Freie Universität Berlin

Masterarbeit am Institut für Informatik der Freien Universität Berlin

Marian Sigler

sigler@inf.fu-berlin.de

Entwurf eines hybriden Verschlüsselungsverfahrens aus einem klassischen und einem Post-Quanten-Verfahren

Berlin, 14. Mai 2019

Betreuer

Benjamin Zengin, M.Sc.

Prof. Dr. Marian Margraf

Gutachter

Prof. Dr. Marian Margraf

Prof. Dr.-Ing. Jochen Schiller

Zusammenfassung

Die vorraussichtlich bevorstehende Entwicklung von großen Quantencomputern macht herkömmliche asymmetrische Verschlüsselungsverfahren wie RSA und Diffie-Hellman unsicher. Es sollte daher zeitnah auf quantencomputerresistente Verfahren, sogenannte Post-Quanten-Verfahren, umgestiegen werden. In dieser Arbeit werden einige solche Verfahren vorgestellt und verglichen.

Um trotz Bedenken hinsichtlich ihrer Sicherheit schnellstmöglich auf Post-Quanten-Kryptografie umsteigen zu können, können diese neuen Verfahren mit einem etablierten Verfahren zu einem Hybrid-Verfahren kombiniert werden. Es werden verschiedene Arten der Umsetzung eines solchen Verfahrens verglichen und zwei Vorzugsvarianten konkret definiert.

Änderungsverzeichnis

Zukünftige Aktualisierungen dieser Arbeit werde ich unter qjym.de/uni/arbeiten/master/ zur Verfügung stellen. Dort finden sich auch die Folien der Verteidigung.

Diese Version

- Falsche Werte für RSA in Tabelle 4.6 korrigiert
- Verdeutlicht, dass das Resultat der Transformation $U_m^{\perp}$ (§ 5.2.3) ein CCA2-sicheres Verfahren ist

14. Mai 2019

Abgabeversion

Inhaltsverzeichnis

1 Einleitung

Quantencomputer sind Computer, die Informationen nicht in Bits speichern wie klassische Computer, sondern in sogenannten Quantenbits (auch *Qubits*). Der Zustand eines Quantenbits ist nicht entweder 0 oder 1, sondern eine Überlagerung beider Zustände [HV09]. Es sei an das berühmte Gedankenexperiment von Erwin Schrödinger erinnert, in dem er eine Katze postulierte, die gleichzeitig tot und lebendig sei. Hat ein Quantencomputer n Qubits, entsteht ein großer Zustandsraum, der eine Überlagerung von 2^n Zuständen speichern kann.

Ein Quantencomputer kann die Informationen nicht nur speichern, sondern auch damit rechnen. Die Rechenoperationen sind nicht so frei definierbar wie bei klassischen Computern, sondern es gibt einige Einschränkungen (z. B. Reversibilität). Jedoch kann nicht nur mit *einem* Zustand gerechnet werden, sondern der gesamte Überlagerungszustand kann umgeformt werden.

Man kann sich dies angenähert so vorstellen, dass sich ein Quantencomputer ein Stück weit wie eine nichtdeterministische Turingmaschine verhält. Man darf diesen Vergleich jedoch nicht überstrapazieren: Wäre ein Quantencomputer eine solche, gälte für Quantencomputer $\mathsf{P} = \mathsf{NP}$ und Kryptografie wäre (in einer vereinfachten Grenzwert-Betrachtung) allgemein unmöglich, vom One-Time-Pad abgesehen.

Der Shor-Algorithmus Peter Shor entwickelte 1994 den nach ihm benannten Algorithmus [Sho94; NW17], mit dem Primfaktorzerlegung und die Berechnung des diskreten Logarithmus auf einem Quantencomputer effizient gelöst werden können. Beide Probleme werden für klassische Computer als schwer (d. h. $\notin \mathsf{P}$) angenommen.

Der dlog diskrete Logarithmus ist die Aufgabe, zu einer Gleichung $g^r \equiv x \pmod{p}$ für gegebene g, x und p den (ganzzahligen) Exponenten r zu finden [Sho94]. Während die Exponentialfunktion im Bereich der reellen Zahlen (also ohne Modulus) stetig ist, wodurch der Logarithmus leicht durch Annäherung gefunden werden kann, „springen“ im diskreten Fall die Werte aufgrund des Modulus, und es scheint (klassisch) keine effiziente Berechnungsmethode zu existieren.

Der Grover-Algorithmus Von Lov Grover stammt ein Algorithmus [Gro96; NW17], mit dem die Komplexität von Suchproblemen auf ungeordneten Daten auf die Wurzel der klassischen Komplexität reduziert werden kann (anders gesagt: eine Suche, die klassisch die Komplexität $\mathcal{O}(2^N)$ hat, ist mit dem Grover-Algorithmus in $\mathcal{O}(2^{N/2})$ zu berechnen).

