachteile aufzeigt [12].

**Tabelle 3: Vergleich von Hashfunktionen**

Hashverfahren	MD5	SHA-1	SHA-2-Familie	SHA-3-Familie
Bitlänge h (Bit)	128	160	256 / 384 / 512	256 / 384 / 512
Aufwand zum Brechen (in Rechenschritten)	$2^{18}$	$2^{61}$	$2^{128} / 2^{192} / 2^{256}$	$2^{128} / 2^{192} / 2^{256}$
Sicherheit	extrem niedrig	niedrig	hoch / sehr hoch / maximal	hoch / sehr hoch / maximal

Da wie bereits erwähnt MD5 und SHA-1 nur eine geringe Sicherheit bieten, empfiehlt das BSI daher den Einsatz nur im Zusammenhang mit HMAC oder ähnlichen kryptografischen Verfahren [11, S. 18f.].

Die zuvor genannten Hashfunktionen werden häufig bei der Verschlüsselung von Passwörtern eingesetzt. Jedoch reicht eine einfache Hashwertgenerierung meist nicht aus. Daher gibt es beispielsweise Verfahren, um mittels einer am Passwort angefügten zufälligen Zeichenkette, einem sogenannten Salt, die Hashfunktion komplexer zu gestalten.

Ein Anwendungsbeispiel von Hashfunktionen zur Passwortverschlüsselung ist die sogenannte Funktion „Password-Based Key Derivation Function 2“ (PBKDF2). Dabei handelt es sich um ein genormtes Verfahren, um das Knacken eines Passworts zu erschweren. Im Wesentlichen werden bei der Passwortverschlüsselung mehrere Runden von Hashfunktionen angewendet. In der ersten Runde wird beispielsweise der

Hashwert des Passworts mittels der Hashfunktion SHA-256 berechnet. Anschließend wird der Hashwert wiederum mittels SHA-256 verschlüsselt. Dieses Verfahren wird unter anderem beim verschlüsselten Export von Smartphone-Backups verwendet. Bei komplexeren Funktionen fließt in jeder Runde das ursprüngliche Passwort immer wieder bei der Berechnung mit ein. Ein Brute-Force-Angriff auf das Passwort wird somit deutlich erschwert.

Eine weitere Möglichkeit, um Angriffe auf Passwörter zu verlangsamen, bietet das bcrypt-Verfahren basierend auf den Blowfish-Algorithmus. Hierbei werden ebenfalls mehrere Verschlüsselungsrunden durchgeführt, wobei abwechselnd der Salt und das Passwort in die Berechnung mit einfließen.

Im Jahr 2015 wurde das Argon2-Verfahren Sieger der Password Hashing Competition und stellt eine Methode dar, die bei bestimmten Angriffen als sicherer gilt als PBKDF2 oder bcrypt. Zudem wird von NIST das 2016 entwickelte Balloon-Hashing empfohlen, um Passwörter ausreichend sicher zu verschlüsseln. Beide Verfahren bieten eine hohe Komplexität in der Rundenverarbeitung und basieren auf den Standard-Hashfunktionen wie der SHA-Familie [13].

## 2.5 Klassische Angriffsmöglichkeiten

Bevor Quantencomputer als Bedrohung für die vorher genannten asymmetrischen Kryptosysteme detaillierter erläutert werden, werden klassische Angriffsmethoden betrachtet. Denn auch in der heutigen Zeit ist es bereits möglich, vermeintlich sichere Verschlüsselungsverfahren in angemessener Zeit zu knacken. Wohingegen bei der symmetrischen Verschlüsselung Angriffe mittels Histogramme und Brute-Force-Attacken zu einer Lösung führen können, sind die Angriffe auf das RSA- oder ElGamal-Verfahren deutlich komplexer.

Die Sicherheit des RSA-Verfahren basiert wie zuvor detailliert erläutert auf der Geheimhaltung der beiden Primfaktoren  $d$  und  $g$ , da mit deren Kenntnis der private Schlüssel  $d$  berechnet werden kann. Das Problem der Primfaktorzerlegung, in diesem Fall das Ermitteln von  $p$  und  $g$  aus dem öffentlichen Schlüsselteil  $n$ , kann bei zu klein gewählten Zahlen einfach gelöst werden. Mögliche Vorgehensweisen stellen hierbei die Faktorisierungsmethode nach Fermat, der Quadratische Sieb oder die Pollard's Rho Methode dar. Alle Angriffstechniken stellen eine Bedrohung für das RSA-Verfahren mit zu kurz gewählten Schlüsseln dar.

Die Grundlage des ElGamal-Verfahrens besteht in der Unlösbarkeit des Problems des diskreten Logarithmus. Das Problem beschreibt die Schwierigkeit, den Exponenten und somit den privaten Schlüssel  $d$  trotz Kenntnis der Basis  $g$ , des Moduls  $p$  und des Ergebnisses  $e$  zu ermitteln. Doch auch dafür wurden bereits Lösungen entwickelt, die bei kleinen Schlüsseln in annehmbarer Zeit Ergebnisse liefern. Die bekannteste Methode ist hierbei der Shanks Babystep-Giantstep Algorithmus [14, S. 83ff].

Bei Hashfunktionen spielt die Kollisionsresistenz eine große Rolle. Findet ein Angriff auf eine Hashfunktion statt, werden zufällige Werte mittels der Hashfunktion verschlüsselt. Anschließend werden die Wertepaare in einer Tabelle gespeichert. Ziel eines Angreifers ist es dabei Kollisionen aufzudecken, das heißt, zwei Werte mit identischem Hashwert zu ermitteln. Bei einfach verschlüsselten Passwörtern ohne Salt oder ohne Methoden auf Basis von Verschlüsselungsrunden können Brute-Force- oder Dictionary-Angriffe schnell zu einem Ergebnis führen. Bei Dictionary-Angriffen wird davon ausgegangen, dass das Passwort real existierende Wörter beinhaltet – keine zufällig generierten Zeichenfolgen. Zudem können Lookup- oder Rainbow-Tables von Angreifern ausgenutzt werden. Diese Tabellen beinhalten Wertepaare (Passwörter und die zugehörigen Hashwerte). Diese Tabellen sind aufwendig in der Erstellung, können sehr groß sein und nehmen entsprechend viel Speicherplatz in Anspruch. Dennoch lassen sich Tabellen online frei zugänglich vorfinden, die beispielsweise Standardpasswörter und die zugehörigen Hashwerte beinhalten. Allein diese Tabellen reichen aus, um Systeme mit zu einfach gewählten Passwörtern zu kompromittieren [14, S. 35ff].

### 3 Post-Quantum Kryptographie

Wie in der Einführung dieser Arbeit erläutert, stellen Quantencomputer für die zuvor aufgezeigten klassischen Verfahren eine große Bedrohung dar. Sobald leistungsfähige Quantencomputer in Zukunft eingesetzt werden, könnten bereits theoretisch vorliegende Quantenalgorithmen verwendet werden, um klassische Algorithmen in kurzer Zeit zu knacken. Um ein Verständnis für die Funktionsweise dieser Quantenalgorithmen zu erhalten, werden im nachfolgenden Kapitel zunächst Grundlagen eines Quantencomputers erläutert. Anschließend werden auf deren Basis die Algorithmen dargelegt, die es ermöglichen, den diskreten Logarithmus und die Primfaktorzerlegung in angemessener Zeit zu lösen. Um diesen Bedrohungen entgegenzuwirken, ist es notwendig, Post-Quantum Kryptographie als Alternative zu klassischen Methoden einzusetzen. Ergänzend dazu muss eine Abgrenzung zwischen den Begriffen „Post-Quantum Kryptographie“ und „Quantenkryptographie“ vorgenommen werden. Die Quantenkryptographie beinhaltet Verfahren, die auf Quantencomputern implementiert werden und auf deren Technologie eine Ver- und Entschlüsselung stattfindet. Bei der Post-Quantum Kryptographie handelt es sich um Verfahren, die Angriffe im Zeitalter der Quantencomputer standhalten können und auf klassischen Computern implementiert werden.

Insbesondere gelten die asymmetrischen Verfahren durch Quantencomputer als stark bedroht. Denn es reicht nicht aus, lediglich die Schlüssellänge zu vergrößern, um Quantenalgorithmen standzuhalten. Im Gegensatz dazu ist es bei den symmetrischen Verfahren möglich, die Sicherheit durch Erhöhung der Schlüssellängen zu gewährleisten. Aktuell bestehende theoretische Quantenalgorithmen sind bisher in der Lage, die Sicherheit von symmetrischen Verfahren lediglich um die Hälfte zu reduzieren. Das bedeutet, dass AES mit der Schlüssellänge von 256 Bit im Quantenzeitalter die gleiche Sicherheit aufweist, wie der heutige AES mit 128 Bit Schlüssellänge [15]. Daher werden in den nachfolgenden Kapiteln symmetrische Verfahren nur am Rande behandelt und stattdessen mögliche Alternativen zu den Public-Key-Verfahren detaillierter aufgezeigt.

### 3.1 Grundlagen eines Quantencomputers

Zu Beginn werden die Charakteristika eines Quantencomputers erläutert, um die Funktionsweise und deren Vorteile gegenüber den heutigen klassischen Rechnern zu verstehen. Einer der relevantesten Begriffe in diesem Zusammenhang ist die sogenannte Superposition. Stehen dem Quantencomputer alle möglichen gleichwahrscheinlichen Lösungen zu einem Problem zur Verfügung, so ist dieser durch die Superposition in der Lage, zeitgleich die korrekte Lösung zu ermitteln und falsche auszuschließen. Der klassische Computer hingegen muss alle Lösungen nacheinander testen, um die Aufgabe zu lösen [16, S. 225]. Um diese Superposition zu veranschaulichen, wird häufig der Vergleich mit „Schrödingers Katze“ vorgenommen. Bei diesem Paradoxon befindet sich eine Katze in einer verschlossenen Kiste, in der zu 50%er Wahrscheinlichkeit ein tödliches Gift ausgeträumt wurde oder mit gleicher Wahrscheinlichkeit nichts geschehen ist. Die Entscheidung hängt dabei von dem Zerfall eines Atoms ab. Die Katze kann somit tot oder lebendig sein. Erst nach Öffnen der Kiste wäre der Zustand eindeutig. Der uneindeutige Zustand jedoch entspricht der einer Superposition. Es sind gleichzeitig zwei Zustände der Teilchen möglich. Dabei ist zu beachten, dass diese Superposition nur in der Kiste vorliegt. Wird die Kiste geöffnet und wird der Zustand sozusagen gemessen, tritt ein eindeutiger Zustand ein [17, S. 17f.].

#### 3.1.1 Definition Quantenbit

Die zuvor beschriebene Superposition ist die wichtigste Eigenschaft eines Quantenbits (Qubit). Während klassische Bits lediglich den Wert 0 oder 1 besitzen können, kann ein Qubit beide Werte gleichzeitig aufweisen, die erst beim Messen eindeutig werden [17, S. 20ff.]. Ein Qubit kann somit folgende Zustände annehmen:

$$\alpha \cdot |0\rangle + \beta \cdot |1\rangle \tag{3.1}$$

Dabei stellen die Amplituden  $\alpha$  und  $\beta$  komplexe Zahlen mit nachfolgender Bedingung dar:

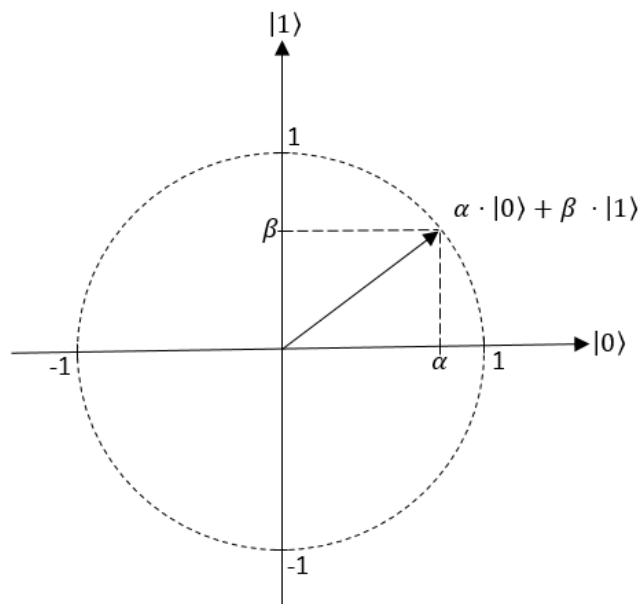
$$|\alpha|^2 + |\beta|^2 = 1$$

Trotz dieser Bedingung sind unendlich viele verschiedene Zustände möglich. Das Messergebnis eines Qubits ist abhängig von den Amplituden, wobei die Superposition zerstört wird. Die Wahrscheinlichkeit, welcher Zustand angenommen wird, ist für den Wert  $|0\rangle$  gleich  $|\alpha|^2$  und für den Wert  $|1\rangle$  gleich  $|\beta|^2$ .

Die Zustände können zudem als Vektoren in einem zweidimensionalen Vektorraum über den komplexen Zahlen abgebildet werden. Dabei ergibt sich folgende Linearkombination, wobei weiterhin die Bedingung  $|\alpha|^2 + |\beta|^2 = 1$  gilt:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \alpha \cdot |0\rangle + \beta \cdot |1\rangle \quad (3.2)$$

Als Vektor dargestellt lässt sich somit Bild 4 ableiten, bei der je nach Kombination von  $\alpha$  und  $\beta$  unendlich viele Punkte auf dem Kreis möglich sind:



**Bild 4: Vektordarstellung einer Superposition [vgl. 15, S. 23]**

### 3.1.2 Definition Quantenregister

Bei einem Quantenregister handelt es sich um eine Folge von Quantenbits. Ein klassisches Register mit beispielsweise zwei Bits besteht daher aus folgenden möglichen Zuständen:  $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ . Das bedeutet, dass ein Register mit  $n$  Bits mittels der Binärdarstellung ganzer Zahlen die Zustände  $|0\rangle, |1\rangle, \dots, |2^{n-1}\rangle$  darstellt. Für den Zustand eines Quantenregisters mit  $n$  Quantenbits ergibt sich somit eine Superposition für die  $2^n$  klassischen Zustände [17, S. 28f.]. Das Quantenregister lässt sich allgemein wie folgt definieren:

$$R = |x_{n-1}\rangle \dots |x_1\rangle |x_0\rangle = |x_{n-1} \dots x_1 x_0\rangle \quad (3.3)$$

Demnach kann das Quantenregister für alle möglichen Zustände in der folgenden Form abgebildet werden:

$$R = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \quad (3.4)$$

Die Bits werden dabei als natürliche Zahl  $i$  dargestellt, wobei für  $|i\rangle$  für  $i = 0, \dots, 2^n-1$  gilt:

$$\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1 \quad (3.5)$$

Bei einer Messung ergibt sich daraus eine Wahrscheinlichkeit von  $|\alpha_i|^2$  für den Zustand  $|i\rangle$ .

### 3.1.3 Quanten-Fouriertransformation

Ein Quantencomputer ist in der Lage, alle möglichen Werte durch Überlagerung beziehungsweise durch die Superposition in einem Register zu speichern. Ein klassischer Rechner hingegen benötigt für die Datenspeicherung entsprechend viele Register. Damit alle möglichen Werte in einem Quantenregister abgebildet werden können, ist die sogenannte Quanten-Fouriertransformation nötig. Dabei handelt es sich um eine Basisoperation auf Quantencomputern. Die diskrete Fouriertransformation (DFT) überführt einen Wertesatz  $x$  auf einen gleich großen Satz von Werten  $y$ . Im Falle der Quantenregister wird die Superpositionsstruktur eines Registers in eine andere Superpositionsstruktur umgewandelt. Die DFT gilt dabei als Grundlage für weitere Quantenalgorithmen, wie beispielsweise der schnellen Fouriertransformation, die zur Bildkompression oder für arithmetischen Operationen verwendet wird. Zudem basieren die für klassische Rechner bedrohlichen Quantenalgorithmen auf der Quanten-Fouriertransformation [15, S. 255]. Diese Algorithmen werden in einem späteren Kapitel detaillierter betrachtet.



### 3.1.4 Deutsch-Jozsa-Algorithmus

Das Problem von Deutsch beschäftigt sich mit der Frage, ob eine Binärfunktion konstant oder balanciert ist. Beim Deutsch-Jozsa-Algorithmus, der von David Deutsch und Richard Jozsa 1992 entwickelt wurde, handelt es sich um eine Lösung des Problems von Deutsch. Zunächst bedeutet das, dass eine Funktion  $f$  vorliegt, die ein Bit zur Eingabe benötigt, um ein Bit auszugeben:

$$f: \{0,1\} \rightarrow \{0,1\} \quad (3.6)$$

Zusätzlich wird ein Baustein zur Verfügung gestellt, um zu einem Bit  $x$  den entsprechenden Wert  $f(x)$  zu ermitteln. Ansonsten stellt die Funktion eine Black Box dar. Es gilt zu prüfen, ob die Funktion konstant, also  $f(0) = f(1)$ , oder balanciert, also  $f(0) \neq f(1)$ , ist. Ein klassischer Computer ist nur durch Ausführen beider Kombinationen in der Lage, eine Aussage über die Funktion zu treffen. Mit dem alleinigen Wissen über beispielsweise  $f(0)$  kann ein klassischer Rechner nicht eindeutig bestimmen, ob die Funktion konstant ist oder nicht. Dazu muss  $f(1)$  berechnet und mit  $f(0)$  verglichen werden. Mittels des Deutsch-Jozsa-Algorithmus ist es für Quantencomputer möglich, dieses Problem zu lösen. Dabei wird mittels eines aus zwei Bit bestehenden Quantenregisters ein Qubit in eine Superposition über die beiden möglichen Eingabebits der Funktion gebracht. Daraus ergibt sich eine Superposition über beide Funktionswerte, welche nur mittels eines Aufrufs ermittelt werden können. Ein Quantencomputer ist demnach in der Lage, schneller eine Aussage über die Funktion zu treffen als klassische Computer [17, S. 33f.].

## 3.2 Quantenalgorithmen als Bedrohung für klassische Kryptographie

Obwohl das BSI für die gängigen klassischen Verschlüsselungsverfahren eine Empfehlung für sichere Schlüssellängen offiziell publiziert, muss bei der Verwendung dieser Algorithmen beachtet werden, dass diese nur gegen heutige Angriffsmethoden eine ausreichende Sicherheit bieten. Der aktuelle Forschungsstand im Bereich der Quantencomputer zeigt jedoch eine rasante Entwicklung. Da angenommen wird, dass in Zukunft leistungsfähige Quantencomputer realisiert werden, wurden bereits vor Jahrzehnten Quantenalgorithmen entwickelt, die die klassischen Probleme der Mathematik, wie der Primfaktorzerlegung und den diskreten Logarithmus, in deutlich kürzerer Zeit lösen können. Dazu bedienen sich die Algorithmen an den zuvor

erläuterten Quanten-spezifischen Grundlagen, wie beispielsweise dem Vorteil der Superposition eines Qubits. Nachfolgend sollen die bedrohlichsten Algorithmen für die heutigen Kryptosysteme detailliert erläutert werden, um die Dringlichkeit des Einsatzes von Quantencomputer-resistenten Verfahren zu verdeutlichen.

### 3.2.1 Grover-Algorithmus

Bei dem Grover-Algorithmus handelt es sich um einen Suchalgorithmus, der von L.K. Grover 1994 entwickelt wurde. Die Grundlage dazu bietet der Deutsch-Jozsa-Algorithmus, bei dem es mittels der Superposition möglich ist, eine korrekte Lösung aus mehreren Lösungen zu ermitteln. Ziel des Algorithmus ist es, in einer unsortierten Liste mit  $n$  Einträgen den gesuchten Wert schnell zu identifizieren. Wobei ein klassischer Rechner einen Aufwand von  $\mathcal{O}(n)$  Rechenschritte benötigt, ist ein Quantencomputer durch die Superposition der Qubits in der Lage, den Rechenaufwand auf  $\mathcal{O}(\sqrt{n})$  zu reduzieren. Diese Rechenschritte werden auch Grover-Iteration genannt [16, S. 280]. Die Bedrohung für die klassischen Algorithmen zeigt sich bei der Suche nach Kollisionen einer Hashfunktion. Durch den Suchalgorithmus von Grover können Kollisionen deutlich schneller ermittelt werden. Dabei ist zu erwähnen, dass das Ergebnis bei einer Suche nach einer korrekten Lösung mittels eines Quantenalgorithmus nur mit einer bestimmten Wahrscheinlichkeit zutrifft. Daher wird die Bedrohung für Hashfunktionen oder symmetrische Verfahren, die auf Hashfunktionen basieren, eher als gering eingestuft. Die Anzahl der Rechenschritte von  $\mathcal{O}(\sqrt{n})$  sind bei einer ausreichend hohen Schlüssellänge weiterhin zu viele, um eine angemessene Lösung darzustellen [16, S. 290].

### 3.2.2 Simon-Algorithmus

Der Simon-Algorithmus ähnelt der Vorgehensweise des Deutsch-Jozsa-Algorithmus. Er wurde 1994 von Daniel Simon entwickelt und ist der erste Quantenalgorithmus, der jedem klassischen deterministischen Verfahren zu dieser Zeit bei der gleichen Problemstellung exponentiell überlegen war [17, S. 223]. Ziel des Algorithmus ist es, die Periode einer Funktion mit den Eingabewerten  $\{0; 1\}$  zu berechnen [15, S. 218f.]. Dabei liegen die Funktion  $f: \{0,1\}^n \rightarrow \{0,1\}^n$  und zwei mögliche Alternativen vor:

- $f$  ist bijektiv und somit haben jegliche Vektoren ein unterschiedliches Bild
- Je zwei Vektoren aus  $f$  haben dasselbe Bild und es existiert ein Vektor  $s \in \{0,1\}^n, s \neq 0$ , wobei  $f(x) = f(x')$ , wenn  $x \oplus s = x'$

Dabei stellt  $s$  die Periode der Funktion  $f$  dar und beschreibt wie zwei Vektoren mit identischem Bild zueinander ermittelt werden können. Deshalb gilt:

$$f(x) = f(x \oplus s) \quad (3.7)$$

Die Suche nach dem Vektor  $s$  spielt besonders beim nachfolgenden Algorithmus eine bedeutende Rolle.

### 3.2.3 Shor-Algorithmus

Der Shor-Algorithmus gilt als eine der größten Bedrohungen für die aktuellen Public-Key-Verfahren. Peter Shor entwickelte den Algorithmus bereits 1994. Dieser besteht aus einem klassischen Anteil und einem Quantenanteil [18, S. 135]. Der Algorithmus bietet eine effiziente Methode, um eine Primzahl in ihre Primfaktoren zu zerlegen.

Der erste klassische Teil des Algorithmus basiert auf der Suche nach dem größten gemeinsamen Teiler mittels des Euklidischen Algorithmus [17, S. 204f.]. Die zu faktorisierende Zahl sei  $n$ . Dazu wird zufällig eine Zahl  $a$  gewählt, die sich in  $\{2, \dots, n - 1\}$  befindet. Anschließend wird der Euklidische Algorithmus angewendet. Falls das Ergebnis  $z$  ungleich 1 ist, wird abgebrochen. Ist  $z$  jedoch 1 wird die Periode  $p$  von  $a^x \bmod n$  ermittelt. Falls die Periode ungerade ist, beginnt der Ablauf erneut. Falls die Periode gerade ist, werden die größten gemeinsamen Teiler von  $a^{\frac{p}{2}} + 1$  und  $n$  und von  $a^{\frac{p}{2}} - 1$  und  $n$  berechnet. Wird dabei kein echter Teiler von  $n$  ausgegeben, wird erneut von vorn mit einer neuen Zufallszahl begonnen. Der Quantenteil des Algorithmus steckt in der Suche nach der Periode  $p$ . Hierbei wird neben der Quanten-Fouriertransformation der Simon-Algorithmus für die effiziente Suche nach dem Vektor bzw. der Periode eingesetzt. Der Shor-Algorithmus weist schließlich die folgende Laufzeit auf, um einen echten Teiler einer ganzen Zahl  $n$  zu ermitteln [17, S. 231]:

$$\mathcal{O}(\log(\log n \cdot (\log n)^3)) = \mathcal{O}((\log n)^4) \quad (3.8)$$

Besonders das Public-Key-Verfahren RSA ist durch diesen Algorithmus stark bedroht. Da RSA auf der Schwierigkeit beruht, eine Primzahl in ihre Primfaktoren zu zerlegen, gehen Experten davon aus, dass ein RSA-Schlüssel mit der Länge von 2048 Bit mittels des Shor-Algorithmus auf einem Quantencomputer mit 4000 Qubits geknackt werden kann. Der Algorithmus wurde bereits 2012 angewendet, um die Zahl 21 zu faktorisieren. In der Publikation des Algorithmus wird zudem die Lösung des diskreten Logarithmus

aufgezeigt, die sich im Wesentlichen der Methode zur Primfaktorzerlegung ähnelt [9, S. 85].

### **3.3 Quantencomputer-resistente Public-Key-Algorithmen**

Die zuvor erläuterten Quantenalgorithmen bedrohen die derzeitig eingesetzten Verschlüsselungsprotokolle zum Nachrichtenaustausch, sowie Software und Hardwaremodule zur Verschlüsselung. Besonders betroffen sind dabei die Public-Key-Algorithmen, weshalb die Entwicklung von Quantencomputer-resistenter Verfahren für klassische Computer von großer Bedeutung ist. Bei Protokollen in Authentisierungsprozessen ist es aktuell ausreichend, Quantencomputer-resistente Algorithmen zum Zeitpunkt der Entwicklung von Quantencomputern mit entsprechender Größe zu verwenden, da das Protokoll lediglich für einen kurzen Zeitraum relevant ist [19]. Bei der Übertragung von kritischen und vertraulichen Informationen ist es jedoch bereits möglich, die Kommunikation aufzuzeichnen, um sie später von Quantencomputern entschlüsseln zu lassen. In diesem Zusammenhang ist der Einsatz von Quantencomputer-resistenten Algorithmen dringend notwendig. Diese Algorithmen lassen sich in fünf Kategorien der Post-Quantum Kryptographie einordnen, die jeweils auf unterschiedlichen kryptographischen Ansätzen basieren. Bei den Kategorien handelt es sich um multivariate, codebasierte, gitterbasierte, hashbasierte und isogeniebasierte Kryptographie. In den nachfolgenden Kapiteln sollen diese Kryptosysteme detailliert dargestellt und abschließend miteinander verglichen werden.