Die Lösung des Suchproblems schließt insbesondere die Berechnung einer Umkehrfunktion ein, wenn die Originalfunktion in einem Quantencomputer berechnet werden kann [GLRS16].

Qubits Die Anzahl der Qubits eines Quantencomputers ist eine wichtige Größe. Wegen der Zustandsüberlagerung können Berechnungen über große Daten nicht in Segmente aufgeteilt und hintereinander ausgeführt werden; der Quantencomputer muss daher über einen ausreichend großen, „zusammenhängenden" Zustandsraum verfügen.

Daher sind heute, obwohl es schon (relativ kleine) Quantencomputer gibt, bisher bekannterweise keine Angriffe auf tatsächlich verwendete kryptografische Protokolle möglich; es wird deshalb oft (ungenau) davon gesprochen, dass es „in Zukunft" Quantencomputer gebe – gemeint ist, dass es noch keine Quantencomputer gibt, die groß genug sind, um eine Gefahr für die heutigen Protokolle darzustellen.

Der bisher größte bekannte Quantencomputer ist *IonQ* mit 79 Qubits [Arm18]. Zum Vergleich: Um eine n-Bit-Zahl zu faktorisieren, sind etwa $2n$ Qubits nötig [RNSL17] – um heute typische RSA-Moduli von 2048 bit [BSI2102] zu brechen, braucht man also einen Quantencomputer mit etwa 4000 Qubit.

Heutige Kryptografie

Kryptografische Basisfunktionalitäten (sog. *Primitiven*) lassen sich in vier grundlegende Kategorien einteilen [NW17]:

- **symmetrische Chiffren** Hier wird zur Ver- und Entschlüsselung der selbe Schlüssel genutzt. Weit verbreitet ist zum Beispiel AES (Advanced Encryption Standard) [BSI2102].
- **asymmetrische Verfahren** Hier gibt es einen öffentlichen Schlüssel, mit dem verschlüsselt wird und der frei verteilt werden kann, und einen geheimen Schlüssel zur Entschlüsselung. Neben asymmetrischer *Verschlüsselung* zähle ich hierzu auch **Schlüsselaustausch**-Verfahren, bei denen die Nachricht keine Ein- sondern eine Ausgabe ist (siehe hierzu § 2.3.2, S. 10).

  Hier gibt es nur wenige etablierte Verfahren. Ihre Sicherheit beruht entweder auf der Schwierigkeit der Primfaktorzerlegung (beispielsweise RSA) oder des diskreten Logarithmus' (zum Beispiel das Diffie-Hellman-Verfahren).
- **Signaturverfahren** Auch hier gibt es einen öffentlichen und einen geheimen Schlüssel. Mit dem geheimen Schlüssel kann eine Art Prüfsumme (*Signatur*) über Daten erstellt werden und diese Daten so unterschrieben werden. Mit dem öffentlichen Schlüssel kann verifiziert werden, ob die Daten mittels des zugehörigen privaten Schlüssels signiert wurden. Alle bisher verbreiteten Verfahren basieren auf der Primfaktorzerlegung (zum Beispiel RSA, das auch als Signaturverfahren genutzt werden kann) oder dem diskreten Logarithmus (beispielsweise DSA).

- **Hashfunktionen** dienen dazu, aus einer Eingabe deterministisch einen zufällig erscheinenden Wert zu berechnen, aus dem nicht auf die Eingabe geschlossen werden kann[1]. Hashfunktionen sind ein wichtiger Baustein für fast alle kryptografischen Anwendungen.

Für symmetrische Chiffren und Hashfunktionen ist nur der Grover-Algorithmus eine Gefahr. Dieser kann relativ leicht begegnet werden, indem die Schlüssellänge verdoppelt wird.

Jedoch stellt der Shor-Algorithmus eine große Gefahr da. Stünde in nächster Zeit ein großer Quantencomputer zur Verfügung, könnten alle heute verbreiteten asymmetrischen und Signatur-Verfahren gebrochen werden. Auch die heute verbreitete Kryptografie über elliptische Kurven ist nicht davor sicher – hier werden die selben Probleme genutzt, lediglich über der Gruppe der elliptischen Kurven statt über Restklassen der natürlichen Zahlen.

Post-Quanten-Kryptografie

Es ist also nötig, neue Verfahren zu entwickeln, die nicht von Quantencomputern gebrochen werden können, sie zu standardisieren und auf sie umzusteigen. Für diese Verfahren haben sich die Begriffe *Post-Quanten-Kryptografie* (engl. *Post Quantum Cryptography*, *PQC*) sowie *quantencomputerresistent* etabliert [NIS16].

Für Verschlüsselung drängt die Zeit hier besonders, denn selbst wenn große Quantencomputer erst in vielen Jahren zur Verfügung stehen, könnte vorher stattfindende Kommunikation aufgezeichnet und dann entschlüsselt werden.

Für Signaturen drängt die Zeit im Allgemeinen nicht, denn hier sind Angriffe auf frühere Kommunikation nicht möglich. Es kann jedoch wünschenswert sein, bei langlebiger Software und vor allem Hardware bereits heute einen quantencomputerresistenten Update-Pfad zu integrieren. Es gibt hier mit hashbasierten Verfahren bereits sowohl sehr alte und beweisbar sichere Verfahren mit einigen praktischen Nachteilen, als auch jüngere, aber handlichere Verfahren (§ 3.3).

Am drängendsten ist also die Entwicklung, Erprobung und Standardisierung von asymmetrischen quantencomputerresistenten Verfahren.

Die Sicherheit kryptografischer Verfahren kann nur in wenigen Fällen mathematisch vollständig bewiesen werden, daher gewinnen Verfahren erst größeres Vertrauen, wenn längere Zeit trotz intensiver Suche keine Schwächen gefunden werden konnten. Viele Post-Quanten-Verfahren sind aber noch recht jung und wenig erforscht, der Umstieg auf Post-Quanten-Kryptografie sollte jedoch möglichst schnell erfolgen. Um trotzdem schon einen Umstieg zu ermöglichen, kann ein neues, quantencomputerresistentes Verfahren mit einem etablierten Verfahren zu einem Hybrid-Verfahren kombiniert werden.

[1] Dies ist nur eine von mehreren Anforderungen.

Aufgabenstellung

Ziel dieser Arbeit ist es, zu einer Beschleunigung des Umstiegs auf quantencomputerresistente Verschlüsselung beizutragen. Die Beiträge hierzu sind zweierlei:

Zunächst sollen verschiedene Post-Quanten-Verfahren ausgewählt werden, die bereits für eine praktische Umsetzung geeignet sind, also vor allem detailliert spezifiziert sind, sowie die hinreichend ausgereift sind, so dass ihrer Sicherheit genügend vertraut werden kann. Diese Verfahren sollen vorgestellt und miteinander verglichen werden.

In einem zweiten Teil soll ein konkretes Hybrid-Verfahren aus einem klassischen und einem PQ-Verfahren entwickelt werden, mit dem ein Umstieg trotz etwaiger Sicherheitsbedenken gegenüber den PQ-Verfahren möglich ist.

Gliederung

Die Arbeit ist wie folgt aufgeteilt: Im nächsten Kapitel findet sich eine kurze Zusammenfassung der Geschichte von (Post-Quanten-)Kryptografie sowie Hintergründe und Definitionen zur Kryptografie sowie ein Abschnitt zur (mathematischen) Notation.

In Kapitel 3 stelle ich die verschiedenen Klassen von PQ-Verfahren und ihre Hintergründe, insbesondere die zugrunde liegenden Probleme, vor.

Im darauffolgenden Kapitel werden die konkreten Verfahren im Detail vorgestellt und miteinander verglichen.

In Kapitel 5 wird die Entwicklung des Hybrid-Verfahrens beschrieben, es beginnt mit einer Variantendiskussion und spezifischen Hintergründen und schließt mit der konkreten Definition zweier Hybrid-Verfahren[2].

Die Arbeit endet mit einer Zusammenfassung und einem Ausblick in Kapitel 6.

[2] Entgegen des ursprünglichen Plans habe ich zwei Varianten umgesetzt. Die Gründe dafür sind in Kapitel 5 beschrieben.

2 Hintergrund

2.1 Geschichte

Die Geschichte der Kryptografie begann mit symmetrischen Verfahren, bei denen der selbe Schlüssel für die Ver- und Entschlüsselung genutzt wird. Sowohl alte, simple Verfahren wie die Cäsar-Verschlüsselung als auch recht moderne Verfahren wie die Enigma sind symmetrisch.

In einem grundlegenden Paper entwarfen Whitfield Diffie und Martin Hellman 1976 das Konzept der asymmetrischen Kryptografie [DH76]. Hierbei werden für zwei verschiedene, zusammengehörige Schlüssel verwendet: der öffentliche Schlüssel dient zur Verschlüsselung, der geheime zur Entschlüsselung. Ersterer kann frei verteilt werden, daher wird diese Klasse auch *Public Key Cryptography* genannt. Im selben Paper beschreiben sie auch gleich ein solches Verfahren, nämlich den bis heute genutzten Diffie-Hellman-Schlüsselaustausch.