#### **3.3.1 Multivariate Kryptographie**

Bei der multivariaten Kryptographie werden quadratische, nichtlineare Gleichungen mit mehreren Variablen über endliche Körper verwendet. Dies stellt ein Problem dar, das weder von Quantencomputern noch klassischen Computern effizient gelöst werden kann. Im Wesentlichen wird eine Gleichung als öffentlicher Schlüssel zur Verschlüsselung einer Nachricht verwendet. Nach der Übertragung ist es nur mit der Kenntnis des privaten Schlüssels möglich, diese Gleichung zu lösen. Ein Teil des privaten Schlüssels stellt dabei eine Art Hintertür (Trapdoor) des Gleichungssystems dar. Basis ist das sogenannte „Multivariate-Quadratische-Problem“ (MQ-Problem). Dabei wird eine Menge von  $m$  polynomischen Gleichungen  $p$  mit  $n$  möglichen Variablen  $x$  auf

verschiedene Weisen über einen endlichen Körper  $K$  abgebildet. Somit stellt die Funktion  $P: K^n \rightarrow K^m$  eine Einwegfunktion dar und entspricht folgender Abbildung:

$$(x_1, \dots, x_n) \rightarrow (p_1(x_1, \dots, x_n), \dots, p_m(x_1, \dots, x_n)) \quad (3.9)$$

Bei Kenntnis von  $x \in K^n$  lässt sich  $P(x) = y$  berechnen. Liegt hingegen nur  $y \in K^m$  vor, ist es nicht möglich, das Gleichungssystem nach  $x \in K^n$  aufzulösen. Dazu benötigt es die sogenannte Hintertür beziehungsweise den privaten Schlüssel. Jedoch weisen nicht alle quadratischen Gleichungen eine solche Hintertür auf, weshalb Gleichungen gezielt konstruiert werden müssen. Der private Schlüssel setzt sich aus der einfach umkehrbaren Abbildungsfunktion  $\bar{Q}$  und den beiden affinen Abbildungen  $S$  und  $T$  zusammen [20, S. 193ff.].

Im Jahr 1984 wurde der erste multivariate Ansatz für ein Signaturverfahren beschrieben, das auf folgender quadratischer Funktion basiert [20, S. 203]:

$$y = x_1^2 + ax_2^2 \pmod n \quad (3.10)$$

Hierbei stellt  $n$  das RSA-Modul aus zwei Primfaktoren  $n = p \cdot q$  dar. Dieses Verfahren wurde bereits 1987 mittels eines Algorithmus auch ohne Wissen des Moduls  $n$  gebrochen. Jedoch leitete diese quadratische Funktion die Entwicklung von multivariaten Kryptosystemen ein. Anschließend wurden Kryptosysteme basierend auf umkehrbaren linearen Feldern entwickelt, die sich wie folgt definieren:

$$T(x_1, x_2) = (x_1 + g(x_2), x_2) \quad (3.11)$$

Die Variable  $g$  stellt dabei ein Polynom dar. Da dieses System lediglich zwei Variablen und Gleichungen verwendet, erkannten die Entwickler die Schwierigkeit, eine Gleichung zu entwerfen, die einerseits sicher ist und andererseits einen öffentlichen Schlüssel mit annehmbarer Länge aufweist. Daraufhin wurde ein Kryptosystem mit vier Variablen entwickelt, das ebenfalls in kurzer Zeit gelöst werden konnte und die Entwicklung von Verfahren mit mehr Variablen forderte.

Im Jahr 1988 wurde von Tsutomu Matsumoto und Hideki Imai das sogenannte C\*-Signaturverfahren präsentiert. Dies basiert auf dem zuvor beschriebenen MQ-Problem. Neben der Polynomfunktion  $P$  und dem endlichen Körper  $K = \mathbb{F}_q$  gibt es einen zugehörigen Erweiterungskörper  $L = \mathbb{F}_q^n$  über  $K$  mit  $n \in \mathbb{N}$ . Zwischen den beiden besteht

der Isomorphismus  $\phi: L \rightarrow K^n$  und es können somit bijektive Abbildungen vorgenommen werden. Zudem wird die umkehrbare geheime Polynomfunktion  $\bar{Q}: \mathbb{L} \rightarrow \mathbb{L}$  definiert, woraus sich folgende multivariate Polynomfunktion aus der Verkettung der Funktionen ableiten lässt:

$$Q = \phi \circ \bar{Q} \circ \phi^{-1} \quad (3.12)$$

Diese Funktion wird mit den beiden affinen Abbildungsfunktionen  $S$  und  $T$  verschleiert, die Teile des privaten Schlüssels sind.

Matsumoto und Imai gehen nun von  $q = 2$  und der Polynomfunktion  $\bar{Q}: x \rightarrow y = x^{1+q^\alpha}$  aus, wobei  $x \in \mathbb{L}$  und  $\alpha \in [1, \dots, n-1]$ , so dass  $ggT(1+q^\alpha, q^n-1)$  gilt. Zudem existiert zu  $\bar{Q}$  folgende Umkehrfunktion:

$$\bar{Q}^{-1}(x) = x^h \quad (3.13)$$

Dabei gilt  $h \in \{1, \dots, q^n-1\}$  und  $h \cdot (1+q^\alpha) = 1 \pmod{q^n-1}$ . Zusammenfassend lässt sich die Abbildungsfunktion  $\bar{Q}$  als letzten Teil neben  $S$  und  $T$  des privaten Schlüssels wie folgt definieren:

$$\bar{Q}: \begin{array}{l} \mathbb{L} \rightarrow \mathbb{L} \\ x \rightarrow x^{1+q^\alpha} \end{array} \quad (3.14)$$

Demnach gilt für alle  $x \in \mathbb{L}$ :

$$(\bar{Q}(x))^h = x^{(1+q^\alpha) \cdot h} = x \quad (3.15)$$

Das bedeutet, dass  $\bar{Q}$  und  $P$  jeweils bijektiv sind und zu jedem Geheimtext genau ein Klartext existiert [20, S. 204f.].

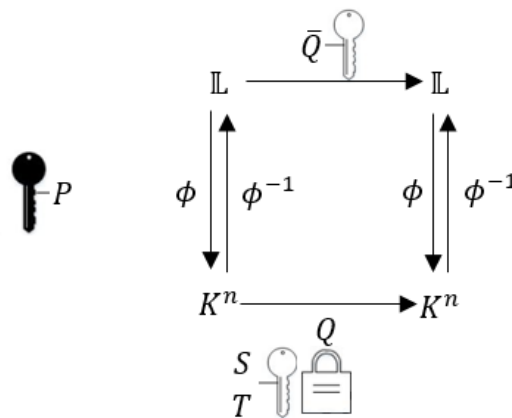
Im Jahr 1995 wurde das Matsumoto-Imai  $C^*$ -Verfahren durch Jacques Patarin gebrochen. Es wurde gezeigt, dass bei ausreichend verfügbaren Geheim- und Klartextpaaren und ohne Wissen von  $S$  und  $T$  eine Relation zwischen  $P(x_1, \dots, x_n)$  und  $(x_1, \dots, x_n)$  ermittelt werden kann [20, S. 215f.]. Das bedeutet, dass  $P$  somit keine Einwegfunktion darstellt und das Verfahren als nicht ausreichend sicher deklariert wurde. Dennoch lieferte das  $C^*$ -Verfahren Jacques Patarin die nötige Grundlage, um das Verfahren „Hidden Field Equations“ (HFE) zu entwerfen. Bei diesem Verfahren wird

die ursprüngliche Abbildungsfunktion  $\bar{Q}$  durch die erweiterte Dembowski-Ostrom Polynomfunktion ersetzt [20, S. 205]:

$$\bar{Q}: x \in \mathbb{L} = \mathbb{F}_q^n \rightarrow y = \sum_{0 \leq i \leq j \leq r} a_{ij} x^{q^i + q^j} + \sum_{0 \leq i \leq r} b_i x^{q^i} + c \quad (3.16)$$

Die Sicherheit stellt der Grad  $d$  der Funktion dar. Umso größer der Parameter, desto sicherer ist die Funktion, aber auch desto geringer ist die Performance. Bis heute ist das HFE-Verfahren mit einem hohen Grad  $d$  ungebrochen. Um die Hintertür in diesem Verfahren zu implementieren, wird der Grad  $d$  auf  $q^i + q^j \leq d, q^i \leq d$  beschränkt, damit eine Umkehrung von  $\bar{Q}$  erzielt werden kann. Bei diesem Verfahren ist  $\bar{Q}$  und somit  $P$  nicht bijektiv und es ist nicht gewährleistet, dass es nur einen Klartext zu einem Geheimtext gibt.

Zusammenfassend lässt sich folgende allgemeine Darstellung der Verfahren in Bild 5 abbilden, wobei sich  $\bar{Q}$  als Hintertüren der Funktionen unterscheiden [20, S. 230]:



**Bild 5: Allgemeine Darstellung der multivariaten Verfahren C\* und HFE**

Multivariate Kryptographie kann sowohl zur Signaturerstellung als auch zur Ver- und Entschlüsselung verwendet werden. Jedoch stellt die Mehrheit der entwickelten Methoden nach Veröffentlichung des HFE-Verfahrens effiziente Lösungen zur Signaturerzeugung dar. Zu nennen sind hierbei die ebenfalls von Patarin entwickelten Verfahren „Unbalanced Oil and Vinegar“ und SFLASH. Hohe Bekanntheit erlangten zudem die Weiterentwicklungen QUARTZ und Rainbow.

### 3.3.2 Codebasierte Kryptographie

Die codebasierte Kryptographie stellt den ältesten Ansatz der Post-Quantum Kryptographie dar. Bereits im Jahr 1978 veröffentlichte Robert McEliece das erste codebasierte Verfahren. Bei dieser Art von Kryptosystem werden bewusst fehlerhafte Codes beim Verschlüsseln einer Nachricht erzeugt, die nur durch den privaten Schlüssel, der einen Fehlerkorrekturalgorithmus darstellt, wieder entschlüsselt werden kann. Ohne Kenntnis über den Korrekturalgorithmus ist es nicht möglich, den fehlerhaften Geheimtext richtig zu dechiffrieren [19].

Im Wesentlichen wird beim McEliece-Verfahren der Goppa-Code zur Fehlerkorrektur verwendet. Dabei handelt es sich um einen binären linearen Code, dessen Eigenschaft es ermöglicht, Fehler in einem Code einfach zu beseitigen. Zunächst wird der Klartext bei der Verschlüsselung mittels einer Generator-Matrix in einen Goppa-Code umgewandelt. Um die Kryptoanalyse zu erschweren, wird dieser Code mit weiteren Matrizen multipliziert. Der öffentliche Schlüssel besteht aus der Generator-Matrix und aus der maximalen Anzahl der einbaubaren Fehler. Die Fehler stellen hierbei eine Invertierung von einzelnen Bits im Code dar. Wie der allgemeine lineare Code wieder in den Goppa-Code überführt werden kann, ist im privaten Schlüssel beschrieben. Abschließend kann der Goppa-Code fehlerkorrigierend und performant entschlüsselt werden [21].

Die Vorgehensweise unterteilt sich beim McEliece-Verfahren in die Schlüsselerzeugung, das Verschlüsseln und das Entschlüsseln. Für eine einheitliche Darstellung werden in den nachfolgenden Kapiteln verwendete Matrizen durch Großbuchstaben, Vektoren fettgedruckt und Skalare unformatiert abgebildet. Das McEliece-Kryptosystem ist demnach wie folgt definiert [22]:

$$McEliece := (m, t) \tag{3.17}$$

Dabei stellt  $m$  die Anzahl der Klartextblöcke und  $t$  die maximale Anzahl der Fehler dar, die der verwendete Goppa-Code  $C$  fehlerfrei korrigieren kann. Damit lassen sich die maximale Geheimtext-Blocklänge  $n$  mit  $n = 2^m$ , die Klartext-Blocklänge  $k$  mit  $k \geq n - m \cdot t$  und die minimale Hamming-Distanz  $d$  mit  $d = 2 \cdot t + 1$  des Goppa-Codes  $C$  berechnen. McEliece verwendete 1978 das einfache, aber sichere Kryptosystem ( $m = 10, t = 50$ ), woraus sich  $(n, k, d) = (1024, 542, 101)$  errechnen lässt.



Für die Schlüsselerzeugung wird zunächst eine  $k \times n$  Generator-Matrix  $G$  für den Goppa-Code  $C$  erstellt, um einen binären Klartext der Bitlänge  $k$  in eine Chiffre der Länge  $n$  umzuwandeln. Daraufhin wird eine zufällige  $k \times k$  Scramble-Matrix  $S$  generiert, die über  $\mathbb{Z}_2$  invertierbar ist. Das bedeutet, die Matrix darf nur Elemente aus  $\mathbb{Z}_2$  besitzen und ihre Determinante muss ungleich 0 sein. Zudem muss eine zufällige  $n \times n$  Permutations-Matrix  $P$  erzeugt werden, die ebenfalls eine binäre Matrix darstellt, die jedoch in jeder Zeile und Spalte genau eine 1 enthalten muss. Schließlich wird die  $k \times n$  Matrix  $G'$  errechnet:

$$G' = S \cdot G \cdot P \quad (3.18)$$

Somit setzt sich der öffentliche Schlüssel wie folgt zusammen:

$$k_{public} = (G', t) \quad (3.19)$$

Der private Schlüssel besteht aus den folgenden Matrizen:

$$k_{private} = (G, S, P) \quad (3.20)$$

Bei der Verschlüsselung wird der Klartext in binäre Blöcke  $m \in \mathbb{Z}_2^k$  der Länge  $k$  geteilt. Diese Klartextblöcke entsprechen binäre Vektoren  $\mathbf{i} = \mathbf{i}_0, \dots, \mathbf{i}_{m-1}$  und werden jeweils mit folgender Funktion chiffriert:

$$E_{k_{public}}(\mathbf{i}, \mathbf{z}) = \mathbf{c} = \mathbf{i} \cdot G' + \mathbf{z} \quad (3.21)$$

Dabei stellt  $\mathbf{z}$  einen beliebigen Vektor mit der Länge  $n$  dar, der ein maximales Gewicht von  $t$  besitzt. Dieser Vektor ist der sogenannte Fehlervektor. Demnach kann  $\mathbf{z}$  nur maximal  $t$  Einsen beinhalten und der Geheimtext nur an maximal  $t$  Stellen invertiert werden.

Bei der Entschlüsselung wird ein  $\mathbf{c}'$  aus dem Geheimtext und aus der Permutationsmatrix berechnet:

$$\mathbf{c}' = \mathbf{c} \cdot P^{-1} \quad (3.22)$$

Daraufhin muss die Dekodierungsfunktion  $decode(\mathbf{c}) = \mathbf{i}'$  des Goppa-Codes  $C$  auf das Ergebnis angewendet werden. Ziel ist es dabei, die Hamming-Distanz auf  $d_H(\mathbf{i}' \cdot G, \mathbf{c}') \leq t$  zu reduzieren.

Abschließend können die jeweiligen Klartextblöcke  $\mathbf{i}$  wie folgt berechnet werden:

$$\mathbf{i} = \mathbf{i}' \cdot S^{-1} \quad (3.23)$$

Zusammenfassend lässt sich die Entschlüsselungsfunktion wie folgt darstellen:

$$D_{k_{private}} = decode(\mathbf{c} \cdot P^{-1}) \cdot S^{-1} \quad (3.24)$$

Ein weiteres bekanntes Verfahren stellt das 1986 entwickelte Niederreiter-Kryptosystem dar, das ebenfalls auf dem Goppa-Code basiert. Anhand dessen wurde aufgezeigt, dass die Goppa-Code-basierten Verfahren nicht nur zur Ver- und Entschlüsselung verwendet werden können, sondern auch in der Lage sind, Signaturen zu erstellen [23, S. 489].

Im Laufe der Zeit wurden weitere fehlerkorrigierende Codes entwickelt, die sich als Grundlage für Public-Key-Kryptosysteme eignen. Dazu zählt der Bose-Chaudhuri-Hocquenghem-Code (BCH-Code), der 1993 anstatt des Goppa-Codes in das McEliece-Verfahren implementiert wurde. Eine Unterart der BCH-Codes ist der Reed-Solomon-Code (RS-Code), der ebenfalls auf binäre Codes basiert, und zwar auf den linearen Blockcodes. Neben der Verwendung in Kryptosystemen finden der BCH-Code und der RS-Code häufig Anwendung in der Telekommunikation. Im Jahr 1991 wurde das Public-Key-Kryptosystem „Gabidulin-Paramonov-Trejtakov“ (GPT) veröffentlicht, das auf Maximum-Rank-Distance-Codes (MRD-Codes) basiert und eine abgewandelte Form des McEliece-Kryptosystems darstellt. In den darauffolgenden Jahren wurden weitere Verfahren auf Basis von MRD-Codes entwickelt, da diese hinsichtlich der Sicherheit und Schlüsselgrößen deutliche Vorteile gegenüber dem klassischen McEliece-Verfahren aufweisen. Durch eine Verallgemeinerung des Goppa-Codes konnte der Algebraic-Geometric-Code (AG-Code) eine Verbesserung der klassischen Methode erzielen. Aufgrund erhöhter Komplexität konnte die Sicherheit und die Fähigkeit der Fehlerkorrektur der Kryptosysteme gesteigert werden. Schließlich gelten sogenannte „Low-Density Parity-Check“-Codes (LDPC-Codes) als mögliche Alternative zum Goppa-Code. Diese wurden bereits 1962 entwickelt und stellen lineare Blockcodes dar. Für die Implementierung empfehlen Experten das Niederreiter-Kryptosystem. Dadurch ergeben

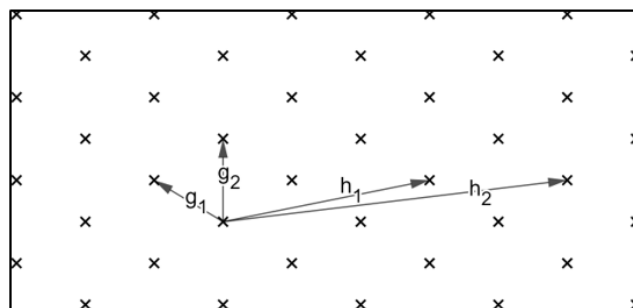
sich Vorteile wie beispielsweise die Reduktion der Menge an privaten Schlüsseln oder die Verbesserung der Fähigkeit zur Fehlerkorrektur [23, S. 491ff.].

### 3.3.3 Gitterbasierte Kryptographie

Die gitterbasierte Kryptographie, auch oft „Lattice-based cryptography“ (LBC) genannt, ist hinsichtlich der Post-Quantum Kryptographie die meist erforschte Kategorie. Darauf basierende Verfahren eignen sich sowohl zur Ver- und Entschlüsselung als auch zur Signaturerstellung oder zum Schlüsselaustausch. Grundlage dieser Verfahren sind sogenannte Gitter, die sich als  $n$  lineare unabhängige Vektoren in einem  $n$ -dimensionalen Raum befinden [24, S. 131]. Das bedeutet, dass ein Gitter  $L$  aus einem Satz von Vektoren  $\mathbf{v}_1, \dots, \mathbf{v}_n \in \mathbb{R}^n$  besteht und sich demnach wie folgt definieren lässt:

$$L(\mathbf{v}_1, \dots, \mathbf{v}_n) := \left\{ \sum_{i=1}^n a_i \mathbf{v}_i \mid a_i \in \mathbb{Z} \right\} \quad (3.25)$$

Die Vektoren  $\mathbf{v}_1, \dots, \mathbf{v}_n$  im Gitter werden auch als Basen des Gitters bezeichnet. Ein Gitter kann unendlich viele Basen besitzen [25, S. 63f.]. Dabei wird zwischen „guten“ und „schlechten“ Basen unterschieden. Eine „gute“ Basis besteht aus Vektoren, die relativ kurz und zueinander annähernd orthogonal sind. Eine „schlechte“ Basis weist Vektoren auf, die eher parallel zueinander sind. Anhand eines 2-dimensionalen Gitters lassen sich die verschiedenen Basen am besten verdeutlichen. Es liegt eine Basis  $B_g$  mit den beiden Vektoren  $\mathbf{g}_1$  und  $\mathbf{g}_2$  und eine Basis  $B_h$  mit den beiden Vektoren  $\mathbf{h}_1$  und  $\mathbf{h}_2$  vor. Bei  $B_g$  handelt es sich um eine „gute“ und bei  $B_h$  um eine „schlechte“ Basis in  $\mathbb{R}^2$  [vgl. 24, S. 131]:



**Bild 6:** Darstellung der Basen  $B_g$  und  $B_h$  in  $\mathbb{R}^2$

Für die Post-Quantum Kryptographie spielen mathematische Probleme auf den beschriebenen Gittern eine große Rolle. Die größten Probleme stellen das „Shortest Vector Problem“ (SVP) und das „Closest Vector Problem“ (CVP) dar [25, S. 64]. Beim SVP sei ein Gitter  $L$  im Vektorraum  $\mathbb{R}^n$  gegeben. Ziel ist es nun, den kürzesten Vektor  $v$  aus  $v_1, \dots, v_n \in L$  zu finden, wobei  $v \neq 0$  gelten soll [24, S. 133]. In einem 2-dimensionalen Raum erscheint dieses Problem trivial. In einem mehrdimensionalen Gitter mit einer hohen Anzahl an Vektoren unterschiedlicher Länge ist dieses Problem jedoch schwer lösbar. Im Jahr 1982 wurde der LLL-Algorithmus von Arjen Lenstra, Hendrik Lenstra und László Lovász entwickelt. Mithilfe des Algorithmus ist es möglich, die Anzahl aller vorhandenen Basen des Gitters auf die kürzesten zu reduzieren. Diese Annäherung ist umso besser, desto kleiner die Dimension  $n$  des Gitters ist. Generell kann mit dem Algorithmus eine  $2^{O(n)}$ -Approximation berechnet werden. Die Herausforderung des CVPs besteht darin, bei einem gegebenen  $n$ -dimensionalen Gitter  $L$  und einem Vektor  $t \in \mathbb{R}^n$ , aber nicht zwingend  $t \in L$ , den nächstgelegenen Vektor  $v \in L$  zu finden [25, S. 64]. Eine mögliche Lösung, um sich dem Problem zu nähern, liefert der sogenannte „Babai’s nearest plane“-Algorithmus, der auf dem LLL-Algorithmus basiert.

Im Jahr 1997 wurde das Kryptosystem Goldreich-Goldwasser-Halevi (GGH) veröffentlicht und basiert auf dem CVP. Es kann sowohl zur Verschlüsselung als auch zur Signaturerstellung verwendet werden. Im Wesentlichen unterteilt sich das Verfahren ebenfalls in die drei Schritte Schlüsselerzeugung, Verschlüsselung und Entschlüsselung. Bei der Schlüsselerzeugung werden zunächst zwei Basen erzeugt. Die erste Basis entspricht einer „guten“ Basis mit fast orthogonalen Vektoren und stellt den ersten Teil des privaten Schlüssels dar. Die zweite Basis ist eine „schlechte“ Basis mit eher parallel verlaufenden Vektoren und wird als öffentlicher Schlüssel herausgegeben. Mittels der beiden Basen kann ein Gitter berechnet werden, das dem zweiten Teil des privaten Schlüssels entspricht. Bei der Verschlüsselung wird die Nachricht als Koeffizient dargestellt und mit der veröffentlichten Basis angewendet. Zusätzlich wird ein zufälliger Fehlervektor addiert. Um das Ergebnis wieder entschlüsseln zu können, wird die „gute“ Basis und der „Babai’s nearest plane“-Algorithmus benötigt. Im Jahre 1999 wurde das Kryptosystem jedoch bereits gebrochen, bietet aber für weitere Entwicklungen die nötige Grundlage. Beispielsweise können Abbildungen auf ein Gitter mittels polynomischer Ringe vorgenommen werden, um die Größe des öffentlichen Schlüssels zu verringern [26, S. 292ff.].