Nur zwei Jahre später wurde das RSA-Verfahren publiziert [RSA78]. In den 80er-Jahren schließlich schlugen Victor Miller und Neal Koblitz unabhängig voneinander elliptische Kurven für die Nutzung in der Kryptografie vor [Mil86; Kob87]. Damit sind bereits die Grundlagen aller heute verbreiteter asymmetrischer Verfahren gelegt.

In den 90er-Jahren wurden die Algorithmen von Shor und Grover publiziert [Sho94; Gro96]. Um die Jahrtausendwende wurden die ersten Quantencomputer gebaut, die aber nur eine einstellige Zahl von Qubits besaßen, während in den letzten Jahren die Zahl der Qubits stark anstieg [Moo17; Arm18].

Erst gegen Ende der 00er-Jahre wurde die Forschung an Post-Quanten-Kryptografie verstärkt, die erste PQCRYPTO-Konferenz fand 2006 statt.

2.1.1 Der NIST-PQC-Standardisierungsprozess

Ende 2016 startete das *National Institute of Standards and Technology* der USA (NIST, eine dem Wirtschaftsministerium unterstellte Forschungs- und Standardisierungsbehörde) einen Standardisierungsprozess für PQ-Kryptografie [NIS16]. Das NIST hat bereits in der Vergangenheit solche Prozesse durchgeführt, zum Beispiel die, die zur Standardisierung des symmetrischen Verschlüsselungsverfahrens AES oder des Hashverfahrens SHA-3 führten; im Unterschied dazu ist bei diesem Prozess nicht die

Auswahl eines, sondern die Standardisierung mehrerer Verfahren geplant [Moo17], weshalb dieses Mal nicht von einem „Wettbewerb“ gesprochen wird.

Einreichungen waren bis Ende November 2017 möglich. Es wurden 59 Verschlüsselungs- und 23 Signaturverfahren eingereicht [Moo17]. In der Folgezeit wurden die Einreichungen sowohl vom NIST als auch von Forscher_innen begutachtet, dabei wurden in manchen der Verfahren Schwächen gefunden und diese zurückgezogen. Anfang 2019 gab das NIST bekannt, welche Verfahren in der zweiten Runde berücksichtigt werden, dies sind 17 Verschlüsselungs- und 9 Signaturverfahren [NIS19], darunter sind auch einige aus mehreren ähnlichen Verfahren zusammengeführte Einreichungen.

Kategorien 1, 3, 5

Das NIST hat fünf Sicherheitskategorien definiert, in die die Einreichungen einzuordnen sind. Dabei bedeutet Kategorie 1, dass das Verfahren mindestens so schwer zu brechen ist wie AES-128, Kategorie 3 entspricht AES-192 und Kategorie 5 AES-256. Die Kategorien 2 und 4 sind anhand von Hashfunktionen definiert und hier nicht relevant.

Das NIST akzeptiert Verschlüsselungsverfahren zweierlei Art: Zum einen Verschlüsselungsverfahren im engeren Sinne (KTMs, siehe unten: § 2.3.2), zum anderen Schlüsseleinigungsverfahren, bei denen das gemeinsame Geheimnis keine Ein- sondern eine Ausgabe darstellt (KEMs, ebd.). Von allen Verfahren wird verlangt, sicher gegen aktive Angriffe zu sein (IND-CCA, die Definition folgt in § 2.3.1)[1] [NIS16].

2.2 Abgrenzung zum Quanten-Schlüsselaustausch

Es gibt neben der hier behandelten Post-Quanten-Kryptografie noch eine andere Technik, die ähnlich klingt und deswegen verwechselt werden könnte; deshalb soll hier kurz der Unterschied erläutert werden.

Post-Quanten-Kryptografie ist der Oberbegriff für Verfahren, die gegen Angriffe durch Quantencomputer resistent sind, also auch in einer Welt, in der es Quantencomputer gibt, weiter sicher sind (daher der Name Post-Quanten). Alle diese Verfahren sind auf klassischen Computern ausführbar.

Der Quanten-Schlüsselaustausch (engl. *Quantum Key Distribution*, QKD, [BBD09, S. 13]) ist ein Verfahren, bei dem mittels Übertragung von Quantenzuständen, beispielsweise durch Photonen in einem Glasfaserkabel, ein geheimer Schlüssel beliebiger Länge mit einem Gesprächspartner geteilt werden kann.

Das Verfahren unterscheidet sich von allen etablierten Verfahren dadurch, dass die Sicherheit nicht auf mathematischen Problemen basiert, sondern auf quantenphysikalischen Gesetzen beruht. Es ist daher nicht *schwer* (im komplexitätstheoretischen

[1] In Spezialfällen genügt auch eine passive Sicherheit, dies ist im Folgenden nicht relevant.

Sinne), die Verschlüsselung zu brechen, sondern physikalisch unmöglich. Es ist folglich auch gegen Quantencomputer sicher.

Jedoch verlangt der Quanten-Schlüsselaustausch einerseits einen Ende-zu-Ende-Kanal zur Übertragung von Photonen (z. B. eine durchgehende Glasfaserleitung), andererseits wird zur Authentifizierung ein gemeinsames Geheimnis benötigt; es ist daher nur für eher kleinere, abgeschlossene Nutzergruppen geeignet und kann derzeit nicht in Netzen wie dem heutigen Internet eingesetzt werden.

2.3 Beweisbare Sicherheit

Um das Vertrauen in kryptografische Verfahren zu erhöhen, versucht man heute, deren Sicherheit mathematisch zu beweisen. Es ist jedoch i. d. R. nicht möglich, die Sicherheit absolut zu beweisen. Stattdessen wird versucht, die Sicherheit auf andere, bekannte Probleme zu reduzieren, also zu zeigen, dass, wenn das Verfahren gebrochen ist, das Problem effizient zu lösen ist. *Effizient* wird hier und im Folgenden im berechnungstheoretischen Sinne verwendet. Ein Algorithmus ist effizient, wenn die Laufzeit polynomiell in der Größe der Eingabe ist. Ansonsten ist er *schwer*.

Reduktion
effizient
schwer

Ein Beispiel ist das RSA-Verfahren. Es kann gezeigt werden, dass ein Angreifer, der aus einem öffentlichen RSA-Schlüssel den privaten Schlüssel berechnen kann, unter Zuhilfenahme dieser Berechnung auch beliebige Zahlen (gleicher Größenordnung) faktorisieren kann. Das Brechen eines RSA-Schlüssels[2] ist also mindestens so schwer wie die Primfaktorzerlegung. Diese Überführung eines Problems auf ein anderes wird *Reduktion* genannt.

Die Primfaktorzerlegung ist ein sehr altes und bekanntes Problem, und dennoch wurde hierfür bis heute kein effizientes Verfahren zu ihrer Lösung gefunden. Man kann also sehr sicher sein, dass es keines gibt. Dank des Reduktionsbeweises kann man mit der selben Gewissheit annehmen, dass es auch für die Berechnung eines geheimen RSA-Schlüssels keinen effizienten Algorithmus gibt.

2.3.1 Sicherheitsdefinitionen

In den vorigen Absätzen habe ich nur abstrakt von *Sicherheit* bzw. *gebrochen* gesprochen, ohne jedoch zu definieren, was damit genau gemeint ist. In diesem Abschnitt werde ich daher einige konkrete Definitionen für *Sicherheit* vorstellen. Die Definitionen unterscheiden sich zum Teil zwischen symmetrischen und asymmetrischen Verfahren; da der Fokus dieser Arbeit auf letzteren liegt, sind meist nur die für asymmetrische Verfahren gültigen Definitionen angegeben.

[2] Die angegebene Reduktion bezieht sich nur auf das Berechnen des geheimen Schlüssels. Zu einem Ciphertext den Klartext zu berechnen, ist möglicherweise leichter – es gibt jedenfalls keine entsprechende Reduktion.

Eine erste mögliche Definition ist die sogenannte Einwegeigenschaft (engl. *one-wayness*, OW). Ein Angreifer bricht sie, wenn er zu einem gegebenen Ciphertext den kompletten zugehörigen Klartext erraten kann. Offensichtlich ist diese Eigenschaft notwendig – ein Verfahren, dass sie nicht erfüllt, kann nicht als sicher gelten. Sie reicht aber für den praktischen Einsatz nicht aus: Auch ein Verfahren, bei dem ein Angreifer nur Teile des Klartexts erraten kann, gälte als sicher.

Naheliegend ist also, zu fordern, dass ein Angreifer aus einem Ciphertext *keinerlei* Informationen über den Klartext erfahren kann.[3] Diese Forderung nennt man **semantische Sicherheit**, sie wurde 1983 von Shafi Goldwasser und Silvio Micali in [GM84] definiert, jedoch in einer für praktische Beweise etwas unhandlichen Art.