Im Jahr 1996 wurde das Kryptosystem NTRU vorgestellt, das als erstes Kryptosystem auf sogenannten idealen Gittern basiert [25, S. 65f.]. Ideale Gitter stellen eine spezielle Art von Gittern dar, die in ringbasierten Kryptosystemen eine bedeutende Rolle spielen. Besonders bekannt sind die beiden NTRU-Verfahren NTRUEncrypt zur Verschlüsselung und NTRUSign zur Signaturerstellung. Beim NTRU-Algorithmus liegt zunächst ein Ring  $R = \frac{\mathbb{Z}[X]}{(X^N - 1)}$  vor, in dem alle Berechnungen stattfinden [27, S. 2f.]. Aufgrund der Division durch  $(X^N - 1)$  wird sichergestellt, dass bei der Multiplikation zweier Polynome der Grad nie größer als  $N$  sein kann. NTRU basiert neben dem Parameter  $N$  zusätzlich auf den beiden festzulegenden Parametern  $p$  und  $q$ . Bei  $p$  und  $q$  handelt es sich nicht zwangsweise um Primzahlen, jedoch gilt  $ggT(p, q) = 1$  und  $q$  sollte deutlich größer als  $p$  sein. Zudem werden die vier Sätze  $\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_\phi, \mathcal{L}_m$  von Polynomen mit dem Grad  $N - 1$  festgelegt. Alle Elemente  $F \in R$  werden als Polynom oder Vektor in folgender Form dargestellt:

$$F = \sum_{i=0}^{N-1} F_i x^i = [F_0, F_1, \dots, F_{N-1}] \quad (3.26)$$

Bei der Schlüsselerzeugung werden zwei zufällige Polynome  $f, g \in \mathcal{L}_g$  gewählt, wobei für  $f$  die beiden inversen Polynome  $f_p^{-1} \bmod p$  und  $f_q^{-1} \bmod q$  existieren müssen. Anschließend wird  $h$  aus  $h = f_q^{-1} \cdot g \bmod q$  berechnet. Somit setzt sich der öffentliche Schlüssel aus  $h, p, q$  zusammen, wobei  $f$  den privaten Schlüssel darstellt. Bei der Verschlüsselung wird eine Nachricht  $m \in \mathcal{L}_m$  und ein zufälliges Polynom  $\phi \in \mathcal{L}_\phi$  gewählt. Mithilfe des öffentlichen Schlüssels wird die geheime Nachricht  $e$  wie folgt berechnet:

$$e = (p \cdot \phi) \cdot h + m \bmod q \quad (3.27)$$

Zur Entschlüsselung von  $e$  wird zunächst die Hilfsvariable  $a$  wie folgt errechnet:

$$a = f \cdot e \bmod q \quad (3.28)$$

Dabei gilt, dass die Koeffizienten von  $a$  im Intervall von  $-\frac{q}{2}$  bis  $\frac{q}{2}$  liegen. Anschließend kann  $m$  entschlüsselt werden:

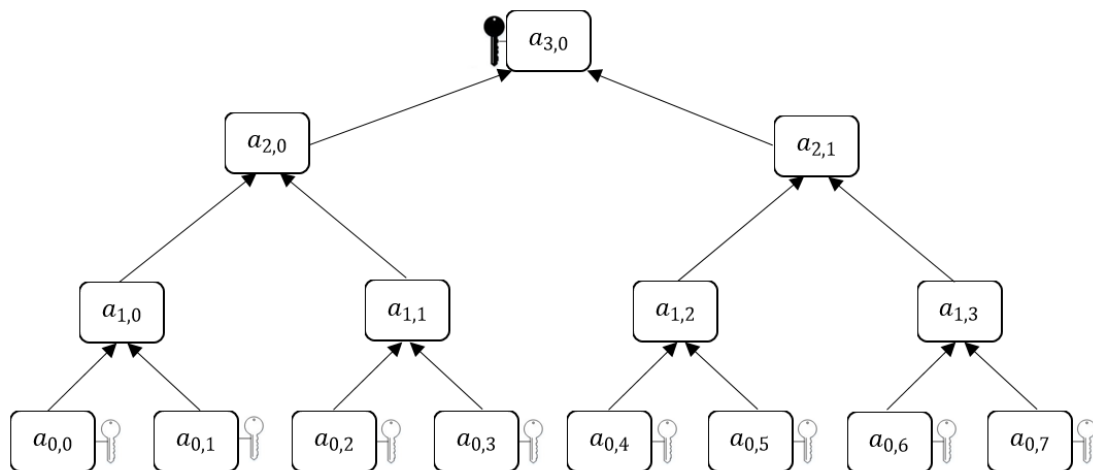
$$m = f_p^{-1} \cdot a \pmod{p} \quad (3.29)$$

Im Jahr 2005 wurde von Oded Regev das sogenannte „Learning with Errors (LWE)“-Problem als ein weiteres Problem auf Gittern vorgestellt. Dieses gilt neben dem SVP und dem CVP als geeignete Alternative für die gitterbasierte Kryptographie. Zum LWE-Problem existieren unterschiedliche Ausprägungen, die auf idealen Gittern basieren. Die ursprüngliche Ausprägung stellt das Such-Problem, auch SLWE-Problem genannt, dar. Die Herausforderung dabei ist es, einen geheimen Vektor in einem gegebenen Ring der rationalen ganzen Zahlen zu finden, wobei ein Satz von annähernden, zufälligen und linearen Gleichungen zu dem Vektor vorliegt. Basierend auf diesem Problem wurden unter anderem von O. Regev verschiedene Public-Key-Kryptosysteme wie beispielsweise der aktuelle FrodoKEM-Algorithmus zum Schlüsselaustausch entwickelt. Eine weitere Ausprägung des LWE-Problems ist das sogenannte Ring-LWE-Problem. Hierbei existiert analog zum NTRU-Verfahren ein Ring  $R = \frac{\mathbb{Z}[X]}{(X^N-1)}$ , in welchem beispielsweise gemäß dem SLWE-Problem ein geheimer Vektor gesucht wird [28, S. 3ff.]. Dieses mathematische Problem wird in den neueren gitterbasierten Kryptosystemen angewendet. Zu nennen ist hierbei der NewHope-Algorithmus, der neben dem FrodoKEM-Algorithmus, als vielversprechendes Schlüsselaustauschverfahren für die Post-Quantum Kryptographie gilt.

### 3.3.4 Hashbasierte Kryptographie

Die hashbasierte Kryptographie gilt im Bereich der Post-Quantum Kryptographie als eine der geeignetsten Methoden, um Signaturen zu erzeugen. Den Signatursystemen liegen hierbei die klassischen Hashverfahren wie SHA oder RSA zugrunde. Wie bereits im Kapitel 2.4 beschrieben, beruht die Sicherheit von Hashfunktionen auf deren Kollisionsresistenz. Es soll demnach nicht möglich sein, für zwei Werte den gleichen Hashwert zu erzeugen. Im Jahr 1979 wurde das Signatursystem „Lamport-Diffie One-Time Signature Scheme“ von den gleichnamigen Entwicklern veröffentlicht. Dieses basiert auf sogenannten „One-Time-Signatures“ (OTS), die mittels Einwegfunktionen genau eine Signatur für eine Nachricht erzeugen können. Damit soll gewährleistet werden, dass keine Rückschlüsse auf den privaten Schlüssel gezogen werden können, wie es bei einer Mehrfachverwendung der Fall wäre. Der Nachteil dieses Verfahrens zeigt sich deutlich, sobald mehrere Nachrichten signiert werden sollen. Die Schlüsselverwaltung ist bei diesem Verfahren mit einem hohen Aufwand verbunden.

Daraufhin entwickelte Ralph Merkle das sogenannte „Merkle Signature Scheme“ (MSS). Mithilfe eines Hash-Baums ist es bei diesem Verfahren möglich, eine Vielzahl von privaten Schlüsseln auf einen öffentlichen Schlüssel abzubilden. Dabei stellen die „Blätter“ die Hashwerte der privaten Schlüssel und die „Wurzel“ des Baums den öffentlichen Schlüssel dar. Das MSS unterteilt sich in Schlüsselerzeugung, Signaturgenerierung und Signaturverifizierung, wobei je nach Baumhöhe  $2^n$  mögliche Nachrichten signiert werden können. Bei der Schlüsselerzeugung werden zunächst mittels eines OTS-Verfahrens für alle  $2^n$  möglichen Nachrichten der jeweilige private Schlüssel  $k_{private}$  und der zugehörige öffentliche Schlüssel  $k_{public}$  erzeugt. Für jeden öffentlichen Schlüssel wird ein Hashwert erzeugt, um den Hash-Baum zu erstellen. Der Baum wird demnach so aufgebaut, dass die inneren Knoten immer den Hashwert der beiden Kindknoten im Baum abbilden. Dabei wird jeder Knoten mit  $a_{i,j}$  beschrieben. Die Baumhöhe entspricht der Variable  $i$ , wobei  $i = 0$  die Stufe der Blätter und  $i = n$  die Wurzel darstellen. Auf den jeweiligen Stufen werden die Knoten von links nach rechts nummeriert, wobei  $a_{i,0}$  der erste Knoten der jeweiligen Stufe ist. So stellt beispielsweise  $a_{1,0}$  den Hashwert der beiden Kindknoten dar, sodass  $a_{1,0} = h(a_{0,0}||a_{0,1})$  gilt. Nachfolgend verdeutlicht Bild 7 die Abbildung eines Merkle-Baums mit der Höhe  $n = 3$  die Struktur, wobei die Blätter des Baums die Hashwerte der privaten Schlüssel sind:



**Bild 7: Aufbau eines Merkle-Baums mit dem öffentlichen und der privaten Schlüssel**

Soll eine Nachricht  $m$  nun signiert werden, wird ein Schlüsselpaar aus dem Baum gewählt und die Signatur  $sig'$  für  $m$  mit dem privaten Schlüssel erstellt. Das bedeutet

gemäß dem Beispiel aus Bild 7, dass  $k_{public} = a_{3,0}$  und  $h(k_{private}) = a_{0,j}$  entspricht, wobei  $0 \leq j < 2^3$  gilt, da die Baumhöhe  $n = 3$  ist. Der Pfad  $A$  vom öffentlichen Schlüssel zum gewählten privaten Schlüssel besitzt  $n + 1$  Knoten. Dabei ist  $A_n = a_{i=n,0}$  die Wurzel und  $A_0 = a_{0,j}$  ein Blatt des Baums. Um den Pfad zu erhalten, müssen für jedes  $A_{i>0}$  die beiden Kindknoten vorliegen. Es ist bekannt, dass  $A_{i=0}$  der Kindknoten von  $A_{i+1}$  ist. Für die Berechnung von  $A_{i+1}$  wird zudem der Nachbarknoten  $auth_i$  von  $A_i$  benötigt, da  $A_{i+1} = h(A_i || auth_i)$  gilt. Damit der Pfad nachvollziehbar ist, müssen alle relevanten Nachbarknoten berechnet werden. Die Signatur setzt sich demnach abhängig vom gewählten privaten Schlüssel wie folgt aus der generierten Signatur  $sig'$  für  $m$  und den errechneten Nachbarknoten jeder Stufe  $auth_0, \dots, auth_{n-1}$  zusammen:

$$sig = (sig' || auth_0 || \dots || auth_{n-1}) \quad (3.30)$$

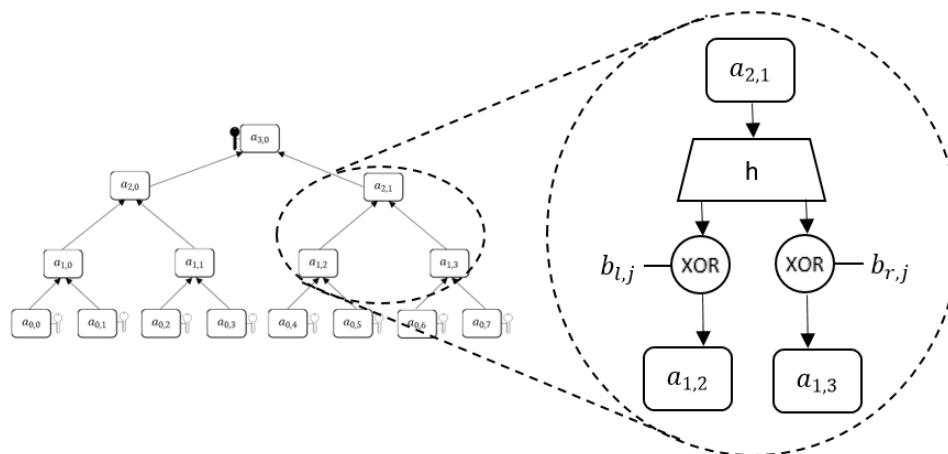
Soll die Signatur beim Erhalten der Nachricht  $m$  nun überprüft werden, wird zunächst der öffentliche Schlüssel für die Prüfung von  $sig'$  verwendet. Ist  $sig'$  eine valide Signatur von  $m$ , werden die mitgegebenen Pfad-Informationen genutzt, um  $A_n$  zu berechnen. Abschließend wird der Hashwert des öffentlichen Schlüssels mit dem aus dem Pfad erhaltenen  $A_n$  verglichen, um die Korrektheit des Schlüssels zu validieren [29, S. 36ff.].

Johannes Buchmann, Erik Dahmen und Andreas Hülsing entwickelten das MSS weiter und veröffentlichten im Jahr 2011 das sogenannte „eXtended Merkle Signature Scheme“ (XMSS). Im Gegensatz zum MSS werden die privaten Schlüssel nur bei Bedarf mittels eines „pseudo-random number generators“ (PRNG) erzeugt. Somit reduziert sich der Speicherplatz enorm, da statt aller möglichen Schlüssel nur noch der öffentliche Schlüssel als Startwert gespeichert werden muss. Bei der Berechnung eines jeden Knotens wird eine Bitmaske  $(b_{l,j} || b_{r,j}) \in \{0,1\}^{2^n}$  mit einer XOR-Verknüpfung verwendet, sodass sich ein Knoten wie folgt errechnen lässt:

$$a_{i,j} = h((a_{i-1,2j} \oplus b_{l,j}) || (a_{i-1,2j+1} \oplus b_{r,j})) \quad (3.31)$$

Graphisch dargestellt lässt sich dies in Bild 8 verdeutlichen:





**Bild 8: Darstellung der XMSS Baumstruktur [vgl. 30, S. 4]**

Beim Aufbau des Baums werden die erzeugten Blätter eines Knotens selbst zu einer Wurzel. Dadurch entsteht ein sogenannter L-Baum. Der öffentliche Schlüssel besteht bei diesem Verfahren nicht nur aus der Wurzel des Baums, sondern zudem aus den verwendeten Bitmasken. Das OTS-Verfahren, das hierbei angewendet wird, ist das sogenannte „Winternitz-OTS“-Schema. Wird eine Signatur erzeugt, beinhaltet diese neben dieser OTS-Signatur und dem Pfad den relevanten Index des verwendeten privaten Schlüssels  $j$  für  $a_{0,j}$ . Bei der Verifikation wird zunächst die OTS-Signatur mit dem öffentlichen Schlüssel überprüft. Anschließend wird mittels der gegebenen Bitmasken und des mitgelieferten Index der Hashwert  $a_{0,j}$  des privaten Schlüssels ermittelt. Anhand des Pfades kann nun die Wurzel berechnet und mit der gegebenen Wurzel aus dem öffentlichen Schlüssel verglichen werden, um diese zu verifizieren [30, S. 1ff.].

Die beiden Verfahren MSS und XMSS sind keine zustandslosen Signaturschemas. Das bedeutet, dass ein privater Schlüssel nur ein einziges Mal verwendet werden darf, um die Sicherheit zu gewährleisten. Daraus folgt, dass bei jeder Signatur der öffentliche Schlüssel modifiziert werden muss. Im Jahr 2014 wurde das erste zustandslose Signaturschema „SPHINCS“ veröffentlicht. Dabei wird das OTS-Verfahren um ein „Few-Time-Signatures“-Schema (FTS-Schema) erweitert, um Schlüsselpaare öfter verwenden zu können. Im Wesentlichen wird ein sogenannter Hyper-Baum aus XMSS-Bäumen erstellt. Zum Aufbau der L-Bäume wird wie zuvor das OTS-Verfahren „Winternitz-OTS“ und für die abschließende Signatur der Blätter wird das FTS-Schema „HORST“ (Hash to Obtain Random Subset Tree) verwendet [31, S. 368ff.]. Varianten

des SPHINCS- sowie des XMSS-Schemas sind für die Post-Quantum Kryptographie von großer Bedeutung und sind zum Teil bereits standardisiert oder befinden sich in einem Standardisierungsprozess.

### 3.3.5 Isogeniebasierte Kryptographie

Bei der isogeniebasierten Kryptographie handelt es sich um das jüngste Gebiet der Post-Quantum Kryptographie. Diese basiert auf mathematischen Problemen im Zusammenhang mit elliptischen Kurven. Besonders die sogenannte supersinguläre isogeniebasierte Kryptographie hat sich für die Post-Quantum Kryptographie durchgesetzt. Die klassische ECC beruht darauf, bei gegebener elliptischer Kurve und den beiden Punkten  $g$  und  $e$  das verwendete Skalar  $d$  zur Berechnung von  $e$  nicht ermitteln zu können (siehe Kapitel 2.3.3). Wohingegen bei der ECC nur auf einer Kurve Operationen durchgeführt werden, stellt das zentrale Problem der isogeniebasierten Kryptographie die Ermittlung der Beziehung oder auch Isogenie zwischen zwei Kurven dar. Eine Isogenie beschreibt in einem endlichen Körper eine Abbildung einer elliptischen Kurve auf eine andere und wird wie folgt definiert:

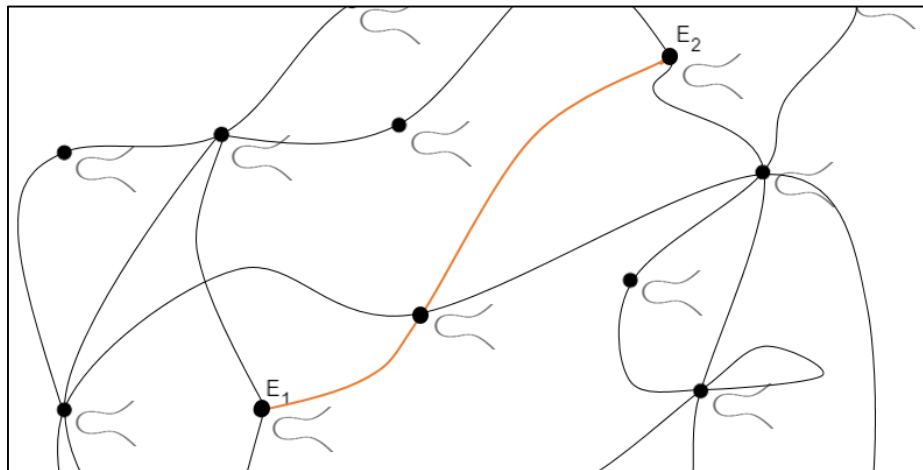
$$\phi: E_1 \rightarrow E_2 \tag{3.32}$$

Dieser Isomorphismus liegt dann vor, wenn  $\phi(\mathcal{O}) = \mathcal{O}$  gilt. Der endliche Körper  $\mathbb{F}_q$  wird mit  $q = p^a$  definiert, wobei  $a$  der Exponent einer hohen Primzahl  $p$  darstellt. Bei supersingulären elliptischen Kurven gilt zudem, dass jede dieser Kurven im Erweiterungskörper  $\overline{\mathbb{F}_p}$  isomorph zu jeder supersingulären Kurve über  $\mathbb{F}_{p^2}$  ist. Eine wichtige Rolle spielen Isomorphismus-Klassen, die sogenannten supersingulären  $j$ -Invarianten der Kurven entsprechen und beispielsweise anhand der elliptischen Kurve der Form  $y^2 = x^3 + ax + b$  wie folgt definiert wird:

$$j = 12^3 \cdot \frac{4a^3}{4a^3 + 27b^2} \tag{3.33}$$

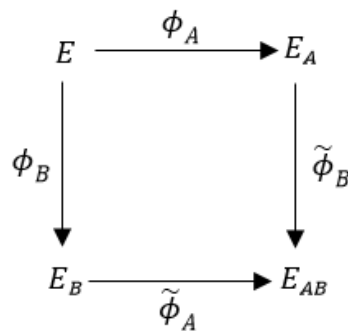
Alle  $j$ -Invarianten müssen in  $\mathbb{F}_{p^2}$  enthalten sein und haben eine ungefähre Anzahl von  $\frac{p}{12}$ , weshalb  $p$  meist sehr groß gewählt wird. Die Schwierigkeit bei isogeniebasierten Kurven besteht darin, die Isogenie  $\phi: E_1 \rightarrow E_2$  bei gegebenen  $j, j' \in \mathbb{F}_q$  zu ermitteln, sodass  $j(E) = j$  und  $j(E') = j'$  gilt. Die  $j$ -Invarianten der jeweiligen elliptischen Kurven stellen hierbei Knoten eines sehr großen Graphen dar. Eine Isogenie beschreibt die

Kanten beziehungsweise den Pfad zwischen zwei Kurven [32, S. 2ff.]. Im folgenden Bild 9 wird anhand eines Beispiel-Graphen die Isogenie zwischen zwei elliptischen Kurven  $E_1$  und  $E_2$  dargestellt, wobei der kürzeste Pfad über einen Knoten gewählt wurde. Das bedeutet, dass der Grad der Isogenie bei zwei Kanten 1 beträgt.



**Bild 9: Abbildung einer Isogenie (Grad 1) zwischen  $E_1$  und  $E_2$**

Im Jahr 2004 wurden die zuvor beschriebenen Grundlagen verwendet, um das sogenannte Schlüsselaustauschverfahren „Supersingular Isogeny Diffie-Hellman“ (SIDH) zu entwickeln. Es wurde erkannt, dass sich die supersingulären elliptischen Kurven für das klassische ECDH-Verfahren als durchaus praktikabel erweisen. Dabei werden die Multiplikation und die Potenzierung von Punkten auf einer Kurve der klassischen Variante durch Isogenieberechnungen ersetzt. Im Wesentlichen läuft der Schlüsselaustausch wie nachfolgend erläutert ab. Zunächst werden ein Graph mit elliptischen Kurven und die Startkurve  $E$  festgelegt. Anschließend wählt A eine zufällige Isogenie  $\phi_A$  und ermittelt somit die Kurve  $E_A$ . B führt diese Schritte analog mit einer zufälligen Isogenie  $\phi_B$  aus und erhält  $E_B$ . Daraufhin werden die beiden Kurven  $E_A$  und  $E_B$  untereinander ausgetauscht. Beide wenden ihre geheimen Isogenien nun auf der erhaltenen Kurve an und können somit die Kurve  $E_{AB}$  berechnen, welche den gemeinsamen Schlüssel entspricht. Zusammenfassend lässt sich das Vorgehen wie folgt darstellen:



**Bild 10:** Allgemeine Darstellung des SIDH-Schlüsselaustauschs [vgl. 33, S. 41]

Der SIDH-Schlüsselaustausch wurde leicht abgewandelt unter dem Namen „Supersingular Isogeny Key Encapsulation“ (SIKE) beim Standardisierungsprozess der NIST als möglicher Kandidat für zukünftige Post-Quantum Kryptographie-Verfahren eingereicht [33, S. 37ff.].

### 3.3.6 Vergleich der Post-Quantum Kryptographie-Kategorien

Bei einer Betrachtung der zuvor beschriebenen Kategorien der Post-Quantum Kryptographie wird deutlich, dass deren Varianten unterschiedliche Vor- und Nachteile aufweisen. Aufgrund dessen ist es nicht möglich, eine Kategorie als die beste für die Post-Quantum Kryptographie zu benennen. Welches Verfahren für welche Einsatzmöglichkeit, wie Signaturerzeugung, Verschlüsselung oder Schlüsselaustausch, geeignet ist, hängt von dessen Eigenschaften ab. So spielen beispielsweise Sicherheit, Schlüssellängen, Signaturlänge oder Performance eine bedeutende Rolle.

Zunächst sollen die Eigenschaften von multivariaten Verfahren genauer betrachtet werden. Obwohl erste Konstruktionen wie das Matsumoto-Imai C\*-Verfahren bereits gebrochen wurden, stellt das MQ-Problem als Grundlage eine ausreichende Sicherheit dar. Beispielsweise ist das HFE-Verfahren mit einem hohen Sicherheitsparameter bis heute ungelöst. Größere Parameter bedeuten jedoch Einbußen in der Performance wie beispielsweise bei der Erzeugung von Schlüsselpaaren. Außerdem sind adäquate Angriffsmethoden entwickelt worden, die an der Sicherheit der multivariaten Kryptographie zweifeln lassen [20, S. 231f.]. Die Algorithmen stellen eine enorme Bedrohung für die multivariaten Verfahren dar. Jedoch sind auch diese noch nicht vollständig ausgereift, um die neusten multivariaten Kryptosysteme zu brechen. Besonders im Vergleich zu den klassischen Verfahren stellen die Schlüssellängen den

größten Nachteil dar [20, S. 197]. Bei RSA-2048 weist der öffentliche Schlüssel eine Größe von 2048 Bit auf, wobei dieser bei einem gängigen Rainbow-Signaturverfahren bereits 22,68 kiloBytes (kB) entspricht. Die Längen der privaten Schlüssel sind ebenfalls ein Vielfaches der privaten Schlüssel der klassischen Verfahren. In der Vergangenheit waren multivariate Verfahren deshalb ungeeignet für den Einsatz auf Smartcards oder anderen kleinen Geräten mit wenig Speicher [20, S. 230]. Des Weiteren konnte ein großer öffentlicher Schlüssel die Performance bei Authentifizierungsprozessen negativ beeinflussen, da der Schlüssel bei jeder Transaktion neu übermittelt werden muss. Ziel bei der Entwicklung von Verfahren ist es daher, die Schlüssellänge zu reduzieren, die Sicherheit aber weiterhin zu gewährleisten. Aufgrund der technologischen Fortschritte stellt der Speicherplatz auch auf Smartcards in der heutigen Zeit kein Problem mehr dar. Dennoch werden multivariate Verfahren besonders in der Signaturerzeugung bevorzugt, da die generierten Signaturen eine vergleichsweise kleine Größe aufweisen.

Bei den codebasierten Verschlüsselungsverfahren spielen die verwendeten fehlerkorrigierende Codes eine große Rolle. Jeder dieser Codes hat in Verbindung mit dem McEliece- oder dem Niederreiter-Kryptosystem jeweilige Vor- und Nachteile. So ist die klassische Variante des McEliece-Kryptosystems mit dem Goppa-Code zwar in ihrer Entschlüsselungsfunktion eine der performantesten Methoden, weist aber eine große Schlüssellänge des öffentlichen Schlüssels auf. Dahingegen besitzen der BCH- und RS-Code bei gleicher Geschwindigkeit eine kleinere Größe der Schlüssel. Jedoch wird die Sicherheit der beiden Codes hinsichtlich des privaten Schlüssels als geringer eingeschätzt. Dennoch gilt für alle der beschriebenen fehlerkorrigierenden Codes, dass eine höhere Fehlerkorrekturfähigkeit zwar eine höhere Sicherheit aufgrund der Komplexität bietet, aber auch zu einer Erhöhung der Schlüssellängen und der Performance führt [23, S. 491ff.]. Im Vergleich zu den klassischen Verfahren weisen bei gängigen Parametern die Schlüssellängen eine Größe von mehreren hundert kiloByte auf. Mittels des sogenannten Gaußschen Eliminationsverfahren ist es möglich, die Schlüsselgröße des öffentlichen Schlüssels zumindest auf etwa 100 kB zu reduzieren. Das klassische McEliece-Verfahren ist seit seiner Entwicklung vor 40 Jahren stets stabil und ungebrochen und gilt daher als gut untersucht und verlässlich. Besonders zum Schlüsselaustausch ist dieses Verfahren zu empfehlen.

Die Sicherheit von gitterbasierten Kryptosystemen steigt mit der Mehrdimensionalität des verwendeten Gitters. Obwohl bereits Annäherungsalgorithmen wie der LLL-Algorithmus existieren, um allgemeine Gitterprobleme zu lösen, sinken deren

Erfolgschancen bei mehrdimensionalen Gittern [24, S. 138f.]. Neben den Annäherungsalgorithmen stellen Brute-Force- oder Meet-in-the-middle-Attacken weitere Sicherheitsrisiken dar. Meet-in-the-middle-Attacken sind eine effizientere Methode als Brute-Force-Angriffe, da beispielsweise beim NTRU-Verfahren nicht alle Polynome der Länge  $N$ , sondern gleichzeitig zwei mögliche Polynome mit Koeffizienten von  $N/2$  getestet werden. Ein großer Vorteil der gitterbasierten Kryptosysteme zeigt sich in der vergleichsweise geringen Schlüssellänge. Bereits im Jahr 1996 zeigten Forschungen zum NTRU-Verfahren die deutlich reduzierten Schlüssellängen bei damaliger ausreichender Sicherheit. Durch die Verwendung von kurzen Schlüsseln ist die Laufzeit des Verfahrens mit der des performanten McEliece-Verfahrens zu vergleichen [27]. Des Weiteren ist das NewHope-Verfahren hinsichtlich des Sicherheitsniveaus und der Laufzeit sehr gut untersucht [28, S. 32f.]. Neben einer hohen Performance punktet NewHope mit einer hohen Speichereffizienz und einer einfachen Implementierung, weshalb es bereits im Browser Google Chrome als Hybridlösung mit einer ECDH-Variante testweise implementiert wurde. Die genannten Vorteile sind teilweise auf die Eigenschaften des Ring-LWE-Problems zurückzuführen. Dabei muss beachtet werden, dass die Kryptoanalyse für dieses Problem am Anfang steht und noch nicht vollständig entwickelt ist.

Die vorgestellten hashbasierten Kryptosysteme werden ausschließlich zur Signaturerstellung verwendet. Die Sicherheit der Post-Quantum Algorithmen basiert hierbei auf der zugrundeliegenden Hashfunktion. Ist diese kollisionsresistent, dann ist das gesamte Kryptosystem sicher. XMSS bietet den großen Vorteil, dass verwendete Hashfunktionen einfach ausgetauscht werden können, sollten diese als nicht mehr sicher gelten. Mittels der entwickelten Merkle-Bäume ist es möglich, ein OTS-Schema nicht nur für ein Dokument anzuwenden, sondern für einen öffentlichen Schlüssel mehrere private Schlüssel zur Signatur zu nutzen. Der Nachteil der ursprünglichen Form MSS zeigt sich bei einer großen Baumstruktur deutlich in der Verwaltung der privaten Schlüssel, der Performance und der Größe der Signatur. Der bedeutende Vorteil der Weiterentwicklung XMSS ist, dass nicht mehr alle möglichen privaten Schlüssel gespeichert werden müssen. Zudem gilt das Signaturschema als vorwärtssicher und kann somit trotz Kompromittierung eines privaten Schlüssels weiter eingesetzt werden [29, S. 35ff.]. Wie bereits erläutert, ist das XMSS jedoch nicht zustandslos und bereits genutzte private Schlüssel können nicht mehr wiederverwendet werden. Bei dem Signaturverfahren SPHINCS ist dies nicht der Fall, da das zugrundeliegende FTS-Schema HORST eine Mehrfachverwendung unterstützt. Dies führt hingegen zu längeren

Signaturen als bei XMSS [30, S. 20] [31, S. 389]. Im Allgemeinen hängt die Performance von der Höhe des Baums und somit von der Anzahl möglicher zu signierender Nachrichten ab. Zur effizienten Berechnung der Pfade im Baum wurden bereits unterschiedliche Algorithmen entwickelt, die die Laufzeit der Signaturerstellung und der Verifikation deutlich verbessern können. Die Laufzeiten des XMSS weisen bei der Schlüssel- und Signaturerzeugung ein schlechteres Ergebnis auf als das SPHINCS Kryptosystem. Eine mögliche Bedrohung für Verfahren, die auf Merkle-Bäumen basieren, ist hier der Such-Algorithmus von Grover. Damit ist es beispielsweise möglich, deutlich schneller Pfade im Merkle-Baum zu identifizieren [31, S. 386].

Bei der isogeniebasierten Kryptographie handelt es sich um die jüngste Kategorie, weshalb diese als noch nicht ausreichend erforscht gilt, obwohl die zugrundeliegenden elliptischen Kurven ein gut untersuchtes Gebiet darstellen. Im Vergleich mit den bisherigen Kryptosystemen werden zwar die geringsten Schlüssellängen und kleine Signaturen erzeugt, jedoch ist die allgemeine Laufzeit die höchste [32, S. 2ff.]. Bei der Betrachtung des bekanntesten isogeniebasierten Kryptosystems, dem SIDH-Schlüsselaustauschverfahren, liegt die Sicherheit in der Wahl von ausreichend großen Parametern. Angriffe auf dieses Verfahren stellen auch hier Meet-in-the-middle-Attacken durch klassische Systeme beziehungsweise der Claw-Finding-Algorithmus dar, der von Quantencomputer angewendet werden kann. Bei den Algorithmen werden ausgehend von der bekannten Startkurve alle möglichen Pfade berechnet. Das gleiche Vorgehen wird mit der ausgetauschten Kurve durchgeführt. Ziel dabei ist es, eine Kollision in Form einer weiteren Kurve zu ermitteln, die von beiden Startkurven aus erreichbar ist. Dadurch kann die geheime Isogenie abgeleitet werden und somit auch der geheime Schlüssel. Bei klassischen Computern entspricht die Komplexität  $\mathcal{O}(p^{1/4})$  und bei Quantencomputern  $\mathcal{O}(p^{1/6})$  [33, S. 44].

Anhand der erläuterten Vor- und Nachteile werden die beschriebenen Kategorien hinsichtlich der definierten Kriterien „Sicherheit“, „Performance“, „Schlüssellängen“ und „Signaturlänge“ bewertet und somit miteinander verglichen. Zudem wird die Eignung der Verfahren für Verschlüsselung, Signaturerstellung oder Schlüsselaustausch in Tabelle 4 dargestellt:

**Tabelle 4: Bewertung und Vergleich der Post-Quantum Kryptographie-Kategorien**

Verfahren	Sicherheit	Performance	Schlüssellängen	Signaturlänge	Besonders geeignet für
<b>Multivariat</b>	0	0	-	+	Signaturerstellung
<b>Codebasiert</b>	+	+	-	0	Verschlüsselung
<b>Gitterbasiert</b>	0	+	+	+	Verschlüsselung + Signaturerstellung
<b>Hashbasiert</b>	+	0	0	0	Signaturerstellung
<b>Isogeniebasiert</b>	-	-	+	+	Schlüsselaustausch

Bei dem Kriterium „Sicherheit“ spielen der Grad der Erforschung und bestehende Angriffsmethoden eine wichtige Rolle. Da es innerhalb der Kategorien zu unterschiedlichen Bewertungen kommen kann, werden diese nachfolgend kurz erläutert, um die allgemeine Bewertung nachvollziehen zu können. Die multivariaten Verfahren können eine hohe Sicherheit aufweisen, dies würde sich aber zu Lasten der Performance auswirken, weshalb diese beiden Kriterien eine neutrale Bewertung erhalten. Bei den codebasierten Verfahren gilt das klassische McEliece-Verfahren auf Basis des Goppa-Codes als am relevantesten und spielt bei der Bewertung eine größere Rolle. Zwar besitzen andere fehlerkorrigierende Codes Vorteile hinsichtlich der Schlüssellängen, weisen aber Sicherheitslücken oder eine hohe Laufzeit auf. Bei den gitterbasierten Verfahren zeugen die bereits eingesetzten Methoden von ausreichender Sicherheit, jedoch steht die Kryptoanalyse besonders bei dem Ring-LWE-Problem noch am Anfang. Die hashbasierten Signaturschemas lassen sich in etwa in zwei unterschiedliche Ansätze unterteilen. Bei dem Kryptosystem SPHINCS ist eine sehr gute Performance erkennbar, jedoch steigt damit die Länge der Signaturen und auch die der Schlüssel. Dahingehend können bei XMSS bei höheren Laufzeiten deutlich kürzere Signatur- und Schlüssellängen nachgewiesen werden, weshalb diese Kriterien insgesamt neutral bewertet werden. Bei den isogeniebasierten Kryptosystemen sind zwar die elliptischen Kurven als Grundlage gut erforscht, dennoch ist die Kategorie noch zu jung, um eine valide Aussage über deren Sicherheit treffen zu können.



## **4 Evaluation ausgewählter Quantencomputer-resistenter Public-Key-Verfahren**

Die zuvor beschriebenen Kategorien sind für unterschiedliche Einsatzmöglichkeiten geeignet. Ziel ist es, für alle möglichen Anwendungsszenarien einen weltweiten Standard zu definieren, um Endanwendern sichere Methoden zur Sicherung ihrer Systeme und Prozesse bereitzustellen. Es hat sich in den vergangenen Jahren etabliert, dass das NIST einen Wettbewerb zur Standardisierung startet. Im Bereich der Quantencomputer-resistenten Verfahren wurde im Jahr 2016 der Auswahlprozess gestartet. Die Vorteile eines Wettbewerbs sind es, dass eingereichte Algorithmen direkt miteinander verglichen werden können und gleichzeitig Kryptoanalyse durch die Wettbewerber stattfindet, um andere Verfahren zu verdrängen. Zudem zielt die Veröffentlichung der Algorithmen darauf ab, dass jeder diese Methoden analysieren und bewerten kann. Somit sollen Schwachstellen aber auch Stärken aufgezeigt werden. Im August 2016 hat das NIST zunächst dazu aufgerufen, Bewertungskriterien und Mindestanforderungen an die gesuchten Algorithmen einzureichen. Anschließend wurde die erste Runde des Auswahlprozesses gestartet, wobei 69 verschiedene Verfahren eingereicht wurden. Im Jahr 2019 begann die zweite Runde mit 26 verbliebenen Algorithmen, die in Tabelle 5 aufgelistet werden. Die Verfahren stellen hauptsächlich Key Encapsulation Mechanismen (KEM) und Signaturerstellungsverfahren dar. Bei KEMs handelt es sich um Methoden, um symmetrische Sitzungsschlüssel zu generieren, deren Parameter zur Berechnung über ein Public-Key-Verfahren sicher übertragen werden. Die dritte Runde ist für Mitte 2020 und das voraussichtliche Ende des Wettbewerbs für 2022 bis 2024 angekündigt [34]. Zu den bereits vorgestellten Kryptographie-Kategorien wurde ein weiterer Algorithmus eingereicht, der einer neuen Kategorie zugeordnet werden kann. Verfahren aus der Kategorie „Zero-Knowledge-Beweis“ werden unter anderem bei Authentifizierungsverfahren angewendet. Dabei muss bewiesen werden, dass der Besitz von Wissen über geheime Informationen vorliegt, ohne diese selbst preiszugeben. Der für die Post-Quantum Kryptographie entwickelte Algorithmus stellt ein Signaturerstellungsverfahren dar. Jedoch konnten sich Verfahren aus dieser Kategorie bisher nicht in der Praxis durchsetzen, weshalb diese nicht näher analysiert werden.

**Tabelle 5: Übersicht der Verfahren der zweiten Runde des NIST-Auswahlprozesses**

Kategorie	Verschlüsselung/Schlüsselaustausch	Signaturerstellung
<b>Multivariat</b>		<ul style="list-style-type: none"> <li>- GeMSS</li> <li>- LUOV</li> <li>- MQDSS</li> <li>- Rainbow</li> </ul>
<b>Codebasiert</b>	<ul style="list-style-type: none"> <li>- BIKE</li> <li>- Klassisches McEliece</li> <li>- HQC</li> <li>- LEDAcrypt</li> <li>- NTS-KEM</li> <li>- ROLLO</li> <li>- RQC</li> </ul>	
<b>Gitterbasiert</b>	<ul style="list-style-type: none"> <li>- CRYSTALS-KYBER</li> <li>- FrodoKEM</li> <li>- LAC</li> <li>- NewHope</li> <li>- NTRU (NTRUEncrypt + NTRU-HRSS-KEM)</li> <li>- NTRU Prime</li> <li>- Round5</li> <li>- SABER</li> <li>- Three Bears</li> </ul>	<ul style="list-style-type: none"> <li>- CRYSTALS-DILITHIUM</li> <li>- FALCON</li> <li>- qTESLA</li> </ul>
<b>Hashbasiert</b>		<ul style="list-style-type: none"> <li>- SPHINCS+</li> </ul>
<b>Isogeniebasiert</b>	<ul style="list-style-type: none"> <li>- SIKE</li> </ul>	
<b>Zero-Knowledge-Beweis</b>		<ul style="list-style-type: none"> <li>- Picnic</li> </ul>

Einige dieser Verfahren wurden durch PQCrypto eingereicht. Dabei handelt es sich um eine Konferenz durchgeführt von Experten zum Thema Post-Quantum Kryptographie. Das durch die EU finanzierte Projekt erarbeitet Algorithmen, Angriffsmethoden und Empfehlungen und veröffentlicht Implementierungen zu den jeweiligen Verfahren. Ebenfalls werden Bibliotheken bereitgestellt, die beispielsweise 77 Kryptosysteme beinhalten und C- und Python-Interfaces anbieten [35]. Des Weiteren unterstützt das BSI den Auswahlprozess der NIST und veröffentlicht entsprechende Studien, wie beispielsweise eine Bewertung von gitterbasierten kryptographischen Verfahren.

Anfang 2020 wurde eine Handlungsempfehlung bezüglich der Migration zur Post-Quantum Kryptographie publiziert. Bei dem Design von neuen Produkten soll demnach Krypto-Agilität eine wichtige Rolle spielen, sodass bei Bedarf verwendete kryptographische Methoden einfacher und schneller ausgetauscht werden können.

Aufgrund der noch jungen Forschung in diesem Bereich, empfiehlt das BSI zunächst hybride Lösungen einzusetzen. Das heißt, dass Quantencomputer-resistente Algorithmen nur in Kombination mit den klassischen Methoden implementiert werden sollten. Das BSI greift zudem dem Auswahlprozess voraus und plant in der kommenden „Technischen Richtlinie zu Algorithmen und Schlüssellängen“ das FrodoKEM-Verfahren und das klassische McEliece-Kryptosystem als Hybridlösungen mit klassischen Verfahren zur Schlüsseleinigung zu empfehlen [36, S. 3ff.]. Dabei wird der Sitzungsschlüssel eines symmetrischen Algorithmus durch ein asymmetrisches Verschlüsselungsverfahren übertragen und kann somit in Netzwerkprotokollen wie SSL (Secure Socket Layer) oder SSH (Secure Shell) genutzt werden. In den nachfolgenden Kapiteln wird mittels definierter Bewertungskriterien eine Evaluation von ausgewählten Public-Key-Verfahren durchgeführt. Ziel dabei ist es, die jeweiligen Algorithmen auf einem klassischen Computer zu implementieren, um die jeweiligen Verfahren miteinander zu vergleichen.

#### 4.1 Rahmenbedingungen der Evaluation

Da sich die Algorithmen aus den verschiedenen Kategorien aufgrund der unterschiedlichen Einsatzmöglichkeiten nicht direkt miteinander vergleichen lassen können, wird die Evaluation in drei grundlegende Bereiche unterteilt. Zum einen werden ausgewählte Algorithmen zur Ver- und Entschlüsselung analysiert. Zum anderen finden Vergleiche in den Bereichen Signaturerstellung und Schlüsselaustausch statt. Bevor mit der Evaluation begonnen wird, werden zunächst grundlegende Rahmenbedingungen festgelegt. Dazu zählen relevante kryptographische und wirtschaftliche Kriterien, nach denen die jeweiligen Verfahren bewertet werden. Bei der Auswahl der Algorithmen stehen die besonderen Eigenschaften im Mittelpunkt. Für jeden Evaluationsbereich werden dementsprechend geeignete Kandidaten aus unterschiedlichen Kategorien gewählt. Abschließend wird die Analyseumgebung definiert, in der die Implementierung und die nachfolgenden Testszenarien durchgeführt werden.

##### 4.1.1 Definition der Bewertungskriterien

Zunächst werden die Bewertungskriterien der Evaluation festgelegt. Aufgrund der spezifischen Anforderungen der Einsatzbereiche ist eine unterschiedliche Gewichtung der Bewertungskriterien notwendig. Während die **Signaturlänge** bei der Signaturerstellung ein ausschlaggebendes Kriterium darstellt, ist sie für die

Verschlüsselung und den Schlüsselaustausch nebensächlich. Ein Hauptkriterium aller Bereiche ist die **Performance** der Schlüsselgenerierung. Bei der Signaturerstellung sind zusätzlich die Laufzeiten zur Signaturerzeugung und der Signaturverifikation relevant. Bei den Verschlüsselungsverfahren sowie den Schlüsselaustauschmethoden ist die jeweilige Performance der Ver- und Entschlüsselungsfunktion bedeutend. Eng mit der Performance verbunden sind oftmals die erforderlichen **Schlüssellängen** der privaten und öffentlichen Schlüssel. Eine Verwendung von großen Schlüsseln bedeutet meist Einbußen bei der Laufzeit der Algorithmen. Für alle Bereiche gelten eine hohe **Sicherheit** und einfache **Implementierbarkeit** der Verfahren als relevante Eigenschaften. Es muss eine hohe Sicherheit gegenüber klassischen Angriffen als auch Bedrohungen ausgehend von Quantencomputern gewährleistet sein. Zudem sollen die Algorithmen beispielsweise einfach in bestehende Protokolle implementiert werden können. Ein öffentlich zugänglicher Code, der in viele unterschiedliche Systeme speicherschonend eingebettet werden kann, bietet große Vorteile bei der Standardisierung.

#### 4.1.2 Auswahl der Quantencomputer-resistenten Algorithmen

Die Auswahl der Algorithmen für die folgende Evaluation richtet sich nach den Verfahren der zweiten Runde des NIST-Auswahlprozesses (vgl. Tabelle 5) und den Empfehlungen des BSI. Zudem werden Verfahren bevorzugt, die in Kapitel 3.3 bereits grundlegend erläutert wurden. Im Bereich der Ver- und Entschlüsselung stellt das auf dem Coppa-Code basierende klassische McEliece-Kryptosystem ein gut untersuchtes Verfahren dar und wird daher im Vergleich zum NTRUEncrypt-Verfahren analysiert. Zur Signaturerstellung werden der hashbasierte Algorithmus SPHINCS+ als einziger Kandidat aus dieser Kategorie im NIST-Auswahlprozess und der multivariate Rainbow-Algorithmus miteinander verglichen. Da das BSI bereits eine Empfehlung für die beiden Verfahren McEliece und FrodoKEM für die Schlüsseleinigung ausspricht, werden diese miteinander verglichen. Zusätzlich soll der bereits testweise von Google implementierte NewHope-Algorithmus und das isogeniebasierte SIKE-Verfahren als Schlüsselaustauschverfahren betrachtet werden. Um die jeweiligen Verfahren gegenüberstellen zu können, werden für die Testszenarien ähnliche Schlüsselgrößen gewählt. In Tabelle 6 wird eine Übersicht über die Algorithmen pro Bereich für die Evaluation gegeben:

**Tabelle 6: Auswahl der Algorithmen für die Evaluation**

Kategorie	Verschlüsselung	Signaturerstellung	Schlüsselaustausch
<b>Multivariat</b>		- Rainbow	
<b>Codebasiert</b>	- klassisches McEliece		- klassisches McEliece
<b>Gitterbasiert</b>	- NTRUEncrypt		- FrodoKEM - NewHope
<b>Hashbasiert</b>		- SPHINCS+	
<b>Isogeniebasiert</b>			- SIKE

#### 4.1.3 Aufbau der Analyseumgebung

Die Implementierung der Algorithmen beschränkt sich auf ein Betriebssystem, um den Rahmen dieser Arbeit nicht zu überschreiten. Da die Analyseumgebung auf einem klassischen und somit durchschnittlichen Rechner basiert, kommen die handelsüblichen Betriebssysteme Microsoft Windows, Linux oder Apple MacOS in Betracht. Die veröffentlichten Code-Sourcen wurden bereits von den Entwicklern auf Linux implementiert. Daher wird für die Evaluation das Betriebssystem die 64-bit Ubuntu-Version 20.04 „Focal Fossa“ gewählt. Es wurde sich für den Einsatz einer virtuellen Umgebung mit der Virtualisierungssoftware VirtualBox der Version 6.1 von Oracle entschieden, um eine isolierte Testumgebung zu erhalten. Der zugeteilte Arbeitsspeicher beläuft sich auf 4 GigaByte (GB). Bei dem zugrundeliegenden Testrechner handelt es sich um einen Desktop-PC mit einem 3.1 Gigahertz (GHz) Dual-Core AMD Phenom II X2 550 Prozessor. Die Algorithmen des NIST-Auswahlprozesses liegen in der Programmiersprache C vor. Da die ursprünglichen Ver- und Entschlüsselungsverfahren nicht im Fokus des Auswahlprozesses stehen, werden diese von der Softwareentwicklungs-Plattform GitHub in der Programmiersprache Python bezogen. In der Testumgebung wird daher die Python-Version 3.8.3 verwendet. Bei der Durchführung der Testszenarien werden parallel laufende Prozesse auf ein Minimum reduziert.

## 4.2 Referenzimplementierungen

Die Implementierung der Algorithmen stellt ein Proof of Concept dar. Es werden daher Varianten der Algorithmen ausgewählt, die aufgrund der Parameterwahl im Vergleich ein gleiches Sicherheitsniveau erzielen. Für die Bewertung des Sicherheitsniveaus werden die vorgegebenen Sicherheitskategorien des NIST und die durchgeführten Sicherheitsanalysen der Entwickler der Verfahren herangezogen, die in den jeweiligen Dokumentationen zu den Algorithmen im Auswahlprozess veröffentlicht wurden. Die Entwickler wurden in den Rahmenbedingungen dazu aufgefordert, eine Bewertung des Sicherheitslevels der eingereichten Algorithmen abzugeben. Beispielsweise sollte eine Aussage über die Sicherheit getroffen werden, wenn einem Angreifer 265 Klartexte oder Signaturen vorliegen. Das NIST stellte den Entwicklern fünf Sicherheitskategorien bereit, um die unterschiedlichen Algorithmen mit den jeweiligen Schlüssellängen einordnen zu können. Die Sicherheitskategorie 5 ist das höchste Sicherheitslevel und entspricht der Sicherheit gegenüber Brute-Force-Angriffen auf AES-256. Die Sicherheitskategorie 1 beschreibt hingegen das Sicherheitslevel eines Brute-Force-Angriffs auf AES-128. Ziele der Kategorisierung sind es unter anderem, die Verfahren vergleichbar zu machen und schnellere Entscheidungen für größere Schlüssellängen treffen zu können [37]. Die unterschiedlichen Bezeichnungen der verwendeten Variablen für beispielsweise Schlüssel basieren auf den verwendeten Notationen in den jeweiligen Verfahrensdokumentationen.

### 4.2.1 Ver- und Entschlüsselungsverfahren

Da beim NIST-Auswahlprozess der Fokus nicht auf reine Public-Key-Encryption-Verfahren (PKE-Verfahren) liegt, wurde bei der Analyse und Implementierung auf frei zugängliche Quellcodes auf GitHub zurückgegriffen. Die beiden nachfolgenden Referenzimplementierungen wurden vom gleichen Entwickler bereitgestellt und verwenden daher die gleiche Notation.

Die Implementierung des McEliece PKE-Verfahrens basiert auf den theoretischen Grundlagen aus Kapitel 3.3.2. Zunächst wird ein Schlüsselpaar mittels der Methode  $gen(M, N, T, PRIV\_KEY\_FILE, PUB\_KEY\_FILE)$  erzeugt [38]. Die Eingangsparameter stellen dabei die Anzahl der Klartextblöcke  $M$ , die Geheimtext-Blocklänge  $N$  und die maximale Anzahl der invertierbaren Fehler  $T$  dar. Zudem müssen die Dateinamen der zu generierenden Schlüssel definiert werden.  $T$  muss wie  $M$  und  $N$  einer positiven Ganzzahl entsprechen, wobei  $T \geq 2$ ,  $M \cdot T < N$  und  $N \leq q = 2^M$  gelten muss [39, S. 6].

Dabei wird unter anderem der Goppa Code mit einem ausgelieferten sogenannten „GoppaCodeGenerator“ anhand der Eingabeparameter erzeugt. Anschließend kann beispielsweise eine beliebige Textdatei mit dem erzeugten öffentlichen Schlüssel verschlüsselt werden. Dazu wird die Methode  $enc(PUB\_KEY\_FILE, FILE)$  verwendet. Um die Datei wieder zu entschlüsseln, wird der erzeugte private Schlüssel für die Methode  $dec(PRIV\_KEY\_FILE, FILE)$  benötigt. Nachfolgend in Bild 11 wird dazu ein Auszug aus der Ausgabe des Aufrufs „mceliece.py -h“ zur Anzeige der Code-Dokumentation aufgezeigt [38].

```
$ ./mceliece.py -h
McEliece v0.1

Usage:
mceliece.py [options] enc PUB_KEY_FILE [FILE]
mceliece.py [options] dec PRIV_KEY_FILE [FILE]
mceliece.py [options] gen M N T PRIV_KEY_FILE PUB_KEY_FILE
mceliece.py (-h | --help)
mceliece.py --version
```

**Bild 11: Auszug aus der Ausgabe des Aufrufs „mceliece.py -h“**

Die Funktionsweise des ausgewählten NTRUEncrypt-Quellcodes entspricht der theoretischen Erläuterung aus Kapitel 3.3.3. Die Schlüsselgenerierung ist dabei von drei Eingabeparametern abhängig und wird gemäß dem Code wie folgt definiert:  $gen(N, P, Q, PRIV\_KEY\_FILE, PUB\_KEY\_FILE)$  [40]. Dabei stellen die Parameter  $N$  und  $P$  jeweils eine Primzahl und  $Q$  eine Zweierpotenz dar, wobei  $P < Q$  und  $ggT(P, Q) = 1$  gilt. Die Dateinamen der jeweiligen Schlüssel müssen ebenfalls festgelegt werden. Der private Schlüssel besteht aus den beiden Polynomen  $f$  und  $f_p$  und der öffentliche Schlüssel aus dem Polynom  $h$ . Daraufhin können die beiden Methoden  $enc(PUB\_KEY\_FILE, FILE)$  und  $dec(PRIV\_KEY\_FILE, FILE)$  zur Ver- und Entschlüsselung einer frei wählbaren Textdatei verwendet werden. In der Dokumentation über den Aufruf „ntru.py -h“ wird die Verwendung der Methoden beschrieben (vgl. Bild 12) [40].

```
$ ./ntru.py -h
NTRU v0.1

Usage:
ntru.py [options] enc PUB_KEY_FILE [FILE]
ntru.py [options] dec PRIV_KEY_FILE [FILE]
ntru.py [options] gen N P Q PRIV_KEY_FILE PUB_KEY_FILE
ntru.py (-h | --help)
ntru.py --version
```

**Bild 12:** Auszug aus der Ausgabe des Aufrufs „ntru.py -h“

#### 4.2.2 Signaturerstellungsverfahren

Da es sich bei den zu analysierenden Signaturerstellungsverfahren um Kandidaten des NIST-Auswahlprozesses handelt, werden alle benötigten Quellcodes aus den veröffentlichten Projektpaketen entnommen und auf der Testumgebung implementiert [34].