IND-CPA

Eine zur semantischen Sicherheit äquivalente[4] [KL07, § 3.2.1] Definition ist die **Ununterscheidbarkeit im Chosen-Plaintext-Angriff** (engl. *indistinguishability under a chosen plaintext attack*, **IND-CPA**), die bedeutet: Ein Angreifer darf, selbst wenn er die Klartexte auswählen kann, die zugehörigen Ciphertexte nicht zuordnen können. Konkret wird IND-CPA für asymmetrische Verfahren[5] über folgendes *Spiel* definiert:

1. Die Umgebung erzeugt ein Schlüsselpaar $(\mathrm{sk}, \mathrm{pk})$, der Angreifer erhält pk.
2. Der Angreifer gibt zwei gleich lange Nachrichten m_0, m_1 zurück.
3. Die Umgebung wählt zufällig $b \xleftarrow{\$} \{0, 1\}$, und verschlüsselt m_b zu $c = \mathsf{enc}(m_b)$.
4. Der Angreifer erhält c und muss raten, welche seiner Nachrichten verschlüsselt wurde, er gibt seine Vermutung b' zurück.
5. Ist $b = b'$, war der Angriff erfolgreich.

Diese Definition kann ein deterministisches Verschlüsselungsverfahren prinzipiell nicht erfüllen, denn der Angreifer kann leicht herausfinden, welche der beiden verschlüsselt wurde, indem er die beiden Nachrichten selbst verschlüsselt.

passive Sicherheit
aktive Sicherheit

IND-CPA ist aber nicht die stärkstmögliche Definition, denn sie schützt nur gegen passive Angreifer, die zwar den Ciphertext abfangen können, aber nicht gegen aktive Angreifer, die den Ciphertext verändern können (man spricht hier auch verkürzend von *aktiv sicher* bzw. *passiv sicher*). Hierdurch entstehen Gefahren zweierlei Art:

1. Einerseits könnte ein nur passiv sicheres Verfahren gezielte Manipulationen des Klartexts durch Veränderung des Ciphertexts ermöglichen, zum Beispiel das Kippen eines bestimmten Bits. RSA ist in seiner unmodifizierten Form („Lehrbuch-RSA“) verwundbar für derlei Angriffe: Das Produkt zweier Ciphertexte entschlüsselt zum Produkt der beiden entsprechenden Klartexte.

[3] Man muss hierbei allerdings die Länge des Klartexts ausnehmen, da es aus informationstheoretischen Gründen immer einen Zusammenhang zwischen der Länge von Klar- und Ciphertext geben muss.

[4] Die Äquivalenz gilt nur für asymmetrische Verschlüsselung. Bei symmetrischer Verschlüsselung ist IND-CPA stärker als semantische Sicherheit[Buc16]. In jedem Fall folgt die semantische Sicherheit aus IND-CPA.

[5] Für symmetrische Verfahren erhält der Angreifer statt dem öffentlichen Schlüssel Zugriff auf ein Verschlüsselungsorakel.

2. Andererseits könnte ein Angreifer aber auch durch solche Manipulationen Kenntnisse über den geheimen Schlüssel oder über andere Nachrichten erlangen. Ein berühmter Fall ist eine von Daniel Bleichenbacher entdeckte Schwachstelle [Ble98] in (u. a.) vielen SSL-3.0-Implementierungen, die ein Angreifer als Padding-Orakel ausnutzen konnte, um beliebige Klartexte entschlüsseln zu können.

Eine Sicherheitsdefinition, die den zweiten Punkt berücksichtigt, ist die **Ununterscheidbarkeit im Chosen-Ciphertext-Angriff (IND-CCA)**. Sie ist auch über das selbe Spiel definiert wie IND-CPA, mit der Erweiterung, dass der Angreifer bei CCA zusätzlich Zugriff auf ein Entschlüsselungsorakel, mit dem er die Entschlüsselung beliebiger Ciphertexte erhalten kann (außer für den Ciphertext c, den er selbst erhalten hat). IND-CCA

Es gibt eine Variante dieser Definition, bei der der Angreifer nur in Schritt 2 Zugriff auf dieses Orakel hat, aber nicht in Schritt 4 (also nur bevor er den Ciphertext erhalten hat, nicht mehr danach) [BDPR98]. Diese Angriffsart wird *non-adaptive chosen ciphertext attack* oder (IND-)CCA1 genannt. Wenn es davon abgegrenzt werden soll, wird CCA auch als (IND-)CCA2 oder *adaptive chosen-ciphertext attack* bezeichnet. In dieser Arbeit ist mit (IND-)CCA grundsätzlich (IND-)CCA2 gemeint. IND-CCA1 IND-CCA2

Offensichtlich ist ein System, das IND-CCA-sicher ist, auch IND-CPA-sicher – ein Angreifer gegen CPA wäre auch ein Angreifer gegen CCA.