Das multivariate Verfahren Rainbow zur Signaturerzeugung wurde mit drei möglichen Parametersets beim Auswahlprozess eingereicht. Bei den Referenzimplementierungen werden unterschiedlich große endliche Körper und Hashfunktionen der SHA-2-Familie verwendet. Die Entwickler bieten zudem eine Komprimierungsfunktion an, mit der die Schlüssellänge des Public-Keys reduziert werden kann, die sich aber negativ auf die Performance auswirkt. Der komprimierte Schlüssel muss bei der Signaturverifikation dekomprimiert werden, was die Laufzeit der Verifikation deutlich erhöht. Um die Sicherheit zu erhöhen, werden die Schlüssel und die Signatur mit einem 16 Byte Salt versehen. In Tabelle 7 werden die generierten Schlüssel- und Signaturlängen gerundet in Bytes, die entsprechende Hashfunktion und die Sicherheitskategorie dargestellt [41, S. 29].



**Tabelle 7: Kenngrößen der Rainbow Parametersets (Längen in Bytes)**

Parameterset	Länge Public Key	Länge Private Key	Länge Signatur (inkl. Salt)	Sicherheits- kategorie
<b>Ia (SHA256)</b>	149.000 (compressed 58.100)	93.000	64	1
<b>IIIc (SHA384)</b>	710.600 (compressed 206.700)	511.400	156	3
<b>Vc (SHA512)</b>	1.705.500 (compressed 491.900)	1.227.100	204	5

Anhand der mitgelieferten Parameter werden beim Aufruf der Methode *RainbowKeyGen()* die notwendigen Abbildungsfunktionen des öffentlichen Schlüssels  $pk$  und unter anderem deren invertierte Abbildungsfunktionen als Hintertür beziehungsweise private Schlüssel  $sk$  erzeugt (vgl. Abschnitt 3.3.1). Mit der Methode *RainbowSign( $sk, d$ )* kann die Signatur  $z$  mittels des privaten Schlüssels und des zu signierenden Dokuments  $d$  generiert werden. Dabei wird der Hashwert des Dokuments mithilfe der jeweiligen Hashfunktion gebildet. Um die Signatur zu prüfen, muss ebenfalls der Hashwert des Dokuments erzeugt werden. Anschließend kann mittels des öffentlichen Schlüssels und der erhaltenen Signatur der Hashwert berechnet werden. Stimmen die beiden errechneten Werte überein, ist die Signatur gültig. Die Methode *RainbowVer( $pk, z, d$ )* führt die genannten Schritte durch und liefert einen booleschen Wert zurück [41, S. 9f.].

Das Verfahren SPHINCS+ bietet die Möglichkeit, verschiedenste Hashfunktionen zu implementieren. Die gängigsten wurden als Referenzimplementierung beim NIST-Auswahlprozess eingereicht und basieren auf SHAKE256 aus der von dem NIST standardisierten SHA-3-Familie, SHA-256 und der neu entwickelten Hashfunktion Haraka. Zudem werden Varianten bereitgestellt, die sich in „einfache“ und „robuste“ Instanziierungen der Hashfunktionen in SPHINCS+ aufgliedern. Der wesentliche Unterschied zwischen den beiden Varianten besteht darin, dass bei der einfachen Variante keine Bitmasken verwendet werden. Dies wirkt sich zwar positiv gegenüber der Performance aus, reduziert jedoch deutlich das Sicherheitsniveau, da der Zufallsfaktor nicht mehr gegeben ist. Dadurch bieten die beiden Varianten eine gewisse Flexibilität, wenn beispielsweise der Fokus bei der Implementierung auf einer geringen Laufzeit liegt

und dabei ein höheres Sicherheitsrisiko vertretbar ist. Die empfohlenen Parametersets unterscheiden sich zudem in der festgelegten Länge des erzeugten Hashwerts der gewählten Funktion. Unabhängig von der verwendeten Hashfunktion ergeben sich aufgrund der definierten Hashwertlänge die gleichen Schlüssel- und Signaturlängen. Daher können die Parametersets mit den daraus resultierenden Längen in Bytes wie folgt in Tabelle 8 zusammengefasst werden und gelten für alle Hashfunktionen [42, S. 57]. Aufgrund der Abhängigkeit zwischen Performance und Signaturlänge werden zudem jeweilige Parametersets für kurze Signaturlängen (*s* = small) und für kurze Laufzeiten (*f* = fast) bereitgestellt. Je nach Anwendung kann somit zusätzlich entschieden werden, welches Ziel verfolgt werden soll – kurze Laufzeit oder kurze Signaturen. Die Sicherheitskategorien entsprechen denen der SHAKE256- oder der SHA-256-Hashfunktion, da die Haraka-Hashfunktion aufgrund bestehender Angriffsmöglichkeiten lediglich ein Sicherheitsniveau von 2 erreichen kann [42, S. 38ff.].

**Tabelle 8: Kenngrößen der SPHINCS+ Parametersets (Längen in Bytes)**

Parameterset	Länge Public Key	Länge Private Key	Länge Signatur	Sicherheitskategorie
<b>SPHINCS+-128s</b>	32	64	8.080	1
<b>SPHINCS+-128f</b>	32	64	16.976	1
<b>SPHINCS+-192s</b>	48	96	17.064	3
<b>SPHINCS+-192f</b>	48	96	35.664	3
<b>SPHINCS+-256s</b>	64	128	29.792	5
<b>SPHINCS+-256f</b>	64	128	49.216	5

Um ein Schlüsselpaar für die Signaturerzeugung zu generieren, wird bei SPHINCS+ die Methode *spx\_keygen()* aufgerufen. Damit wird der private Schlüssel berechnet, der aus zwei Teilen besteht. Einerseits beinhaltet dieser *SK.seed*, das für die Erstellung aller privaten Schlüssel benötigt wird, und andererseits *SK.prf* für die Erzeugung der verwendeten Zufallszahlen. Zusätzlich wird der öffentliche Schlüssel gespeichert, der sich aus der Wurzel des Hashbaums *PK.root* und der Zufallszahl *PK.seed* zusammensetzt. Für die Berechnung der Schlüsselteile wird die Zufallsfunktion

$sec\_rand(n)$  mit dem im Parameterset festgelegten Sicherheitsparameter  $n$  ausgeführt. Alle generierten Werte weisen dadurch eine Länge von  $n$  auf. Deshalb lassen sich die Schlüssellängen je nach Parameterset anhand des Parameters  $n$  berechnen. Der öffentliche Schlüssel entspricht somit der Länge  $2n$  und der private Schlüssel der Länge  $4n$ . Die Methode  $spx\_sign(M, SK)$  zur Signaturerzeugung kann anschließend mit der zu signierenden Nachricht  $M$  und dem privaten Schlüssel  $SK$  durchgeführt werden. Die Länge der Signatur ist hier ebenfalls abhängig von den verwendeten Parametern, wie beispielsweise der festgelegten Baumhöhe. Um die Signatur zu überprüfen, wird die Methode  $spx\_verify(M, SIG, PK)$  bereitgestellt. Stimmt die berechnete mit der mitgelieferten Signatur überein, gibt die Methode einen booleschen Wert zurück [42, S. 31ff.].

### 4.2.3 Schlüsselaustauschverfahren

Neben den zuvor beschriebenen Signaturerstellungsvorgängen sind alle zu betrachtenden Schlüsselaustauschmethoden Teil des NIST-Auswahlprozesses. Daher basiert die Implementierung auf der Testumgebung ebenfalls auf den bereitgestellten Programmcodes der Entwickler [34]. Je nach Wahl des Parametersets und der verwendeten Hashfunktion mit festgelegter Hashwertlänge ist die Länge des Session Keys für jede neue Schlüsselgenerierung identisch. Daher können auf Basis der empfohlenen Parameter die jeweiligen Größen direkt abgeleitet werden. Zu jedem Parameterset wird eine Referenzimplementierung geliefert. Alle nachfolgenden Methoden der KEM-Verfahren basieren auf den entsprechenden Methoden des ursprünglichen PKE-Verfahrens, deren Grundlagen bereits im Kapitel 3.3 erläutert wurden. Die PKE-Methoden werden dazu verwendet, um einen Geheimtext auszutauschen, der auf beiden Seiten zur Berechnung des gleichen Session Keys nötig ist. Dabei ist der öffentliche Schlüssel immer Bestandteil dieser Berechnung, weshalb er zusätzlich einen Teil des privaten Schlüssels darstellt und in den folgenden Schlüssellängen der privaten Schlüssel bereits inkludiert ist. In Bild 13 wird der wesentliche Ablauf eines Schlüsselaustausches dargestellt, wobei sich die Methoden der jeweiligen Algorithmen im Detail unterscheiden und in den nachfolgenden Abschnitten erläutert werden.

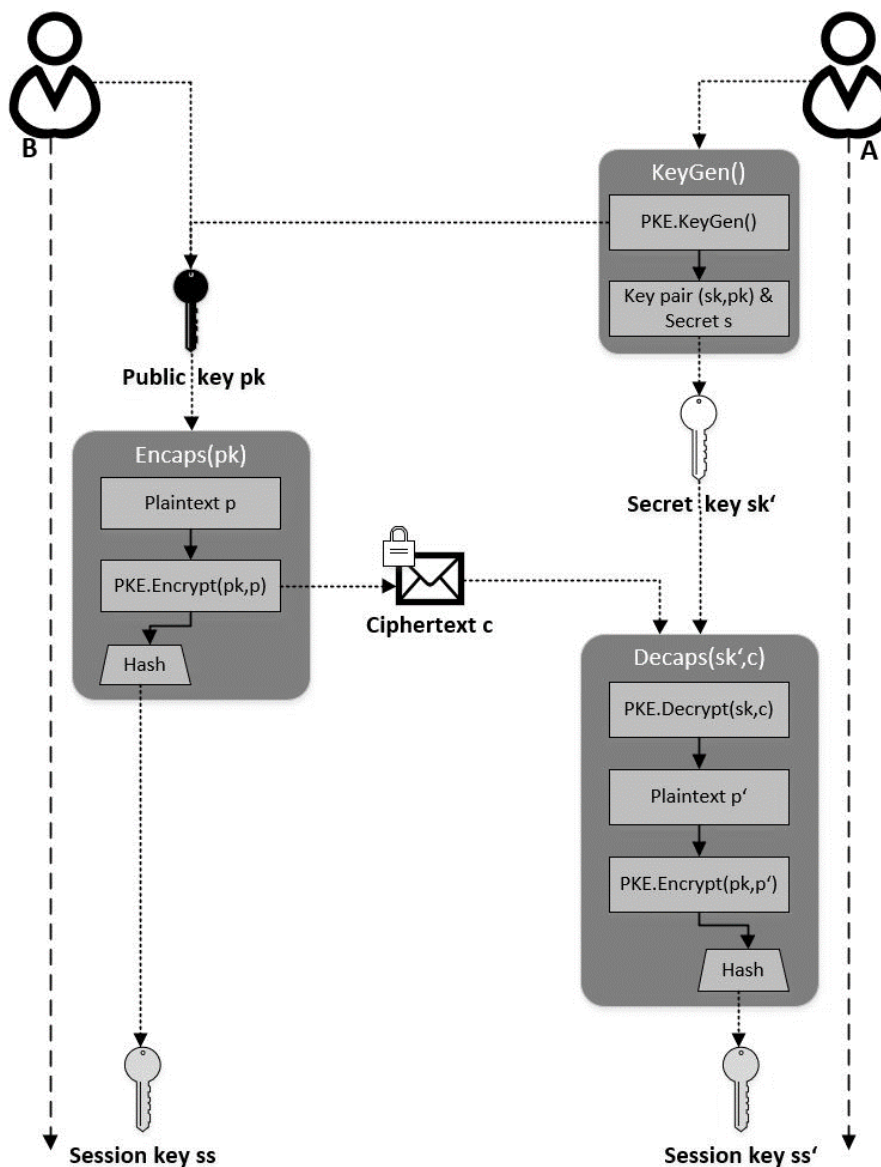


Bild 13: Genereller Ablauf eines KEM

Wird McEliece als Schlüsseleinigungsverfahren angewendet, werden von den Entwicklern fünf grundlegende Parametersets empfohlen, die unterschiedliche Schlüssellängen und Sicherheitslevel aufweisen. Für alle Implementierungen wird bei der Schlüsseleinigung abschließend die Hashfunktion SHAKE256 mit gleicher Hashwertlänge verwendet. Daher weist der zufällig erzeugte Sitzungsschlüssel bei jedem Parameterset eine einheitliche Länge von 32 Byte auf. Nachfolgend in Tabelle 9 werden für die gängigsten Parametersets die zugehörigen Längen der Schlüssel und der Geheimtexte in Bytes, sowie die jeweiligen Sicherheitskategorien aufgezeigt [39, S. 23]:

Tabelle 9: Kenngrößen der McEliece KEM Parametersets (Längen in Bytes)

Parameterset	Länge Public Key	Länge Private Key	Länge Geheimtext	Länge Session Key	Sicherheitskategorie
<b>McEliece348864</b>	261.120	267.572	128	32	1
<b>McEliece460896</b>	524.160	537.728	188	32	3
<b>McEliece6960119</b>	1.044.992	1.058.884	240	32	5
<b>McEliece6688128</b>	1.047.319	1.061.227	226	32	5
<b>McEliece8192128</b>	1.357.824	1.371.904	240	32	5

Bevor der Sitzungsschlüssel generiert wird, muss die Methode *KeyGen()* ausgeführt werden, um den öffentlichen und privaten Schlüssel zu erzeugen. Dabei wird zunächst die Schlüsselgenerierung des McEliece PKE-Verfahrens angewendet und zudem ein zufälliger Klartext  $s$  erzeugt. Der private Schlüssel setzt sich demnach aus dem berechneten öffentlichen und dem privaten Schlüssel, sowie dem Klartext  $s$  zusammen. Anschließend wird der Sitzungsschlüssel auf Seite des Senders mittels der Methode *Encaps(K)* erzeugt. Dabei wird ein zufälliger Klartext  $p$  mit dem öffentlichen Schlüssel  $K$  über die Methode *PKE.Encrypt(p, K)* zu einem Geheimtext  $C$  chiffriert. Woraufhin ein Hashwert aus dem Klartext und dem Geheimtext erzeugt wird. Der Hashwert entspricht dabei dem Sitzungsschlüssel. Daraufhin wird die Methode *Decaps(C, k)* aufgerufen. Der übertragene Geheimtext wird mithilfe des privaten Schlüssels  $k$  über die Methode *PKE.Decrypt(C, k)* entschlüsselt und der Klartext  $p'$  ermittelt. So können ebenfalls mit der Methode *Encrypt(p', K)* der Hashwert und somit der Sitzungsschlüssel aus dem öffentlichen Schlüssel und dem entschlüsselten Klartext  $p'$  berechnet werden [39, S. 8ff.].

Die Entwickler des FrodoKEM-Verfahrens empfehlen drei grundlegende Parametersets, die sich anhand der Gitterdimensionen und somit des Sicherheitsniveaus unterscheiden. Zudem können entweder Hashfunktionen der AES- oder der SHAKE-Familie angewendet werden. Dies bietet höhere Flexibilität bei der Implementierung. Die Dimensionen weisen eine Größe von 640, 976 und 1344 auf. Dementsprechend können die jeweiligen Schlüssellängen errechnet werden, die neben der Geheimtextlänge und der Sicherheitskategorie in Tabelle 10 aufgeführt werden [43, S. 24].

Tabelle 10: Kenngrößen der FrodoKEM Parametersets (Längen in Bytes)

Parameterset	Länge Public Key	Länge Private Key	Länge Geheimtext	Länge Session Key	Sicherheitskategorie
<b>Frodo-640</b>	9.616	19.888	9.720	16	1
<b>Frodo-976</b>	15.632	31.296	15.744	24	3
<b>Frodo-1344</b>	21.520	43.088	21.632	32	5

Analog dem McEliece KEM-Verfahren werden bei FrodoKEM zunächst die Schlüsselpaare mithilfe der Generierungsmethode  $PKE.KeyGen()$  erzeugt. Im Gegensatz zum McEliece-Verfahren wird neben einer Zufallszahl  $s$  zusätzlich der Hashwert  $pkh$  des öffentlichen Schlüssels  $pk$  mit einer Hashfunktion berechnet. Mittels des Hashwerts, der Zufallszahl, der zuvor erhaltenen privaten und öffentlichen Schlüssel wird der private Schlüssel  $sk'$  der KEM-Funktion gebildet ( $sk' = sk, s, pk, pkh$ ). Somit kann der Sitzungsschlüssel mit der Methode  $Encaps(pk)$  ermittelt werden. Zunächst wird ein zufälliger Klartext gewählt, um anschließend die zwei Hashwerte  $r$  und  $k$  und somit zufällige Bitzahlen aus dem Hashwert des öffentlichen Schlüssels und dem Klartext zu generieren. Daraufhin wird die Verschlüsselungsmethode mit dem Klartext, dem öffentlichen Schlüssel und der Zufallszahl  $r$  ausgeführt, um den Geheimtext  $c$  zu erhalten. Der Sitzungsschlüssel  $ss$  bildet den Hashwert aus der Hashfunktion mit dem Geheimtext  $c$  und der Zufallszahl  $k$  als Eingabeparameter. Der Kommunikationspartner kann anschließend mit der Methode  $Decaps(c, sk')$  ebenfalls den Session Key ermitteln. Dabei wird zunächst der Geheimtext über die Methode  $PKE.Dec(c, sk)$  mithilfe des privaten Schlüssels entschlüsselt. Daraufhin können die beiden Zufallszahlen  $r'$  und  $k'$  durch eine Hashfunktion mit dem Klartext und dem Hashwert des öffentlichen Schlüssels berechnet werden. Abschließend wird geprüft, ob der erhaltene Geheimtext  $c$  mit dem Ergebnis der Verschlüsselungsmethode des PKE-Verfahrens übereinstimmt. Wenn dies der Fall ist, wird der Hashwert aus dem Geheimtext und der Zufallszahl  $k'$  gebildet, der dem Session Key  $ss'$  entspricht. Sollte keine Übereinstimmung vorliegen, wird der Hashwert aus dem Geheimtext und der Zufallszahl  $s$  generiert [43, S. 19ff.].

Die empfohlenen zwei Parametersets des NewHope-Algorithmus zur Schlüsseleinigung werden in die beiden Gitterdimensionen 512 und 1024 unterteilt. Andere Dimensionen sollten nicht verwendet werden, um die Sicherheitseigenschaften des

zugrundeliegenden Ring-LWE-Problems zu gewährleisten. Die empfohlenen zwei Parametersets stellen Verfahren der niedrigsten und höchsten Sicherheitskategorie dar und verwenden ebenfalls die Hashfunktion SHAKE256. In Tabelle 11 ist ersichtlich, welche Schlüssellängen in Bytes die Parametersets mit den beiden Sicherheitskategorien aufweisen [44, S.19f.].

**Tabelle 11: Kenngrößen der NewHope Parametersets (Längen in Bytes)**

Parameterset	Länge Public Key	Länge Private Key	Länge Geheimtext	Länge Session Key	Sicherheitskategorie
<b>NewHope512</b>	928	1.888	1.120	32	1
<b>NewHope1024</b>	1.824	3.680	2.208	32	5

Die Schlüsselgenerierungsmethode des KEM-Verfahrens weist im Wesentlichen die gleiche Funktionsweise wie die der FrodoKEM-Methode auf. In der nachfolgenden Methode  $Encaps(pk)$  werden hingegen mehrere Hashwerte von einer generierten Zufallszahl und deren PKE-Verschlüsselung gebildet und sogar gekapselt angewendet. Wie bei den vorherigen Algorithmen wird auch hier der Geheimtext  $c$  übergeben, der mithilfe des privaten Schlüssels entschlüsselt werden kann, um anschließend den gleichen Session Key berechnen zu können [44, S.16].

Für das SIKE-Verfahren werden vier mögliche Parametersets von den Entwicklern vorgeschlagen, die sich unter anderem in der Größe des zugrundeliegenden endlichen Körpers unterscheiden. Die verwendete Hashfunktion stellt auch hier SHAKE256 dar. Die Entwickler bieten zudem eine Komprimierungsfunktion der Schlüssellängen an. Dadurch ist es beispielsweise vor Übermittlung des öffentlichen Schlüssels oder des Geheimtexts möglich, diese zu verkleinern und nach der Übertragung wieder zu dekomprimieren. In Tabelle 12 werden die Kenngrößen der jeweiligen Parametersets aufgezeigt [45, S. 32].

Tabelle 12: Kenngrößen der SIKE Parametersets (Längen in Bytes)

Parameter-set	Länge Public Key	Länge Private Key	Länge Geheimtext	Länge Session Key	Sicherheits-kategorie
<b>SIKEp434</b>	330 (compressed 197)	374 (compressed 350)	346 (compressed 236)	16	1
<b>SIKEp503</b>	378 (compressed 225)	434 (compressed 407)	402 (compressed 280)	24	2
<b>SIKEp610</b>	462 (compressed 274)	524 (compressed 491)	486 (compressed 336)	24	3
<b>SIKEp751</b>	564 (compressed 335)	644 (compressed 602)	596 (compressed 410)	32	5

Im Wesentlichen ist die Funktionsweise identisch mit denen der vorangegangenen Verfahren. Jedoch setzt sich der Geheimtext aus den zwei Teilen  $c_0$  und  $c_1$  zusammen, die für die PKE-Verschlüsselungsmethoden als Berechnungsgrundlage dienen [45, S. 18].

### 4.3 Testszzenarien

Zu beiden auf Python-Code basierenden PKE-Algorithmen werden einerseits ihre Laufzeiten und andererseits ihre erzeugten Schlüsselgrößen gegenübergestellt. Bei der Auswahl der Parameter werden im Gegensatz zu den KEM-Parametersets kleinere Werte festgelegt, da die Ver- und Entschlüsselung einer deutlich größeren Datei anstelle eines Sitzungsschlüssels durchgeführt werden soll. Die ausgewählten Parameter werden daher für das McEliece-Verfahren mit  $(M, N, T) = (6, 63, 8)$  und für das NTRUEncrypt-Verfahren mit  $(N, P, Q) = (168, 3, 128)$  definiert.

In den nachfolgend durchzuführenden Tests werden die Referenzimplementierungen der Signaturerstellungsverfahren Rainbow-Vc und SPHINCS+-256s hinsichtlich ihrer Performance miteinander verglichen. Die zugrundeliegende Hashfunktion der Algorithmen stellt dabei die SHAKE256-Hashfunktion dar. Um die Vergleichbarkeit der gewählten Algorithmen zu gewährleisten, wurden Parametersets der gleichen Sicherheitsstufe ohne Kompression oder Performanceverbesserung gewählt.



Bei der Auswahl der Referenzimplementierung des jeweiligen Schlüsselaustauschverfahrens für die praktische Laufzeitmessung wurden ebenfalls die Parametersets mit gleicher Sicherheitskategorie und Hashfunktion gewählt. Daher werden nachfolgend die Parametersets McEliece6960119, Frodo-1344, NewHope1024 und SIKEp751 implementiert und gegenübergestellt. Die Länge des Sessions Keys ist bei allen Algorithmen identisch und kann bei der Evaluation daher vernachlässigt werden. Die Varianten mit Komprimierungsfunktion werden hierbei nicht betrachtet.

#### **4.3.1 Definition und Vorbereitung der Testszenarien**

Damit die Algorithmen hinsichtlich ihrer Laufzeit vergleichbar gegenübergestellt werden können, wird die Zeit eines einfachen Prozessorschritts, eines sogenannten CPU-Zyklus (englisch „Cycle“), einer Operation gemessen. Dies hat zudem den Vorteil, dass bereits gemessene Cycles pro Parameterset und pro verwendetem Prozessor in den Dokumentationen aufgelistet werden und als zusätzliche Referenz herangezogen werden können. Um die Varianz der Ergebnisse zu reduzieren, wird bei den Verschlüsselungsalgorithmen der Mittelwert von zehn aufeinanderfolgenden Durchläufen gebildet. Die ausgelieferten Quellcodes der Signaturerstellungs- und Schlüsselaustauschverfahren bieten durch bereits implementierte Schleifen die Möglichkeit, die Anzahl der Tests zu definieren. Es werden einhundert automatisierte Tests pro Lauf durchgeführt und jeweils deren durchschnittliche Laufzeit ausgegeben. Der Mittelwert wird anschließend aus den Einzelergebnissen von zehn Durchläufen berechnet.

Bei der Evaluation der beiden Verschlüsselungsalgorithmen werden zudem die generierten Schlüssel hinsichtlich ihrer Größe verglichen. Um die Cycles zu erhalten, muss eine Anpassung des Python-Codes erfolgen. Relevante Auszüge aus den Quellcodes zur Berechnung der Cycles sind im Anhang A.1 und A.2 zu finden, die vor und nach dem eigentlichen Methodenaufruf eingefügt werden. Des Weiteren wird der Aufruf der jeweiligen Methoden über die erstellte Bash-Datei Testrun.sh gestartet. Das für die Ver- und Entschlüsselung verwendete Testdokument entspricht dabei einer Textdatei mit einer Größe von 10 kB mit dem Inhalt „Test“ wiederholt aufgeführt. Die genannten Dateien befinden sich in der Anlage A.11 [3.2.1.4] und [3.2.2.4] dieser Arbeit.

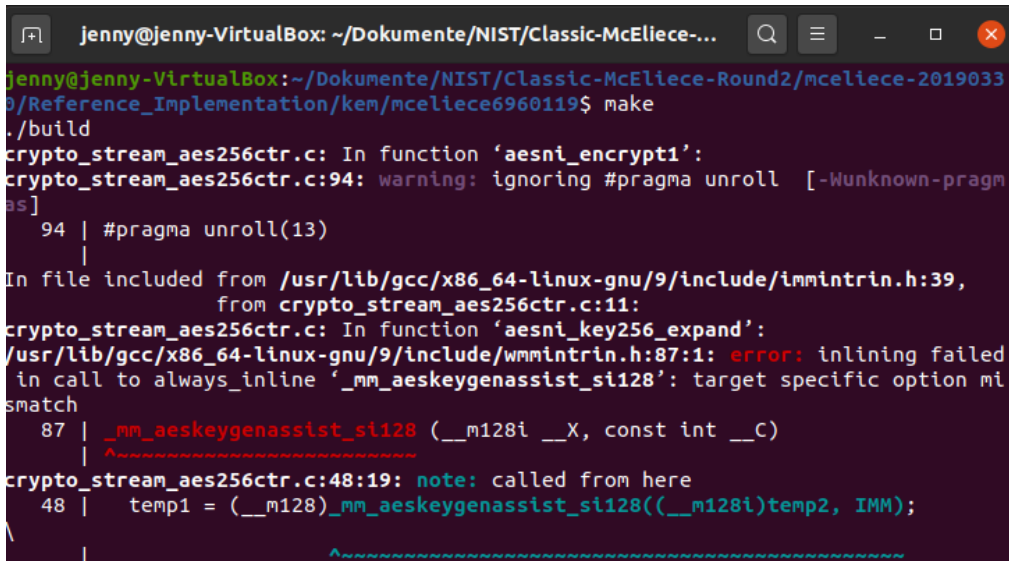
Um die Messung der CPU-Zyklen in den Signaturerstellungs- und KEM-Verfahren zu implementieren, wird auf die bereits verwendete Logik des SIKE-Algorithmus zurückgegriffen, die in der ausgelieferten test\_extras.c und deren Header-Datei zu

finden ist (vgl. Anlage A.11 [3.1]). Aufgrund der einheitlichen Verwendung, wird die Vergleichbarkeit der Messergebnisse gesichert. Die entsprechenden Methoden werden dahingehend erweitert und die relevanten Dateien eingebunden. In den Anhängen A.3 bis A.8 sind notwendige Anpassungen in den betroffenen Quellcode-Dateien auszugsweise dargestellt. In den Anlagen A.11 [3.2] bis [3.4] befinden sich im jeweiligen Unterordner „Modifizierter\_Quellcode“ die entsprechenden Dateien.

#### **4.3.2 Herausforderungen**

Zur Signaturerstellung wird eine zu signierende Nachricht benötigt. Die Entwickler implementierten zur automatisierten mehrfachen Durchführung die Erzeugung einer sogenannten Request-Datei. Damit wird bei Aufruf des Programms eine Textdatei mit hundert zufälligen Nachrichten mit unterschiedlicher Länge erzeugt. Auf Basis dieser Datei wird die Response-Datei erstellt. Die in der Request-Datei hinterlegten Nachrichten werden nach Schlüsselerzeugung signiert, woraufhin deren Signatur geprüft und die Ergebnisse entsprechend in der Response-Datei festgehalten werden. Um eine einheitliche Datenbasis zu erhalten, muss daher bei den manuellen Durchläufen nur beim ersten Test die Request-Datei erzeugt und anschließend der entsprechende Quellcode auskommentiert werden. Somit sind die zu signierenden Nachrichten stets identisch. Ein Auszug aus den erzeugten Dateien eines SPHINCS+-Testlaufs sind im Anhang A.9 und Beispieldateien für beide Algorithmen in den Anlagen A.11 [3.3.1.3] und [3.3.2.3] zu finden.

Die bereitgestellten Quellcodes zum McEliece-Schlüsselaustauschverfahren können auf der definierten Testumgebung mit AMD Phenom Prozessor nicht kompiliert werden (vgl. Bild 14). Grund dafür ist die fehlende AES-Befehlssatzerweiterung AES-NI. Diese wird zur Beschleunigung von AES Ver- und Entschlüsselungsoperationen verwendet [46]. Da der zugrundeliegende Prozessor die benötigten Anforderungen nicht erfüllt, werden die Testläufe alternativ auf einem Laptop mit Intel Core i7-7500U Prozessor mit 2.70 GHz und einer identisch konfigurierten virtuellen Maschine durchgeführt.



```

jenny@jenny-VirtualBox: ~/Dokumente/NIST/Classic-McEliece-...
jenny@jenny-VirtualBox:~/Dokumente/NIST/Classic-McEliece-Round2/mceliece-20190330/Reference_Implementation/kem/mceliece6960119$ make
./build
crypto_stream_aes256ctr.c: In function 'aesni_encrypt1':
crypto_stream_aes256ctr.c:94: warning: ignoring #pragma unroll [-Wunknown-pragmas]
   94 | #pragma unroll(13)
      |
In file included from /usr/lib/gcc/x86_64-linux-gnu/9/include/immintrin.h:39,
      from crypto_stream_aes256ctr.c:11:
crypto_stream_aes256ctr.c: In function 'aesni_key256_expand':
/usr/lib/gcc/x86_64-linux-gnu/9/include/wmmmintrin.h:87:1: error: inlining failed in call to always_inline '_mm_aeskeygenassist_si128': target specific option mismatch
   87 | _mm_aeskeygenassist_si128 (__m128i __X, const int __C)
      | ^
crypto_stream_aes256ctr.c:48:19: note: called from here
   48 |     temp1 = (__m128i)_mm_aeskeygenassist_si128((__m128i)temp2, IMM);

```

Bild 14: AES-NI Fehler beim Kompilieren der McEliece-Sourcen

### 4.3.3 Testergebnisse

Die Laufzeitergebnisse der PKE-Algorithmen basieren auf der jeweiligen Ausführung des erstellten Skripts Testrun.sh. Diese beinhalten den Aufruf der entsprechenden Methoden und werden im Bild 15 und Bild 16 aufgezeigt.

```

#/bin/sh
./mceliece.py -i gen 6 63 8 sk pk
./mceliece.py -b enc pk.npz Testfile.txt > Encrypted_Testfile.txt
./mceliece.py -b dec sk.npz Encrypted_Testfile.txt

```

Bild 15: Testrun.sh zum Aufruf der „mceliece.py“-Methoden

```

#/bin/sh
./ntru.py -i gen 167 3 128 sk pk
./ntru.py -b enc pk.npz Testfile.txt > Encrypted_Testfile.txt
./ntru.py -b dec sk.npz Encrypted_Testfile.txt

```

Bild 16: Testrun.sh zum Aufruf der „ntru.py“-Methoden

Zunächst werden die Schlüssel pk.npz (Public Key) und sk.npz (Private Key) erzeugt, um anschließend die Datei Testfile.txt zu verschlüsseln. Der verschlüsselte Inhalt wird in der Textdatei Encrypted\_Testfile.txt gespeichert. Referenzdateien dazu befinden sich in

den Anlagen A.11 [3.2.1.3] und [3.2.1.4] für McEliece und in A.11 [3.2.2.3] und [3.2.2.4] für NTRUEncrypt. Daraufhin wird diese Datei wieder entschlüsselt und der ursprüngliche Text mit den gemessenen Laufzeiten in der Konsole ausgegeben (vgl. Bild 17 und Bild 18).

```

jenny@jenny-VirtualBox: ~/Dokumente/mceliece-master
jenny@jenny-VirtualBox:~/Dokumente/mceliece-master$ ./Testrun.sh
KeyGen runs in .....112853197819
Encrypt runs in .....99331998751
Decrypt runs in .....1765636086378
test test test test test test test test test test test test test test test
test test test test test test test test test test test test test test test
test test test test test test test test test test test test test test test
test test test test test test test test test test test test test test test
test test test test test test test test test test test test test test test
test test test test test test test test test test test test test test test

```

**Bild 17: McEliece PKE – Auszug aus der Ausgabe von ./Testrun.sh**

```

jenny@jenny-VirtualBox: ~/Dokumente/ntru-master
jenny@jenny-VirtualBox:~/Dokumente/ntru-master$ ./Testrun.sh
KeyGen runs in .....22245407763
Encrypt runs in .....579867867503
Decrypt runs in .....875340066830
test test test test test test test test test test test test test test test
test test test test test test test test test test test test test test test
test test test test test test test test test test test test test test test
test test test test test test test test test test test test test test test
test test test test test test test test test test test test test test test

```

**Bild 18: NTRUEncrypt – Auszug aus der Ausgabe von ./Testrun.sh**

Die Schlüssellängen weisen bei jedem der zehn Durchläufe eine identische Größe auf. Die Einzelergebnisse der Laufzeitmessungen sind dem Anhang A.10 zu entnehmen. Die Mittelwerte werden auf deren Basis berechnet und mit den Schlüssellängen in Bytes in Tabelle 13 dargestellt.

Tabelle 13: PKE – Mittelwerte der gemessenen Cycles (Anzahl) und Schlüssellängen in Bytes

Verfahren	Schlüssel- erstellung	Verschlüsselung	Entschlüsselung	Länge pk	Länge sk
McEliece	113.360.682.212	101.178.165.144	1.775.342.791.271	5.090	54.613
NTRU	22.716.583.483	630.727.408.998	889.222.522.982	1.968	1.493

Die Algorithmen zur Signatuererstellung werden mittels des bereitgestellten Programms PQCgenKAT\_sign des jeweiligen Verfahrens getestet (vgl. A.11 [3.3.1.2] und [3.3.2.2]). Das Programm beinhaltet den Aufruf der Methoden und die Erstellung der Request- und der Response-Datei. Nach Anpassung der Quellcodes enthält die Ausgabe Informationen zu den benötigten Cycles pro Methode. In Bild 19 ist auszugsweise die Durchführung der Performancemessung des Rainbow-Verfahrens und in Bild 20 die Ausgabe des SPHINCS+-Verfahrens dargestellt.

```

jenny@jenny-VirtualBox: ~/Dokumente/NIST/Rainbow-Round2/...
jenny@jenny-VirtualBox:~/Dokumente/NIST/Rainbow-Round2/rainbow/Reference_Implementations$ ./PQCgenKAT_sign
KeyGen runs in ..... 1729380536
Sign runs in ..... 11247541
Verify runs in ..... 11659927
jenny@jenny-VirtualBox:~/Dokumente/NIST/Rainbow-Round2/rainbow/Reference_Implementations$ ./PQCgenKAT_sign
KeyGen runs in ..... 1321245158
Sign runs in ..... 8188840
    
```

Bild 19: Rainbow – Auszug aus der Ausgabe von ./PQCgenKAT\_sign

```

jenny@jenny-VirtualBox: ~/Dokumente/NIST/SPHINCS-Round2...
Sign runs in ..... 4695210787
Verify runs in ..... 6543057
jenny@jenny-VirtualBox:~/Dokumente/NIST/SPHINCS-Round2/NIST-PQ-Submission-SPHINCS-20190329/Reference_Implementation/crypto_sign/sphincs-shake256-256s-simple$ ./PQCgenKAT_sign
KeyGen runs in ..... 337698887
Sign runs in ..... 4071770711
Verify runs in ..... 5633840
    
```

Bild 20: SPHINCS+ – Auszug aus der Ausgabe von ./PQCgenKAT\_sign

Die Mittelwerte aus den Einzelergebnissen der Cycle-Messungen (vgl. Anhang A.10) werden in Tabelle 14 aufgezeigt.

**Tabelle 14: Signaturerstellungsverfahren – Mittelwerte der gemessenen Cycles (Anzahl)**

Verfahren	Schlüsselerstellung	Signaturerstellung	Signaturprüfung
<b>Vc (SHA512)</b>	1.046.100.847	7.000.834	7.585.878
<b>SPHINCS+-256s</b>	353.659.058	4.258.779.465	6.127.785

Werden diese mit den Werten aus den Dokumentationen verglichen, ist zu erkennen, dass die Messungen im Verhältnis annähernd den Testergebnissen entsprechen. Jedoch ist dabei zu beachten, dass die Entwickler nicht die gleichen Analyseumgebungen und Prozessoren verwendeten und die Vergleichbarkeit somit nicht vollständig gewährleistet werden kann. Die berechneten Mittelwerte der Referenzmessungen und die zugrundeliegenden Prozessoren werden zum Vergleich nachfolgend in Tabelle 15 dargestellt:

**Tabelle 15: Signaturerstellungsverfahren – Referenzmessungen der Cycles (Anzahl)**

Verfahren	Schlüssel- erstellung	Signaturerstellung	Signaturprüfung	Prozessor
<b>Vc (SHA512)</b>	757.000.000	3.640.000	2.390.000	Intel Xeon, 3.3 GHz (Skylake)
<b>SPHINCS+- 256s</b>	410.092.354	3.584.966.087	7.839.810	Intel Core i7- 4770K, 3.5 GHz (Haswell)

Für die Laufzeitmessungen der KEM-Verfahren werden von den Entwicklern unterschiedliche Lösungen bereitgestellt. Beispielsweise wird beim FrodoKEM-Verfahren durch Aufruf des Programms test\_KEM die Laufzeit zusätzlich in Sekunden ausgegeben (vgl. Bild 21).

```

jenny@jenny-VirtualBox:~/Dokumente/NIST/FrodoKEM-Round2/FrodoKEM-20190331/Reference_Implementatio
n/reference/FrodoKEM-1344/frodo$ ./test_KEM
=====
=====
Testing correctness of key encapsulation mechanism (KEM), system FrodoKEM-1344, tests for 100 ite
rations
=====
=====
KeyGen runs in ..... 92805629
Encaps runs in ..... 443994716
Decaps runs in ..... 452413822
Tests PASSED. All session keys matched.

Operation                Iterations  Total time (s)  Time(us): mean   pop. stdev
Key generation            35          1.026           29306.143        1194.963
KEM encapsulate           7           1.125           160676.000       14127.817
KEM decapsulate           7           1.012           144630.000       6142.207
    
```

Bild 21: FrodoKEM – Auszug aus der Ausgabe von ./test\_kem

Beim SIKE-Verfahren wird bereits ein Programm zur Ausführung eines Benchmarks über alle Parametersets mitgeliefert. Das Bash-Programm build\_run\_bench\_default.bash liefert die durchschnittliche Anzahl der CPU-Zyklen von einhundert Ausführungen (vgl. Bild 22). Diese Logik bildet die Basis für die Laufzeitmessungen der anderen Algorithmen. Auszüge aus den angepassten Ausgaben des McEliece KEM-Verfahrens und des NewHope-Algorithmus werden in den Bildern 23 und 24 dargestellt.

```

Encapsulation runs in ..... 5136147969 cycles
Decapsulation runs in ..... 8186191106 cycles
Performance of SIKEp751 (avg. of 100 runs):
Key generation runs in ..... 7237971928 cycles
Encapsulation runs in ..... 9748107039 cycles
Decapsulation runs in ..... 12025050581 cycles
jenny@jenny-VirtualBox:~/Dokumente/NIST/SIKE-Round2/SIKE/Reference_Implementation$ ./build_run_bench_defaul
t.bash
-- Configuring done
-- Generating done
-- Build files have been written to: /home/jenny/Dokumente/NIST/SIKE-Round2/SIKE/Reference_Implementation/b
uild
[ 27%] Built target sike_ref_for_test
[ 53%] Built target sike_ref
[ 65%] Built target sike_test
[ 70%] Built target sikep434_ref
[ 74%] Built target kat_SIKEp434
[ 78%] Built target sikep503_ref
[ 82%] Built target kat_SIKEp503
[ 87%] Built target sikep610_ref
[ 91%] Built target kat_SIKEp610
[ 95%] Built target sikep751_ref
[100%] Built target kat_SIKEp751
Performance of SIKEp434 (avg. of 100 runs):
Key generation runs in ..... 2774537941 cycles
Encapsulation runs in ..... 3272232627 cycles
Decapsulation runs in ..... 3170876104 cycles
Performance of SIKEp503 (avg. of 100 runs):
    
```

Bild 22: SIKE – Auszug aus der Ausgabe von ./build\_run\_bench\_default.bash,

```

KeyGen runs in ..... 874213760
Encaps runs in ..... 963558
Decaps runs in ..... 165820057
jenny@jenny-VirtualBox:~/Downloads/Classic-McEliece-Round2/mceliece-20190330/Reference_Implementation/kem/mceliece6960119$ ./run
KeyGen runs in ..... 904146124
Encaps runs in ..... 889804
Decaps runs in ..... 149852902
jenny@jenny-VirtualBox:~/Downloads/Classic-McEliece-Round2/mceliece-20190330/Ref

```

Bild 23: McEliece KEM – Auszug aus der Ausgabe von ./run

```

jenny@jenny-VirtualBox:~/Dokumente/NIST/NewHope-Round2/NIST-PQ-Submission-NewHope-20190330/Reference_Implementation/crypto_kem/newhope1024cca$ ./PQCgenKAT_kem
KeyGen runs in ..... 471894
Encaps runs in ..... 1954690
Decaps runs in ..... 2105468
jenny@jenny-VirtualBox:~/Dokumente/NIST/NewHope-Round2/NIST-PQ-Submission-NewHope-20190330/Reference_Implementation/crypto_kem/newhope1024cca$ ./PQCgenKAT_kem
KeyGen runs in ..... 596408
Encaps runs in ..... 821176

```

Bild 24: NewHope – Auszug aus der Ausgabe von ./PQCgenKAT\_kem

Die Einzelergebnisse sind ebenfalls im Anhang A.10 zu finden, wobei die durchschnittliche Anzahl der Cycles wie folgt in Tabelle 16 zusammengefasst werden kann.

Tabelle 16: KEM – Mittelwerte der gemessenen Cycles (Anzahl)

Verfahren	Schlüsselerstellung	Encaps	Decaps
McEliece6960119	897.768.632	936.699	164.040.581
Frodo-1344	100.171.598	460.733.264	467.003.605
NewHope1024	525.350	1.356.684	1.167.491
SIKEp751	7.087.622.338	9.736.084.170	11.732.146.463

Die Referenzmessungen beruhen beim McEliece- als auch beim NewHope-Verfahren und beim SIKE- als auch FrodoKEM-Verfahren auf den gleichen Prozessor. Jedoch entspricht die zugrundeliegende Hashfunktion der Referenzmessungen bei FrodoKEM



nicht SHAKE256, sondern lediglich SHAKE128. Wie bereits bei den Signaturerstellungsalgorithmen stehen die Werte annähernd im gleichen Verhältnis zueinander. In Tabelle 17 werden die entsprechenden Durchschnittswerte und die zugrundeliegenden Prozessoren der Referenztests abgebildet.

**Tabelle 17: KEM – Referenzmessungen der Cycles (Anzahl)**

Verfahren	Schlüssel- erstellung	Encaps	Decaps	Prozessor
<b>McEliece6960119</b>	1.202.081.992	156.826	303.207	Intel Core i7-4770K, 3.5 GHz (Haswell)
<b>Frodo-1344</b>	30.301.000	32.611.000	32.387.000	Intel Core i7-6700, 3.4 GHz (Skylake)
<b>NewHope1024</b>	244.944	377.092	437.056	Intel Core i7-4770K, 3.5 GHz (Haswell)
<b>SIKEp751</b>	4.743.861.000	6.534.356.000	8.016.158.000	Intel Core i7-6700, 3.4 GHz (Skylake)

#### 4.4 Bewertung der Ergebnisse

Anhand der zuvor erläuterten Eigenschaften und praktischen Implementierungen ist es möglich, die ausgewählten Algorithmen hinsichtlich der definierten Bewertungskriterien abschließend gegenüberzustellen. Die Vergleiche werden innerhalb der Anwendungsbereiche vorgenommen.

##### 4.4.1 Ver- und Entschlüsselungsverfahren

Werden das McEliece- und das NTRUEncrypt-Verfahren anhand der Testergebnisse miteinander verglichen, wird deutlich, dass das McEliece-Verfahren in der Praxis schlechter abschneidet als der NTRUEncrypt-Algorithmus. Lediglich die Laufzeit der Verschlüsselung erzielt ein besseres Ergebnis. Der Performance-Verlust bei der Entschlüsselung ist unter anderem auf die große Schlüssellänge des privaten Schlüssels

zurückzuführen. Dennoch bieten codebasierte Kryptosysteme wie McEliece den Vorteil, dass diese einfach in Soft- und Hardware implementiert werden können, da diese auf einfachen mathematischen Operationen beruhen. Dennoch kann eine hohe Sicherheit nur durch ausreichend große Schlüssel gewährleistet werden. Dahingegen bieten gitterbasierte Verfahren bei zu kurz gewählten Schlüsseln ein hohes Sicherheitsniveau. Die Methoden basieren auf Vektoren im Gitter und sind daher ebenfalls vergleichsweise einfach in Soft- und Hardware zu implementieren. Gitterbasierte Verfahren werden für praktische Umsetzungen bisher bevorzugt. Die in Kapitel 3.3.6 auf Basis der theoretischen Eigenschaften vorgenommene Bewertung der Kategorien hat sich auch bei der praktischen Implementierung bewiesen (vgl. Tabelle 4). Nachfolgend in Tabelle 18 werden die betrachteten Algorithmen anhand ihrer Bewertungskriterien gegenübergestellt.

**Tabelle 18: Bewertung der PKE-Algorithmen im Vergleich**

<b>Bewertungskriterien</b>	<b>McEliece</b>	<b>NTRUEncrypt</b>
<b>Performance</b>	-	+
<b>Schlüssellängen</b>	-	+
<b>Sicherheit</b>	-	+
<b>Implementierbarkeit</b>	+	+

#### 4.4.2 Signaturerstellungsverfahren

Bei der Betrachtung der Ergebnisse der Performancemessung der Signaturerstellungsverfahren zeigt sich, dass der SPHINCS+-Algorithmus zwar in der Schlüsselerstellung eine bessere Laufzeit aufweist, jedoch bei der Signaturerstellung dem Rainbow-Algorithmus unterliegt. Die Performance zur Prüfung der Signatur ist bei beiden Algorithmen annähernd gleich hoch. Zudem sind die Schlüssellängen beim Rainbow-Verfahren weitaus größer, dahingegen ist die Länge der Signatur deutlich geringer (vgl. Tabelle 19).

**Tabelle 19: Vergleich der Kenngrößen der Signaturerzeugungsverfahren**

Verfahren	Länge Public Key	Länge Private Key	Länge Signatur	Sicherheitskategorie
Rainbow-Vc (SHA512)	1.705.500	1.227.100	204	5
SPHINCS+-256s	64	128	29.792	5

Da multivariate Verfahren wie Rainbow auf einfachen Operationen der linearen Algebra basieren, ist eine Implementierung auf jeglichen Geräten möglich. Dabei ist der Speicheraufwand der Schlüssel nicht zu vernachlässigen. Trotz der einfachen mathematischen Grundlagen konnte das Kryptosystem bisher nicht durch Kryptoanalyse gebrochen werden. Dahingegen beruht die Sicherheit des SPHINCS+-Verfahren auf der verwendeten Hashfunktion. Durch die hohe Flexibilität des Verfahrens kann die Hashfunktion bei Bedarf jederzeit ausgetauscht werden. Je nachdem für welchen Zweck ein Signaturprozess benötigt wird, kann ein multivariater oder hashbasierter Ansatz gewählt werden. Besonders bei häufigem Austausch von Zertifikaten sind kleinere Schlüssel von Vorteil und eine hashbasierte Lösung ist zu bevorzugen. In der folgenden Tabelle 20 wird die Bewertung der jeweiligen Algorithmen abschließend aufgezeigt.

**Tabelle 20: Bewertung der Signaturerstellungsverfahren im Vergleich**

Bewertungskriterien	Rainbow	SPHINCS+
Performance	0	0
Signaturlänge	+	-
Schlüssellängen	-	+
Sicherheit	+	+
Implementierbarkeit	+	+

#### 4.4.3 Schlüsselaustauschverfahren

Beim Vergleich der Laufzeitmessungen der Schlüsselaustauschverfahren ist klar erkennbar, dass das SIKE-Verfahren das langsamste Verfahren darstellt. Dennoch bietet es den Vorteil sehr kurze Schlüssellängen zu erzeugen (vgl. Tabelle 21).

**Tabelle 21: Vergleich der Kenngrößen der Schlüsselaustauschverfahren**

Verfahren	Länge Public Key	Länge Private Key	Länge Geheimtext	Sicherheitskategorie
<b>McEliece6960119</b>	1.044.992	1.058.884	240	5
<b>Frodo-1344</b>	21.520	43.088	21.632	5
<b>NewHope1024</b>	1.824	3.680	2.208	5
<b>SIKEp751</b>	564	644	596	5

Ebenfalls weist der NewHope-Algorithmus vergleichsweise kurze Schlüssellängen auf und ist zudem das Verfahren mit der besten Performance. Das McEliece- und das FrodoKEM-Verfahren haben dahingegen eine mittlere Laufzeit, wobei die Schlüssellängen bei McEliece im Vergleich deutlich größer sind. Da die Entwicklung der isogeniebasierten Verfahren wie SIKE noch am Anfang steht, kann eine ausreichende Sicherheit noch nicht gewährleistet werden. Die Implementierbarkeit ist aufgrund der verwendeten elliptischen Kurven mit wenig Aufwand an bestehende Implementierungen in diesem Bereich anzupassen. NewHope und FrodoKEM hingegen weisen einen kurzen Quellcode auf und sind daher einfach in Soft- und Hardware zu implementieren. Dies reduziert Fehler bei der Implementierung und erhöht dahingehend die Sicherheit der Verfahren. Der McEliece-Algorithmus kann ebenfalls durch Verwendung von einfachen mathematischen Objekten wie binären Vektoren und Operationen wie der binären Matrix-Vektor-Multiplikation problemlos in Soft- und Hardware implementiert werden. Anhand der in Tabelle 22 vorgenommenen Bewertung der Verfahren wird deutlich, dass insbesondere NewHope für ein Schlüsselaustausch-Verfahren geeignet ist. Jedoch ist die geringe Auswahl der Parametersets nicht zu vernachlässigen. Die fehlende Flexibilität bei der Auswahl eines geeigneten Sicherheitsniveaus ist dabei von Nachteil. Soll jedoch eine hohe Sicherheitsstufe gewährleistet werden, ist das

verwendete Parameterset zu empfehlen. Liegt der Fokus auf kurzen Schlüssellängen, stellt das SIKE-Verfahren die optimale Lösung dar.

**Tabelle 22: Bewertung der KEM-Verfahren im Vergleich**

<b>Bewertungskriterium</b>	<b>McEliece</b>	<b>FrodoKEM</b>	<b>NewHope</b>	<b>SIKE</b>
<b>Performance</b>	0	0	+	-
<b>Schlüssellängen</b>	-	0	+	+
<b>Sicherheit</b>	+	+	+	-
<b>Implementierbarkeit</b>	+	+	+	0

## 5 Fazit und Ausblick

Die zunehmende Bedrohung durch Quantencomputer zeigt sich durch rasante Entwicklungen in der Quantenforschung. Klassische Verschlüsselungsalgorithmen bieten zwar aktuell noch eine ausreichende Sicherheit, jedoch zeigen die erläuterten klassischen Angriffsmöglichkeiten als auch theoretisch entwickelte Quantenalgorithmen, dass deren Wirksamkeit durchaus bald an ihre Grenzen geraten kann. Quantencomputer-resistenten Verfahren, die bereits heute eingesetzt werden können, müssen daher zeitnah entwickelt und geprüft werden. Besonders der Standardisierungsprozess der NIST bietet dazu den passenden Rahmen und treibt die Entwicklung voran. Ziel dabei ist es, die Public-Key-Verfahren zu vergleichen und eine Empfehlung für einen Standard auszusprechen. Die aktuell vorliegenden Verfahren können in fünf grundlegende Kategorien eingeteilt werden, die jeweils einen anderen Ansatz verfolgen. Die praktische Implementierung auf einem klassischen Computer von ausgewählten Verfahren dieser Kategorien basieren auf modifizierten veröffentlichten Quellcodes. Die Evaluation wurde in die drei Anwendungsbereiche Ver- und Entschlüsselung, Schlüsselaustausch, Signaturerstellung untergliedert.