Vor der erstgenannten Gefahr durch aktive Angreifer schützt die Definition der **Unverformbarkeit** oder Unverfälschbarkeit, engl. *non-malleability* (NM). Unverformbarkeit bedeutet, dass ein Angreifer keine gezielten Manipulationen am Ciphertext vornehmen kann – jede Veränderung resultiert entweder in einem ungültigen Ciphertext, oder in einem, dessen Klartext in keinem Zusammenhang zum ursprünglichen Klartext steht [KL07, § 3.7]. Auch sie kann gegen verschiedene Angriffe definiert werden, wobei auch hier gilt NM-CCA > NM-CPA [BDPR98]. NM-CCA

Da NM-CCA und IND-CCA äquivalent sind [BDPR98], ist es egal, für welche der Definitionen man die Sicherheit eines Systems beweist. In der Praxis hat sich IND-CCA/IND-CCA2 als die Standarddefinition durchgesetzt. Aus ihr folgen sämtliche bisher genannten Definitionen (OW, IND-*, NM-*).

Quantifizierung

Auch die genannten Definitionen sind wertlos, wenn sie nicht quantifiziert werden: Wird die dem Angreifer zur Verfügung stehende Zeit nicht begrenzt, kann er durch Ausprobieren aller Schlüssel den Klartext berechnen. Auch ist es nicht praktikabel, zu fordern, dass der Angreifer *nie* Erfolg hat: Würde er einfach versuchen, den Schlüssel zu raten, wäre seine Erfolgswahrscheinlichkeit leicht größer als Null.

Man fordert daher i. d. R., dass die Erfolgswahrscheinlichkeit des Angreifers *vernachlässigbar* klein ist. In Fällen wie dem des Ununterscheidbarkeits-Angreifers liegt die

Erfolgswahrscheinlichkeit schon bei simplem Raten bei $\frac{1}{2}$, hier muss man die Schwelle daher anpassen: Man fordert, dass die Erfolgswahrscheinlichkeit unter $\frac{1}{2} + \epsilon$ liegt für ein vernachlässigbar kleines ϵ.

Die Zeit, die der Angreifer benötigt, um über diese Schwelle zu kommen, ist ein Maß für das Sicherheitsniveau des Verfahrens: Ist die Erfolgswahrscheinlichkeit des Angreifers nach $\mathcal{O}(2^l)$ Rechenschritten noch vernachlässigbar klein, spricht man von einem Sicherheitsniveau von l, oder auch von „l bit Sicherheit“ [Buc16, S. 127].

2.3.2 Primitiven

Kryptografische Grundbausteine nennt man *Primitiven*. Das sind zum Beispiel elementare Verschlüsselungs- und Signaturverfahren oder Hashfunktionen. Für ein vollständiges Protokoll wie zum Beispiel TLS (Transport Layer Security), das u. a. innerhalb von HTTPS verwendet wird, werden mehrere solcher Primitiven zusammen verwendet.

Ich werde im Folgenden die für diese Arbeit relevanten Arten von Primitiven vorstellen und die entsprechende Schnittstelle definieren. Die genannten Schnittstellen sind die Minimaldefinitionen, es können noch weitere Parameter hinzukommen, beispielsweise ein Initialisierungsvektor.

Verschlüsselung

Zu allen folgenden Definitionen gehören implizit immer auch Mengen, aus denen die jeweiligen Werte stammen müssen, beispielsweise muss eine Nachricht i. d. R. ein Bitstring fester Länge sein[6]. Diese Mengen sind MSP (Nachrichtenraum, engl. *Message Space*), CSP (Ciphertextraum) sowie KSP bzw. PKSP und SKSP (symmetrischer bzw. öffentlicher und geheimer Schlüsselraum). Die Definitionen stammen aus [CS03, § 7; NewH, § 1.3].

MSP
CSP
KSP

Die einfachste Art der Verschlüsselung ist **symmetrische Verschlüsselung**, bei der für die Ver- und Entschlüsselung der selbe Schlüssel verwendet wird. Hierzu sind folgende Funktionen zu implementieren:

- $\mathsf{enc} : \mathsf{msg}, \mathsf{k} \mapsto \mathsf{ct}$
- $\mathsf{dec} : \mathsf{ct}, \mathsf{k} \mapsto \mathsf{msg}$

Als Schlüssel kann ein beliebiger Wert $\mathsf{k} \in \mathsf{KSP}$ verwendet werden. Das Verfahren ist *korrekt*, wenn für alle $k \in \mathsf{KSP}$ und alle $\mathsf{msg} \in \mathsf{MSP}$ gilt: $\mathsf{dec}(\mathsf{enc}(\mathsf{msg}, \mathsf{k}), \mathsf{k}) = \mathsf{msg}$.