Die aus dieser Arbeit resultierenden Quellcodes auf Basis der veröffentlichten Algorithmen bieten die Möglichkeit, die ausgewählten Verfahren auf unterschiedlichen Prozessoren zu implementieren und aufgrund der integrierten Cycle-Messung vergleichbar gegenüberzustellen. Die aufgetretenen Herausforderungen und deren Lösungen können Hilfestellungen für zukünftige Analysen darstellen. Obwohl es sich bei der Evaluation lediglich um einen POC handelt, konnten die in der Theorie erläuterten Vor- und Nachteile der Quantencomputer-resistenten Verfahren in der praktischen Umsetzung verifiziert werden.

Das Ziel dieser Arbeit ist die Beantwortung der Forschungsfrage, welches der Quantencomputer-resistenten Public-Key-Verfahren am besten für den Einsatz auf klassischen Computern geeignet ist. Bei der Evaluation der Kategorien wird deutlich, dass eine Empfehlung für die beste Kategorie nicht ausgesprochen werden kann. Jede der einzelnen Kategorien hat gewisse Vor- und Nachteile und kann je Anwendungsgebiet eine mehr oder weniger gut geeignete Lösung darstellen. Bei der Auswahl eines Algorithmus kann der Fokus beispielsweise entweder auf Laufzeit, Sicherheit oder Speicherverbrauch liegen. Pro Algorithmus können zudem entsprechende Parameter gewählt werden, die für die Zielerreichung in Frage kommen.

Da es sich bei der Post-Quantum Kryptographie noch um ein sehr junges Forschungsgebiet handelt, besteht für die Weiterentwicklung eines jeden Algorithmus enormes Potential. Die Algorithmen werden zunehmend performanter und ressourcenschonender. Bereits anhand der vorgestellten Parametersets der jeweiligen Algorithmen zeigt sich, dass diese je nach Anforderung eine gewisse Flexibilität bieten. Diese Flexibilität spielt zukünftig eine bedeutende Rolle, da sich auch die Kryptoanalyse der Verfahren noch am Anfang befindet. So ist es beispielsweise möglich, dass zeitnah Lösungen entwickelt werden, die sich den grundlegenden mathematischen Problemen nähern und die Sicherheit der Verfahren beeinträchtigen. Wie bereits mit der Implementierung des NewHope-Verfahrens in den Webbrowser Google Chrome bewiesen, ist ein Hybridverfahren aus klassischen und Quantencomputer-resistenten Verfahren vorerst eine gute Möglichkeit, die Vorteile beider Methoden zu vereinen. Einerseits gewährleistet dies eine ausreichende Sicherheit gegenüber Quantenalgorithmen und andererseits werden potentielle Risiken durch verbesserte Kryptoanalyse und mögliche Laufzeiteinbußen minimiert. Neben der Post-Quantum Kryptographie ist es zukünftig zudem durchaus denkbar, Verfahren der Quantenkryptographie anzuwenden, um hochkritische Prozesse und Daten zu sichern. Dies wäre jedoch aufgrund geringer Kommerzialisierung zunächst nur für einzelne Bereiche umsetzbar.

---

**6 Literaturverzeichnis**

- [1] S. Filipp, Quantencomputer: Beginn der kommerziellen Quanten-Ära. [Online] Verfügbar unter: <https://www.ibm.com/de-de/blogs/think/2018/02/23/quantencomputer/>. Zugriff am: 22. Februar 2020.
- [2] M. Lindinger, Der Quantencomputer verlässt das Labor. [Online] Verfügbar unter: <https://www.faz.net/aktuell/wissen/computer-mathematik/ibm-praesentiert-den-ersten-kommerziellen-quantencomputer-15980196.html>. Zugriff am: 22. Februar 2020.
- [3] A. Grävemeyer, Jetzt auch offiziell: Googles Quantencomputer beweist "Quantum Supremacy". [Online] Verfügbar unter: <https://www.heise.de/newsticker/meldung/Jetzt-auch-offiziell-Googles-Quantencomputer-zeigt-Quantum-Supremacy-4565771.html>. Zugriff am: 22. Februar 2020.
- [4] BSI, Entwicklungsstand Quantencomputer. [Online] Verfügbar unter: <https://www.bsi.bund.de/DE/Publikationen/Studien/Quantencomputer/quantencomputer.html>. Zugriff am: 22. Februar 2020.
- [5] H. Hagemeyer, Kryptografie – heute und zukünftig: Grundbaustein für IT-Sicherheit. In *Datenschutz und Datensicherheit – DuD*, vol. 43, Wiesbaden: Springer-Verlag, 2019, S. 361-365.
- [6] D. Wätjen, Kryptographie: Grundlagen, Algorithmen, Protokolle. 3. Aufl. Wiesbaden: Springer-Verlag, 2018.
- [7] BSI, Kryptografie. [Online] Verfügbar unter: [https://www.bsi.bund.de/DE/Themen/Kryptografie\\_Kryptotechnologie/Kryptografie/kryptografie\\_node.html](https://www.bsi.bund.de/DE/Themen/Kryptografie_Kryptotechnologie/Kryptografie/kryptografie_node.html). Zugriff am: 31. Januar 2020.
- [8] A. Beutelspacher, Kryptologie: Eine Einführung in die Wissenschaft vom Verschlüsseln, Verbergen und Verheimlichen. 10. Aufl. Wiesbaden: Springer-Verlag, 2015.
- [9] N. Pohlmann, Cyber-Sicherheit: Das Lehrbuch für Konzepte, Prinzipien, Mechanismen, Architekturen und Eigenschaften von Cyber-Sicherheitssystemen in der Digitalisierung. 1. Aufl. Wiesbaden: Springer-Verlag, 2019.
- [10] Prof. Dr.-Ing. habil. A. Ahrens, „Studienbrief: Kryptografische Methoden und Anwendungen“. Studienbrief\_Kryptografische Methoden und Anwendungen.pdf, Wismar, 2019.
- [11] BSI, Kryptographische Verfahren: Empfehlungen und Schlüssellängen: BSI – Technische Richtlinie. BSI TR-02102-1. [Online] Verfügbar unter: [https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?\\_\\_blob=publicationFile&v=8](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=8). Zugriff am: 01. Februar 2020.



- 
- [12] Kryptografische Verfahren und ihre Sicherheit (Übersicht). [Online] Verfügbar unter: <https://www.elektronik-kompendium.de/sites/net/2006261.htm>. Zugriff am: 16. Februar 2020.
- [13] D. Bachfeld, Passwörter unknackbar speichern. [Online] Verfügbar unter: <https://m.heise.de/security/artikel/Passwoerter-unknackbar-speichern-1253931.html?seite=all>. Zugriff am: 21. März 2020.
- [14] Prof. Dr.-Ing. habil. A. Ahrens, „Studienbrief: Kryptoanalyse“. Studienbrief\_Kryptoanalyse.pdf, Wismar, 2019.
- [15] Post-Quanten-Kryptographie. [Online] Verfügbar unter: <https://www.kryptowissen.de/post-quanten-kryptographie.php> Zugriff am: 06. März 2020.
- [16] G. Brands, Einführung in die Quanteninformatik. 1. Aufl. Berlin/Heidelberg: Springer-Verlag, 2011.
- [17] M. Homeister, Quantum Computing verstehen: Grundlagen - Anwendungen - Perspektiven. 5. Aufl. Wiesbaden: Springer-Verlag, 2018.
- [18] G. Fürnkranz, Vision Quanten-Internet – Ultraschnell und hackersicher. 1. Aufl. Berlin/Heidelberg: Springer-Verlag, 2019.
- [19] L. Bruckert, J.-M. Schmidt, Post-Quanten-Kryptografie: Sicherheit in Zeiten des Quantencomputers. [Online] Verfügbar unter: <https://www.kes.info/archiv/leseproben/%202018/post-quanten-kryptografie/>. Zugriff am: 31. März 2020.
- [20] J. Ding, B.-Y. Yang, Multivariate Public Key Cryptography. In: *Post-Quantum Cryptography*, 1. Aufl. Berlin/Heidelberg: Springer-Verlag, 2009, S. 193-241.
- [21] R.-J. McEliece, A Public-Key Cryptosystem Based On Algebraic Coding Theory. [Online] Verfügbar unter: [https://ipnpr.jpl.nasa.gov/progress\\_report/42-44/44N.PDF](https://ipnpr.jpl.nasa.gov/progress_report/42-44/44N.PDF). Zugriff am: 03. April 2020.
- [22] Hochschule Augsburg, McEliece-Kryptosystem. [Online] Verfügbar unter: <https://glossar.hs-augsburg.de/McEliece-Kryptosystem>. Zugriff am: 03. April 2020.
- [23] Z. Li, C. Qu, X. Zhou, Review of Public-Key Cryptosystem Based on the Error Correcting Code. In *Wuhan University Journal of Natural Sciences*, vol. 19, no. 6., Berlin/Heidelberg: Springer-Verlag, 2014, S. 489-496.
- [24] O. Regev, Lattice-Based Cryptography in Advances. In: *Cryptology - CRYPTO 2006*, vol. 4117, Berlin/Heidelberg: Springer-Verlag, 2006, S. 131-141.
- [25] C. Gentry, A Fully Homomorphic Encryption Scheme. Stanford, 2009.
- [26] S.-H. Paeng, B. Jung, A Lattice Based Public Key Cryptosystem Using Polynomial Representations. In: *PKC 2003*, vol. 2567, Berlin/Heidelberg: Springer-Verlag, 2003, S. 292-308.
- [27] J. Hoffstein, J. Pipher, NTRU: A Ring-Based Public Key Cryptosystem. [Online] Verfügbar unter:

- <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.8422&rep=rep1&type=pdf>. Zugriff am: 13. April 2020.
- [28] E. Alkim, R. Avanzi, NewHope - Algorithm Specifications and Supporting Documentation. [Online] Verfügbar unter: <https://cryptojedi.org/papers/newhopenist-20171128.pdf>. Zugriff am: 14. April 2020.
- [29] J. Buchmann, E. Dahmen, Hash-based Digital Signature Schemes. In: *Post-Quantum Cryptography*, 1. Aufl. Berlin/Heidelberg: Springer-Verlag, 2009, S. 35-93.
- [30] J. Buchmann, E. Dahmen, XMSS - A Practical Forward Secure Signature Scheme based on Minimal Security Assumptions. [Online] Verfügbar unter: <https://eprint.iacr.org/2011/484.pdf>. Zugriff am: 17. April 2020.
- [31] D.-J. Bernstein, D. Hopwood, SPHINCS: Practical Stateless Hash-Based Signatures. In: *Advances in Cryptology - EUROCRYPT 2015*, vol. 9056, Berlin/Heidelberg: Springer-Verlag, 2015, S. 368-397.
- [32] S.-D. Galbraith, F. Vercauteren, Computational problems in supersingular elliptic curve isogenies. [Online] Verfügbar unter: <https://doi.org/10.1007/s11128-018-2023-6>. Zugriff am: 21. April 2020.
- [33] M. Meyer, S. Reith, Elliptische Kurven in der Post-Quantum-Kryptographie. [Online] Verfügbar unter: <https://doi.org/10.1007/s00591-018-00239-8>. Zugriff am: 21. April 2020.
- [34] NIST, Post-Quantum Cryptography. [Online] Verfügbar unter: <https://csrc.nist.gov/Projects/post-quantum-cryptography/round-2-submissions>. Zugriff am: 08. Mai 2020.
- [35] PQCrypto, PQCrypto ICT-645622. [Online] Verfügbar unter: <https://pqcrypto.eu.org/>. Zugriff am: 08. Mai 2020.
- [36] BSI, Migration zu Post-Quanten-Kryptografie - Handlungsempfehlungen des BSI. [Online] Verfügbar unter: [https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Krypto/Post-Quanten-Kryptografie.pdf;jsessionid=DF8ADB685250A12FE89ED55845E29108.2\\_cid360?\\_\\_blob=publicationFile&v=2#download=1](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Krypto/Post-Quanten-Kryptografie.pdf;jsessionid=DF8ADB685250A12FE89ED55845E29108.2_cid360?__blob=publicationFile&v=2#download=1). Zugriff am: 08. Mai 2020.
- [37] NIST, Post-Quantum Cryptography - Security (Evaluation Criteria). [Online] Verfügbar unter: [https://csrc.nist.gov/Projects/post-quantum-cryptography/Post-Quantum-Cryptography-Standardization/Evaluation-Criteria/Security-\(Evaluation-Criteria\)](https://csrc.nist.gov/Projects/post-quantum-cryptography/Post-Quantum-Cryptography-Standardization/Evaluation-Criteria/Security-(Evaluation-Criteria)). Zugriff am: 18. Mai 2020.
- [38] J. Krauze, Simple python implementation of McEliece cryptosystem. [Online] Verfügbar unter: <https://github.com/jkrauze/mceliece>. Zugriff am: 25. Mai 2020.
- [39] D.-J. Bernstein, T. Chou, T. Lange, Classic McEliece: conservative code-based cryptography. [Online] Verfügbar unter: <https://classic.mceliece.org/nist/mceliece-20190331.pdf>. Zugriff am: 25. Mai 2020.
- [40] J. Krauze, Simple python implementation of NTRUEncrypt cryptosystem. [Online] Verfügbar unter: <https://github.com/jkrauze/ntru>. Zugriff am: 25. Mai 2020.

- [41] J. Ding, Rainbow. [Online] Verfügbar unter: <https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-2/submissions/Rainbow-Round2.zip>. Zugriff am: 25. Mai 2020.
- [42] J.-P. Aumasson, D.-J. Bernstein, SPHINCS+ - Submission to the NIST post-quantum project. [Online] Verfügbar unter: <https://sphincs.org/data/sphincs+-round2-specification.pdf>. Zugriff am: 25. Mai 2020.
- [43] E. Alkim, J. Bos, FrodoKEM - Learning With Errors Key Encapsulation. [Online] Verfügbar unter: <https://frodokem.org/files/FrodoKEM-specification-20190330.pdf>. Zugriff am: 25. Mai 2020.
- [44] E. Alkim, R. Avanzi, NewHope - Algorithm Specifications And Supporting Documentation. [Online] Verfügbar unter: [https://newhopecrypto.org/data/NewHope\\_2019\\_04\\_10.pdf](https://newhopecrypto.org/data/NewHope_2019_04_10.pdf). Zugriff am: 25. Mai 2020.
- [45] D. Jao, R. Azarderakhsh, Supersingular Isogeny Key Encapsulation. [Online] Verfügbar unter: <https://sike.org/files/SIDH-spec.pdf>. Zugriff am: 25. Mai 2020.
- [46] Wikipedia, AES instruction set. [Online] Verfügbar unter: [https://en.wikipedia.org/wiki/AES\\_instruction\\_set](https://en.wikipedia.org/wiki/AES_instruction_set). Zugriff am: 20. Juni 2020.

## 7 Bilderverzeichnis

Bild 1: Ablauf einer symmetrischen Verschlüsselung .....	12
Bild 2: Ablauf einer asymmetrischen Verschlüsselung .....	15
Bild 3: Beispiel einer elliptischen Kurve .....	19
Bild 4: Vektordarstellung einer Superposition [vgl. 15, S. 23] .....	30
Bild 5: Allgemeine Darstellung der multivariaten Verfahren C* und HFE .....	38
Bild 6: Darstellung der Basen $B_g$ und $B_h$ in $\mathbb{R}^2$ .....	42
Bild 7: Aufbau eines Merkle-Baums mit dem öffentlichen und der privaten Schlüssel .....	46
Bild 8: Darstellung der XMSS Baumstruktur [vgl. 30, S. 4] .....	48
Bild 9: Abbildung einer Isogenie (Grad 1) zwischen $E_1$ und $E_2$ .....	50
Bild 10: Allgemeine Darstellung des SIDH-Schlüsselaustauschs [vgl. 33, S. 41] .....	51
Bild 11: Auszug aus der Ausgabe des Aufrufs „mceliece.py -h“ .....	62
Bild 12: Auszug aus der Ausgabe des Aufrufs „ntru.py -h“ .....	63
Bild 13: Genereller Ablauf eines KEM .....	67
Bild 14: AES-NI Fehler beim Kompilieren der McEliece-Sourcen .....	74
Bild 15: Testrun.sh zum Aufruf der „mceliece.py“-Methoden .....	74
Bild 16: Testrun.sh zum Aufruf der „ntru.py“-Methoden .....	74
Bild 17: McEliece PKE – Auszug aus der Ausgabe von ./Testrun.sh .....	75
Bild 18: NTRUEncrypt – Auszug aus der Ausgabe von ./Testrun.sh .....	75
Bild 19: Rainbow – Auszug aus der Ausgabe von ./PQCgenKAT_sign .....	76
Bild 20: SPHINCS+ – Auszug aus der Ausgabe von ./PQCgenKAT_sign .....	76
Bild 21: FrodoKEM – Auszug aus der Ausgabe von ./test_kem .....	78
Bild 22: SIKE – Auszug aus der Ausgabe von ./build_run_bench_default.bash .....	78
Bild 23: McEliece KEM – Auszug aus der Ausgabe von ./run .....	79
Bild 24: NewHope – Auszug aus der Ausgabe von ./PQCgenKAT_kem .....	79

---

**8 Tabellenverzeichnis**

Tabelle 1: Vergleich von symmetrischen Verfahren.....	14
Tabelle 2: Vergleich von asymmetrischen Verfahren.....	21
Tabelle 3: Vergleich von Hashfunktionen.....	25
Tabelle 4: Bewertung und Vergleich der Post-Quantum Kryptographie-Kategorien .....	55
Tabelle 5: Übersicht der Verfahren der zweiten Runde des NIST-Auswahlprozesses.....	57
Tabelle 6: Auswahl der Algorithmen für die Evaluation.....	60
Tabelle 7: Kenngrößen der Rainbow Parametersets (Längen in Bytes) .....	64
Tabelle 8: Kenngrößen der SPHINCS+ Parametersets (Längen in Bytes) .....	65
Tabelle 9: Kenngrößen der McEliece KEM Parametersets (Längen in Bytes) .....	68
Tabelle 10: Kenngrößen der FrodoKEM Parametersets (Längen in Bytes) .....	69
Tabelle 11: Kenngrößen der NewHope Parametersets (Längen in Bytes).....	70
Tabelle 12: Kenngrößen der SIKE Parametersets (Längen in Bytes) .....	71
Tabelle 13: PKE – Mittelwerte der gemessenen Cycles (Anzahl) und Schlüssellängen in Bytes .....	76
Tabelle 14: Signaturerstellungsverfahren – Mittelwerte der gemessenen Cycles (Anzahl) .....	77
Tabelle 15: Signaturerstellungsverfahren – Referenzmessungen der Cycles (Anzahl) .....	77
Tabelle 16: KEM – Mittelwerte der gemessenen Cycles (Anzahl) .....	79
Tabelle 17: KEM – Referenzmessungen der Cycles (Anzahl) .....	80
Tabelle 18: Bewertung der PKE-Algorithmen im Vergleich .....	81
Tabelle 19: Vergleich der Kenngrößen der Signaturerzeugungsverfahren .....	82
Tabelle 20: Bewertung der Signaturerstellungsverfahren im Vergleich .....	82
Tabelle 21: Vergleich der Kenngrößen der Schlüsselaustauschverfahren .....	83
Tabelle 22: Bewertung der KEM-Verfahren im Vergleich .....	84

---

**9 Anlagenverzeichnis und Anlagen**

A.1	McEliece-PKE: Auszug mceliece.py .....	94
A.2	NTRUEncrypt: Auszug ntru.py .....	95
A.3	Rainbow: Auszug PQCgenKAT_sign.c .....	96
A.4	SPHINCS+: Auszug PQCgenKAT_sign.c .....	98
A.5	McEliece-KEM: Auszug kat_kem.c .....	100
A.6	FrodoKEM: Auszug test_kem.c .....	102
A.7	NewHope: Auszug PQCgenKAT_kem.c .....	103
A.8	SIKE: Auszug test_sike.c .....	105
A.9	Beispiel Request- und Response-Datei eines SPHINCS+-Testlaufs .....	107
A.10	Einzelergebnisse der Performancemessungen .....	108
A.11	Anlagen USB-Stick .....	112

## A.1 McEliece-PKE: Auszug mceliece.py

Vor den jeweiligen Methoden der Schlüsselerzeugung, Verschlüsselung und Entschlüsselung (generate, encrypt, decrypt) wurde der Startwert durch die Standardfunktion count() ermittelt. Nach der Methodenausführung wurde mittels der Methode count\_end erneut die Anzahl der Cycles gemessen und der Startwert subtrahiert, um die eigentliche Cycle-Laufzeit zu erhalten. Abschließend wird der berechnete Wert ausgegeben.

```
# Generate the public/private keypair
if args['gen']:
    start=count()
    generate(int(args['M']), int(args['N']), int(args['T']),
            args['PRIV_KEY_FILE'], args['PUB_KEY_FILE'])
    elapsed=count_end()-start
    eprint(f'KeyGen runs in
    .....{elapsed}')

# Encryption
elif args['enc']:
    start=count()
    output = encrypt(args['PUB_KEY_FILE'], input_arr, block=block)
    elapsed=count_end()-start
    eprint(f'Encrypt runs in
    .....{elapsed}')

# Decryption
elif args['dec']:
    start=count()
    output = decrypt(args['PRIV_KEY_FILE'], input_arr, block=block)
    elapsed=count_end()-start
    eprint(f'Decrypt runs in
    .....{elapsed}')
```

## A.2 NTRUEncrypt: Auszug ntru.py

Vor den jeweiligen Methoden der Schlüsselerzeugung, Verschlüsselung und Entschlüsselung (generate, encrypt, decrypt) wurde der Startwert durch die Standardfunktion count() ermittelt. Nach der Methodenausführung wurde mittels der Methode count\_end erneut die Anzahl der Cycles gemessen und der Startwert subtrahiert, um die eigentliche Cycle-Laufzeit zu erhalten. Abschließend wird der berechnete Wert ausgegeben.

```
# Generate the public/private keypair
if args['gen']:
    start=count()
    generate(int(args['N']), int(args['P']), int(args['Q']),
            args['PRIV_KEY_FILE'], args['PUB_KEY_FILE'])
    elapsed=count_end()-start
    eprint(f'KeyGen runs in
    .....{elapsed}')

# Encryption
elif args['enc']:
    start=count()
    output = encrypt(args['PUB_KEY_FILE'], input_arr, bin_output=not
    poly_output, block=block)
    elapsed=count_end()-start
    eprint(f'Encrypt runs in
    .....{elapsed}')

# Decryption
elif args['dec']:
    start=count()
    output = decrypt(args['PRIV_KEY_FILE'], input_arr, bin_input=not
    poly_input, block=block)
    elapsed=count_end()-start
    print(f'Decrypt runs in
    .....{elapsed}')
```



### A.3 Rainbow: Auszug PQCgenKAT\_sign.c

Der aktuelle Stand der CPU-Zyklen wird vor der Ausführung der jeweiligen Methode zur Schlüsselerzeugung, Signaturerzeugung und Signaturprüfung (`crypto_sign_keypair`, `crypto_sign`, `crypto_sign_open`) mittels der Methode `cpucycles()` ermittelt, die in der Datei `test_extras.c` definiert ist. Nach der Ausführung der Methoden wird erneut die Cycle-Anzahl gemessen. Daraufhin wird die Differenz der beiden Werte berechnet und pro Durchlauf zum vorherigen Ergebnis addiert. Die Summe wird abschließend durch die Anzahl der durchlaufenen Tests (100) dividiert und ausgegeben.

```
// Generate the public/private keypair
cycles1 = cpucycles();
if ( (ret_val = crypto_sign_keypair(pk, sk)) != 0) {
    cycles2 = cpucycles();
    printf("crypto_sign_keypair returned <%d>\n", ret_val);
    return KAT_CRYPTTO_FAILURE;
}
else {
    cycles2 = cpucycles();
}
gencycles = gencycles+(cycles2-cycles1);
fprintf(fp_rsp, "pk = ", pk, CRYPTO_PUBLICKEYBYTES);
fprintf(fp_rsp, "sk = ", sk, CRYPTO_SECRETKEYBYTES);

// Generate signature
cycles1 = cpucycles();
if ( (ret_val = crypto_sign(sm, &smlen, m, mlen, sk)) != 0) {
    cycles2 = cpucycles();
    printf("crypto_sign returned <%d>\n", ret_val);
    return KAT_CRYPTTO_FAILURE;
}
else {
    cycles2 = cpucycles();
}
sigcycles = sigcycles+(cycles2-cycles1);
fprintf(fp_rsp, "smlen = %llu\n", smlen);
fprintf(fp_rsp, "sm = ", sm, smlen);
fprintf(fp_rsp, "\n");

// Verify signature
cycles1 = cpucycles();
if ( (ret_val = crypto_sign_open(m1, &mlen1, sm, smlen, pk)) != 0) {
    cycles2 = cpucycles();
    printf("crypto_sign_open returned <%d>\n", ret_val);
    return KAT_CRYPTTO_FAILURE;
}
else {
```

```
        cycles2 = cpucycles();
    }
    vercycles = vercycles+(cycles2-cycles1);

    ...

    printf("  KeyGen runs in ..... %10lld ",
           gencycles/100);
    printf("\n");
    printf("  Sign runs in ..... %10lld ",
           sigcycles/100);
    printf("\n");
    printf("  Verify runs in ..... %10lld ",
           vercycles/100);
    printf("\n");
```

#### A.4 SPHINCS+: Auszug PQCgenKAT\_sign.c

Der aktuelle Stand der CPU-Zyklen wird vor der Ausführung der jeweiligen Methode zur Schlüsselerzeugung, Signaturerzeugung und Signaturprüfung (`crypto_sign_keypair`, `crypto_sign`, `crypto_sign_open`) mittels der Methode `cpucycles()` ermittelt, die in der Datei `test_extras.c` definiert ist. Nach der Ausführung der Methoden wird erneut die Cycle-Anzahl gemessen. Daraufhin wird die Differenz der beiden Werte berechnet und pro Durchlauf zum vorherigen Ergebnis addiert. Die Summe wird abschließend durch die Anzahl der durchlaufenen Tests (100) dividiert und ausgegeben.

```
// Generate the public/private keypair
cycles1 = cpucycles();
if ( (ret_val = crypto_sign_keypair(pk, sk)) != 0) {
    cycles2 = cpucycles();
    printf("crypto_sign_keypair returned <%d>\n", ret_val);
    return KAT_CRYPTOFailure;
}
else {
    cycles2 = cpucycles();
}
gencycles = gencycles+(cycles2-cycles1);
fprintf(fp_rsp, "pk = ", pk, CRYPTO_PUBLICKEYBYTES);
fprintf(fp_rsp, "sk = ", sk, CRYPTO_SECRETKEYBYTES);

// Generate signature
cycles1 = cpucycles();
if ( (ret_val = crypto_sign(sm, &smlen, m, mlen, sk)) != 0) {
    cycles2 = cpucycles();
    printf("crypto_sign returned <%d>\n", ret_val);
    return KAT_CRYPTOFailure;
}
else {
    cycles2 = cpucycles();
}
sigcycles = sigcycles+(cycles2-cycles1);
fprintf(fp_rsp, "smlen = %llu\n", smlen);
fprintf(fp_rsp, "sm = ", sm, smlen);
fprintf(fp_rsp, "\n");

// Verify signature
cycles1 = cpucycles();
if ( (ret_val = crypto_sign_open(m1, &mlen1, sm, smlen, pk)) != 0) {
    cycles2 = cpucycles();
    printf("crypto_sign_open returned <%d>\n", ret_val);
    return KAT_CRYPTOFailure;
}
else {
```

```
        cycles2 = cpucycles();
    }
    vercycles = vercycles+(cycles2-cycles1);

    ...

} while ( !done );

fclose(fp_req);
fclose(fp_rsp);

printf("  KeyGen runs in ..... %10lld ",
gencycles/100);
printf("\n");
printf("  Sign runs in ..... %10lld ",
sigcycles/100);
printf("\n");
printf("  Verify runs in ..... %10lld ",
vercycles/100);
printf("\n");
```

## A.5 McEliece-KEM: Auszug kat\_kem.c

Der aktuelle Stand der CPU-Zyklen wird vor der Ausführung der jeweiligen Methode zur Schlüsselerzeugung, Encapsulation und Decapsulation (`crypto_kem_keypair`, `crypto_kem_enc`, `crypto_kem_dec`) mittels der Methode `cpucycles()` ermittelt, die in der Datei `test_extras.c` definiert ist. Nach der Ausführung der Methoden wird erneut die Cycle-Anzahl gemessen. Daraufhin wird die Differenz der beiden Werte berechnet und pro Durchlauf zum vorherigen Ergebnis addiert. Die Summe wird abschließend durch die Anzahl der durchlaufenen Tests (100) dividiert und ausgegeben.

```
// Generate the public/private keypair
cycles1 = cpucycles();
if ( (ret_val = crypto_kem_keypair(pk, sk)) != 0) {
    cycles2 = cpucycles();
    fprintf(stderr, "crypto_kem_keypair returned <%d>\n", ret_val);
    return KAT_CRYPTOP_FAILURE;
}
else {
    cycles2 = cpucycles();
}
gencycles = gencycles+(cycles2-cycles1);
fprintfBstr(fp_rsp, "pk = ", pk, crypto_kem_PUBLICKEYBYTES);
fprintfBstr(fp_rsp, "sk = ", sk, crypto_kem_SECRETKEYBYTES);

// Encapsulation
cycles1 = cpucycles();
if ( (ret_val = crypto_kem_enc(ct, ss, pk)) != 0) {
    cycles2 = cpucycles();
    fprintf(stderr, "crypto_kem_enc returned <%d>\n", ret_val);
    return KAT_CRYPTOP_FAILURE;
}
else {
    cycles2 = cpucycles();
}
enccycles = enccycles+(cycles2-cycles1);
fprintfBstr(fp_rsp, "ct = ", ct, crypto_kem_CIPHERTEXTBYTES);
fprintfBstr(fp_rsp, "ss = ", ss, crypto_kem_BYTES);
fprintf(fp_rsp, "\n");

// Decapsulation
cycles1 = cpucycles();
if ( (ret_val = crypto_kem_dec(ss1, ct, sk)) != 0) {
    cycles2 = cpucycles();
    fprintf(stderr, "crypto_kem_dec returned <%d>\n", ret_val);
    return KAT_CRYPTOP_FAILURE;
}
else {
```

```
        cycles2 = cpucycles();
    }
    deccycles = deccycles+(cycles2-cycles1);

    if ( memcmp(ss, ss1, crypto_kem_BYTES) ) {
        fprintf(stderr, "crypto_kem_dec returned bad 'ss' value\n");
        return KAT_CRYPTO_FAILURE;
    }
}

fprintf( stderr, " KeyGen runs in .....
%10lld \n", gencycles/100);
fprintf( stderr, " Encaps runs in .....
%10lld \n", encycles/100);
fprintf( stderr, " Decaps runs in .....
%10lld \n", deccycles/100);
```

## A.6 FrodoKEM: Auszug test\_kem.c

Der aktuelle Stand der CPU-Zyklen wird vor der Ausführung der jeweiligen Methode zur Schlüsselerzeugung, Encapsulation und Decapsulation (`crypto_kem_keypair`, `crypto_kem_enc`, `crypto_kem_dec`) mittels der Methode `cpucycles()` ermittelt, die in der Datei `test_extras.c` definiert ist. Nach der Ausführung der Methoden wird erneut die Cycle-Anzahl gemessen. Daraufhin wird die Differenz der beiden Werte berechnet und pro Durchlauf zum vorherigen Ergebnis addiert. Die Summe wird abschließend durch die Anzahl der durchlaufenen Tests (100) dividiert und ausgegeben.

```

for (int i = 0; i < iterations; i++) {
// Generate the public/private keypair
    cycles1 = cpucycles();
    crypto_kem_keypair(pk, sk);
    cycles2 = cpucycles();
    gencycles = gencycles+(cycles2-cycles1);

// Encapsulation
    cycles1 = cpucycles();
    crypto_kem_enc(ct, ss_encap, pk);
    cycles2 = cpucycles();
    enccycles = enccycles+(cycles2-cycles1);

// Decapsulation
    cycles1 = cpucycles();
    crypto_kem_dec(ss_decap, ct, sk);
    cycles2 = cpucycles();
    deccycles = deccycles+(cycles2-cycles1);

    if (memcmp(ss_encap, ss_decap, CRYPTO_BYTES) != 0) {
        printf("\n ERROR!\n");
        return false;
    }
}

printf(" KeyGen runs in ..... %10lld ",
gencycles/100);
printf("\n");
printf(" Encaps runs in ..... %10lld ",
encycles/100);
printf("\n");
printf(" Decaps runs in ..... %10lld ",
deccycles/100);
printf("\n");

```

## A.7 NewHope: Auszug PQCgenKAT\_kem.c

Der aktuelle Stand der CPU-Zyklen wird vor der Ausführung der jeweiligen Methode zur Schlüsselerzeugung, Encapsulation und Decapsulation (`crypto_kem_keypair`, `crypto_kem_enc`, `crypto_kem_dec`) mittels der Methode `cpucycles()` ermittelt, die in der Datei `test_extras.c` definiert ist. Nach der Ausführung der Methoden wird erneut die Cycle-Anzahl gemessen. Daraufhin wird die Differenz der beiden Werte berechnet und pro Durchlauf zum vorherigen Ergebnis addiert. Die Summe wird abschließend durch die Anzahl der durchlaufenen Tests (100) dividiert und ausgegeben.

```
// Generate the public/private keypair
cycles1 = cpucycles();
if ( (ret_val = crypto_kem_keypair(pk, sk)) != 0) {
    cycles2 = cpucycles();
    printf("crypto_kem_keypair returned <%=d>\n", ret_val);
    return KAT_CRYPT0_FAILURE;
}
else {
    cycles2 = cpucycles();
}
gencycles = gencycles+(cycles2-cycles1);
fprintfBstr(fp_rsp, "pk = ", pk, CRYPTO_PUBLICKEYBYTES);
fprintfBstr(fp_rsp, "sk = ", sk, CRYPTO_SECRETKEYBYTES);

// Encapsulation
cycles1 = cpucycles();
if ( (ret_val = crypto_kem_enc(ct, ss, pk)) != 0) {
    cycles2 = cpucycles();
    printf("crypto_kem_enc returned <%=d>\n", ret_val);
    return KAT_CRYPT0_FAILURE;
}
else {
    cycles2 = cpucycles();
}
enccycles = enccycles+(cycles2-cycles1);
fprintfBstr(fp_rsp, "ct = ", ct, CRYPTO_CIPHERTEXTBYTES);
fprintfBstr(fp_rsp, "ss = ", ss, CRYPTO_BYTES);
fprintf(fp_rsp, "\n");

// Decapsulation
cycles1 = cpucycles();
if ( (ret_val = crypto_kem_dec(ss1, ct, sk)) != 0) {
    cycles2 = cpucycles();
    printf("crypto_kem_dec returned <%=d>\n", ret_val);
    return KAT_CRYPT0_FAILURE;
}
else {
```



---

```
        cycles2 = cpucycles();
    }
    deccycles = deccycles+(cycles2-cycles1);

    if ( memcmp(ss, ss1, CRYPTO_BYTES) ) {
        printf("crypto_kem_dec returned bad 'ss' value\n");
        return KAT_CRYPTO_FAILURE;
    }

} while ( !done );

fclose(fp_req);
fclose(fp_rsp);

printf("  KeyGen runs in ..... %10lld ",
gencycles/100);
printf("\n");
printf("  Encaps runs in ..... %10lld ",
enccycles/100);
printf("\n");
printf("  Decaps runs in ..... %10lld ",
deccycles/100);
printf("\n");
```

## A.8 SIKE: Auszug test\_sike.c

Der aktuelle Stand der CPU-Zyklen wird vor der Ausführung der jeweiligen Methode zur Schlüsselerzeugung, Encapsulation und Decapsulation (`crypto_kem_keypair_generic`, `crypto_kem_enc_generic`, `crypto_kem_dec_generic`) mittels der Methode `cpucycles()` ermittelt, die in der Datei `test_extras.c` explizit für den SIKE-Algorithmus definiert ist und als Referenz für die anderen Algorithmen herangezogen wurde. Nach der Ausführung der Methoden wird erneut die Cycle-Anzahl gemessen. Daraufhin wird die Differenz der beiden Werte berechnet und pro Durchlauf zum vorherigen Ergebnis addiert. Die Summe wird abschließend durch die Anzahl der durchlaufenen Tests (100) dividiert und ausgegeben.

```
// Generate the public/private keypair
cycles = 0;
for (i = 0; i < runs; ++i) {
    cycles1 = cpucycles();
    crypto_kem_keypair_generic(params, pk3, sk3);
    cycles2 = cpucycles();

    cycles = cycles + (cycles2 - cycles1);
}
printf(" Key generation runs in .....
%10ld ", (cycles / runs));
print_unit

// Encapsulation
cycles = 0;
for (i = 0; i < runs; ++i) {
    cycles1 = cpucycles();
    rc = crypto_kem_enc_generic(params, ct, ss, pk3);
    cycles2 = cpucycles();
    if ( rc ) goto end;

    cycles = cycles + (cycles2 - cycles1);
}
printf(" Encapsulation runs in .....
%10ld ", (cycles / runs));
print_unit

// Decapsulation
cycles = 0;
for (i = 0; i < runs; ++i) {
    cycles1 = cpucycles();
    rc = crypto_kem_dec_generic(params, ss_rec, ct, sk3);
    cycles2 = cpucycles();
    if ( rc ) goto end;
```

```
        cycles = cycles + (cycles2 - cycles1);
    }
    printf(" Decapsulation runs in .....
%10ld ", (cycles / runs));
    print_unit
```

**A.9 Beispiel Request- und Response-Datei eines SPHINCS+-Testlaufs**

```
count = 0
seed =
061550234D158C5EC95595FE04EF7A25767F2E24CC2BC479D09D86DC9ABCFDE7056A
8C266F9EF97ED08541DBD2E1FFA1
mlen = 33
msg =
D81C4D8D734FCBFBEADE3D3F8A039FAA2A2C9957E835AD55B22E75BF57BB556AC8
pk =
sk =
smlen =
sm =
```

**Auszug aus der Request-Datei eines SPHINCS+-Testlaufs**

```
count = 0
seed =
061550234D158C5EC95595FE04EF7A25767F2E24CC2BC479D09D86DC9ABCFDE7056A
8C266F9EF97ED08541DBD2E1FFA1
mlen = 33
msg =
D81C4D8D734FCBFBEADE3D3F8A039FAA2A2C9957E835AD55B22E75BF57BB556AC8
pk =
3E784CCB7EBCDCFD45542B7F6AF778742E0F4479175084AA488B3B74340678AAFA7E
03FED35725D5BF1837BE21D387785F0F63B3E76C703C0C243F42386F5BF7
sk =
7C9935A0B07694AA0C6D10E4DB6B1ADD2FD81A25CCB148032DCD739936737F2DB505
D7CFAD1B497499323C8686325E4792F267AAFA3F87CA60D01CB54F29202A3E784CCB
7EBCDCFD45542B7F6AF778742E0F4479175084AA488B3B74340678AAFA7E03FED357
25D5BF1837BE21D387785F0F63B3E76C703C0C243F42386F5BF7
smlen = 29825
sm = 60F74DB4F30...
```

**Auszug aus der Response-Datei eines SPHINCS+-Testlaufs**

## A.10 Einzelergebnisse der Performancemessungen

Tabelle 23: Einzelergebnisse Performancemessung McEliece-PKE (vgl. A.11 [3.2.1.1])

Laufnr.	KeyGen	Encrypt	Decrypt
1	112.853.197.819	99.331.998.751	1.765.636.086.378
2	103.812.615.373	107.237.086.777	1.763.318.575.303
3	129.666.393.773	93.710.597.583	1.648.795.277.412
4	134.903.145.239	95.154.952.170	1.719.408.294.635
5	92.298.573.177	97.625.586.731	1.678.032.971.855
6	126.336.254.861	106.131.583.271	1.791.554.235.263
7	107.483.343.134	104.902.426.979	1.945.473.909.418
8	105.020.928.092	106.794.631.487	1.820.739.784.333
9	112.449.391.130	106.125.261.531	1.924.162.956.336
10	108.782.979.526	94.767.526.157	1.696.305.821.772
<b>Avg</b>	<b>113.360.682.212</b>	<b>101.178.165.144</b>	<b>1.775.342.791.271</b>

Tabelle 24: Einzelergebnisse Performancemessung NTRUEncrypt-PKE (vgl. A.11 [3.2.2.1])

Laufnr.	KeyGen	Encrypt	Decrypt
1	22.134.786.236	585.421.638.529	908.190.150.696
2	22.855.600.875	931.464.709.277	981.269.103.403
3	22.245.407.763	579.867.867.503	875.340.066.830
4	23.308.412.676	583.064.918.877	841.568.193.430
5	21.840.684.702	633.778.818.990	842.642.262.530
6	22.097.392.914	570.139.683.875	824.950.040.711
7	22.151.568.032	580.337.631.798	898.087.849.075
8	23.185.073.336	580.404.072.964	896.948.092.113
9	25.383.998.620	591.556.237.879	928.751.951.304
10	21.962.909.679	671.238.510.289	894.477.519.732
<b>Avg</b>	<b>22.716.583.483</b>	<b>630.727.408.998</b>	<b>889.222.522.982</b>

Tabelle 25: Einzelergebnisse Performancemessung Rainbow (vgl. A.11 [3.3.1.1])

Laufnr.	KeyGen	Sign	Verify
1	1.125.732.290	8.193.625	7.197.486
2	1.044.169.075	6.791.513	7.679.331
3	1.016.362.240	6.621.135	7.202.137
4	1.038.466.995	7.189.194	7.802.506
5	1.076.677.923	7.283.955	7.725.859
6	1.117.123.233	6.984.680	8.191.464
7	1.011.588.331	6.845.818	7.480.156
8	1.033.053.012	7.378.961	7.637.760
9	1.001.618.181	6.409.090	7.280.414
10	996.217.192	6.310.364	7.661.670
<b>Avg</b>	<b>1.046.100.847</b>	<b>7.000.834</b>	<b>7.585.878</b>

Tabelle 26: Einzelergebnisse Performancemessung SPHINCS+ (vgl. A.11 [3.3.2.1])

Laufnr.	KeyGen	Sign	Verify
1	386.857.016	4.805.342.266	6.410.972
2	342.757.680	4.151.773.081	6.058.298
3	378.099.896	4.695.210.787	6.543.057
4	337.698.887	4.071.770.711	5.633.840
5	341.128.110	3.867.079.370	6.279.475
6	352.203.197	4.046.881.530	6.559.674
7	336.765.972	3.971.717.839	6.482.583
8	358.682.229	4.177.345.715	5.765.854
9	375.007.677	4.838.801.057	6.060.477
10	327.389.916	3.961.872.289	5.483.624
<b>Avg</b>	<b>353.659.058</b>	<b>4.258.779.465</b>	<b>6.127.785</b>

Tabelle 27: Einzelergebnisse Performancemessung McEliece-KEM (vgl. A.11 [3.4.1.1])

Laufnr.	KeyGen	Encaps	Decaps
1	909.403.110	827.315	154.618.444
2	874.213.760	963.558	165.820.057
3	904.146.124	889.804	149.852.902
4	972.549.171	980.736	179.192.576
5	845.339.033	921.856	157.793.920
6	843.403.289	818.048	147.673.753
7	857.700.966	964.761	157.440.076
8	864.588.569	898.585	156.737.766
9	1.031.410.176	1.087.206	201.826.739
10	874.932.121	1.015.116	169.449.574
<b>Avg</b>	<b>897.768.632</b>	<b>936.699</b>	<b>164.040.581</b>

Tabelle 28: Einzelergebnisse Performancemessung FrodoKEM (vgl. A.11 [3.4.2.1])

Laufnr.	KeyGen	Encaps	Decaps
1	109.662.880	495.899.016	503.788.684
2	112.876.497	503.551.594	505.563.268
3	105.877.327	454.587.209	466.638.734
4	102.209.095	461.504.222	477.704.194
5	97.047.837	474.270.867	465.089.653
6	99.564.839	440.667.559	455.697.155
7	92.288.125	437.107.626	453.410.844
8	99.817.191	454.141.354	446.275.656
9	92.805.629	443.994.716	452.413.822
10	89.566.564	441.608.472	443.454.036
<b>Avg</b>	<b>100.171.598</b>	<b>460.733.264</b>	<b>467.003.605</b>

Tabelle 29: Einzelergebnisse Performancemessung NewHope (vgl. A.11 [3.4.3.1])

Laufnr.	KeyGen	Encaps	Decaps
1	650.096	2.191.327	1.311.169
2	529.770	1.136.340	1.000.412
3	530.374	1.538.449	1.274.756
4	471.894	1.954.690	2.105.468
5	596.408	821.176	1.580.239
6	495.072	1.731.148	1.310.806
7	798.163	1.563.579	1.142.573
8	652.019	966.630	1.016.662
9	529.700	1.663.499	932.827
10	479.425	1.740.009	1.140.411
<b>Avg</b>	<b>525.350</b>	<b>1.356.684</b>	<b>1.167.491</b>

Tabelle 30: Einzelergebnisse Performancemessung SIKE (vgl. A.11 [3.4.4.1])

Laufnr.	KeyGen	Encaps	Decaps
1	7.440.069.758	9.190.694.011	11.863.702.697
2	7.272.418.624	9.806.322.043	11.633.435.804
3	6.093.746.642	8.328.837.592	10.673.805.623
4	6.725.321.993	10.055.559.153	12.205.972.039
5	7.237.971.928	9.748.107.039	12.025.050.581
6	9.266.473.603	10.500.618.549	11.587.421.418
7	6.717.809.690	9.114.127.404	12.169.296.845
8	7.547.356.204	9.881.128.374	11.140.874.836
9	5.998.529.443	8.271.863.140	10.130.785.236
10	6.576.525.498	12.463.584.393	13.891.119.550
<b>Avg</b>	<b>7.087.622.338</b>	<b>9.736.084.170</b>	<b>11.732.146.463</b>



## A.11 Anlagen USB-Stick

Neben der Masterthesis in PDF- und Wordformat werden folgende Anlagen gemäß der erläuterten Ordnerstruktur dieser Arbeit beigefügt.

**Tabelle 31: Ordnerstruktur und Dateien des USB-Sticks**

Ebene 1	Ebene 2	Ebene 3	Ebene 4	Inhalt
[1] Abbildungen				Verwendete Abbildungen
[2] Literaturquellen				Verwendete Quellen
[3] Evaluationsdokumentation	[3.1] Cyclemessung			test_extras.c test_extras.h Übersicht_Einzelergebnisse.xlsx
	[3.2] PKE	[3.2.1] McEliece	[3.2.1.1] Ergebnisse_Performancemessung	McEliece_Durchläufe.txt
			[3.2.1.2] Modifizierter_Quellcode	mceliece.py
			[3.2.1.3] Schlüssel	pk.npz sk.npz
			[3.2.1.4] Zusatzdateien	Encrypted_Testfile.txt Testfile.txt Testrun.sh
		[3.2.2] NTRUEncrypt	[3.2.2.1] Ergebnisse_Performancemessung	NTRUEncrypt_Durchläufe.txt
			[3.2.2.2] Modifizierter_Quellcode	ntru.py
			[3.2.2.3] Schlüssel	pk.npz sk.npz
			[3.2.2.4] Zusatzdateien	Encrypted_Testfile.txt Testfile.txt Testrun.sh
	[3.3] Signaturerstellungsverfahren	[3.3.1] Rainbow	[3.3.1.1] Ergebnisse_Performancemessung	Rainbow_Durchläufe.txt
			[3.3.1.2] Modifizierter_Quellcode	PQCgenKAT_sign.c
			[3.3.1.3] Zusatzdateien	PQCsignKAT_1227104.req PQCsignKAT_1227104.rsp
		[3.3.2] SPHINCS+	[3.3.2.1] Ergebnisse_Performancemessung	SPHINCS_Durchläufe.txt
			[3.3.2.2] Modifizierter_Quellcode	PQCgenKAT_sign.c
			[3.3.2.3] Zusatzdateien	PQCsignKAT_128.req PQCsignKAT_128.rsp
	[3.4] KEM	[3.4.1] McEliece	[3.4.1.1] Ergebnisse_Performancemessung	McEliece_Durchläufe.txt
			[3.4.1.2] Modifizierter_Quellcode	kat_kem.c
[3.4.2] Frodo		[3.4.2.1] Ergebnisse_Performancemessung	Frodo_Durchläufe.txt	

---

			[3.4.2.2] Modifizierter_Quellcode	test_kem.c
		[3.4.3] NewHope	[3.4.3.1] Ergebnisse_Performancemessung	NewHope_Durchläufe.txt
			[3.4.3.2] Modifizierter_Quellcode	PQCgenKAT_kem.c
		[3.4.4] SIKE	[3.4.4.1] Ergebnisse_Performancemessung	SIKE_Durchläufe.txt
			[3.4.4.2] Modifizierter_Quellcode	test_sike.c

**10 Verzeichnis der Abkürzungen**

AES	Advanced Encryption Standard
AG-Code	Algebraic-Geometric-Code
BCH-Code	Bose-Chaudhuri-Hocquenghem-Code
BSI	Bundesamt für Sicherheit in der Informationstechnik
CVP	Closest Vector Problem
DES	Data Encryption Standard
DFT	Diskrete Fouriertransformation
DSA	Digital Signature Algorithm
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
FTS	Few-Time-Signatures
GB	GigaByte
GGH	Goldreich-Goldwasser-Halevi
GHz	Gigahertz
GPT	Gabidulin-Paramonov-Trejtakov
HFE	Hidden Field Equations
HMAC	Hash-based Message Authentication Code
HORST	Hash to Obtain Random Subset Tree
ISO	International Standards Organization
kB	kiloByte
KEM	Key Encapsulation Mechanism
LBC	Lattice-based cryptography
LDPC-Code	Low-Density Parity-Check-Code
LWE	Learning with Errors
MAC	Message Authentication Code
MQ-Problem	Multivariate-Quadratische-Problem
MSS	Merkle Signature Scheme
NIST	National Institute of Standards
OTS	One-Time-Signatures
PBKDF2	Password-Based Key Derivation Function 2
PGP	Pretty-Good-Privacy

PKE	Public-Key-Encryption
POC	Proof of Concept
PRNG	Pseudo-Random Number Generator
Qubit	Quantenbit
RS-Code	Reed-Solomon-Code
RSA-Algorithmus	Rivest-Shamir-Adleman-Algorithmus
SHA	Secure Hash Algorithm
SIDH	Supersingular Isogeny Diffie-Hellman
SIKE	Supersingular Isogeny Key Encapsulation
SSL	Secure Socket Layer
SSH	Secure Shell
SVP	Shortest Vector Problem
XMSS	eXtended Merkle Signature Scheme

## 11 Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die hier vorliegende Arbeit selbstständig, ohne unerlaubte fremde Hilfe und nur unter Verwendung der in der Arbeit aufgeführten Hilfsmittel angefertigt habe.

Unterelldorf, 10. Juli 2020

Ort, Datum

\_\_\_\_\_

(Unterschrift)

## 12 Thesen

1. Die Entwicklung von Quantencomputern nimmt rasant an Geschwindigkeit zu und die daraus resultierende Bedrohung für klassische Verschlüsselungsalgorithmen steigt.
2. Durch die Verwendung von ausreichend großen Schlüssellängen geht das BSI aktuell noch von einer ausreichend hohen Sicherheit der klassischen Kryptographie-Verfahren aus.
3. Verschlüsselte Dokumente und Daten, die heute übertragen werden, können aufgezeichnet werden und im Quantenzeitalter schnell durch Quantenalgorithmen entschlüsselt werden.
4. Die Notwendigkeit, Quantencomputer-resistente Public-Key-Algorithmen auf klassischen Systemen zu implementieren, steigt.
5. Es gibt nicht die beste Post-Quantum Kryptographie-Kategorie oder den besten Quantencomputer-resistenten Public-Key-Algorithmus.
6. Abhängig von Fokus und Anwendungsbereich kann ein anderer Quantencomputer-resistenter Public-Key-Algorithmus am geeignetsten sein.
7. Die Post-Quantum Kryptographie sowie deren Kryptoanalyse ist noch nicht ausreichend erforscht und bietet weiterhin Entwicklungspotential.
8. Hybride Kryptosysteme aus klassischen und Quantencomputer-resistenten Verfahren stellen vorerst eine gute Lösung dar, um sich gegen klassische Angriffsmethoden und gegen Quantenalgorithmen abzusichern.