Asymmetrische Verschlüsselung verhält sich ähnlich, allerdings gibt es zwei Schlüssel, die zueinander passen müssen. Sie können daher nicht frei gewählt werden,

[6] Um Nachrichten beliebiger Länge zu verschlüsseln, wird die Verschlüsselungsfunktion mittels eines sogenannten Betriebsmodus geeignet mehrfach aufgerufen.

sondern werden von einer Schlüsselerzeugungsfunktion generiert. Die Funktionen sind:

- $\mathsf{gen} : \emptyset \mapsto \mathsf{pk}, \mathsf{sk}$
- $\mathsf{enc} : \mathsf{msg}, \mathsf{pk} \mapsto \mathsf{ct}$
- $\mathsf{dec} : \mathsf{ct}, \mathsf{sk} \mapsto \mathsf{msg}$

Die Korrektheit ist entsprechend zu oben definiert, dabei wird vorausgesetzt, dass die verwendeten Schlüssel mit gen generiert wurden. Verfahren nach dieser Definition werden – insbesondere in Abgrenzung zur nachfolgenden Definition – **Schlüsseltransportverfahren** (engl. *Key Transfer Mechanism, KTM*) oder *PKE* (*Public Key Encryption*) genannt.

KTM
PKE

Asymmetrische Verschlüsselung ist typischerweise rechenaufwändiger als symmetrische. Man verschlüsselt daher in der Regel Nachrichten mit einem symmetrischen Verfahren und übermittelt nur den dafür nötigen Schlüssel mit einem asymmetrischen Verfahren. Dieser symmetrische Schlüssel wird dann *gemeinsames Geheimnis* genannt. Dieses Verfahren hat einen weiteren Vorteil: Da die „Nachricht“ des asymmetrischen Verfahrens dann nicht der zu übertragenden Nachricht entsprechen muss, sondern ein zufälliger Wert sein kann, kann man deren Wahl der Verschlüsselungsfunktion überlassen.

hybride Verschlüsselung

Geheimnis

Derlei Verfahren nennt man **Schlüsselkapselungsverfahren** oder **Schlüsseleinigungsverfahren**, engl. *Key Encapsulation Mechanism*, **KEM**s. Eine solche Konstruktion kann Effizienzvorteile haben (beispielsweise bei Classic McEliece, § 4.2) oder ein Verfahren überhaupt erst möglich machen (z. B. den Diffie-Hellman-Schlüsselaustausch). Zur Unterscheidung spricht man von Ver- und Entkapselung statt -schlüsselung:

KEM

- $\mathsf{gen} : \emptyset \mapsto \mathsf{pk}, \mathsf{sk}$
- $\mathsf{encaps} : \mathsf{pk} \mapsto \mathsf{msg}, \mathsf{ct}$
- $\mathsf{decaps} : \mathsf{ct}, \mathsf{sk} \mapsto \mathsf{msg}$

Andere Primitiven

Eine wichtige weitere Primitive sind **Hashfunktionen**. Hashfunktionen wandeln eine Eingabe deterministisch in eine zufällig erscheinende Ausgabe fester Länge um. Sie müssen dabei drei Anforderungen erfüllen [BDS09, § 7.1]:

- *Urbildresistenz:* Es muss schwer sein, zu einem Ausgabewert einen Eingabewert zu finden.
- *schwache Kollisionsresistenz:* Es muss schwer sein, zu einem Eingabewert einen weiteren Eingabewert zu finden, der die selbe Ausgabe erzeugt (engl. *second preimage resistance*).
- *starke Kollisionsresistenz:* Es muss schwer sein, zwei Werte zu finden, die die selbe Ausgabe erzeugen.

Die Aussagen können quantifiziert sein, so bedeutet t-Urbildresistenz, dass ein Angreifer mindestens die Zeit t benötigt, um ein Urbild zu finden.

Die Eingabelänge von generischen Hashfunktion ist i. d. R. praktisch unbegrenzt, sie akzeptiert Eingaben beliebiger Länge. Die Signatur ist also $H : \mathbb{B}^* \to \mathbb{B}^\ell$.

Für manche Fälle werden jedoch Ausgaben variabler Länge, aber mit den selben Sicherheitsanforderungen, benötigt. Solche Funktionen nennt man **Extendable Output Functions** (**XOFs**), ihre Signatur ist $XOF : \vec{s} \in \mathbb{B}^*, n \in \mathbb{N} \mapsto \vec{x} \in \mathbb{B}^n$.

2.3.3 Das Random Oracle Model

Viele Verfahren verwenden intern Hashfunktionen, insbesondere innerhalb einer sogenannten Transformation, mit der aus dem passiv sicheren Kern-Schema ein aktiv sicheres Schema gemacht wird (siehe § 5.2). Oft gelingt es dabei aus praktischen Gründen nicht, einen Sicherheitbeweis zu führen.

Mit einer vereinfachenden Annahme sind solche Beweise oft doch möglich, nämlich, indem angenommen wird, dass die Hashfunktion sich wie ein Zufallsorakel (engl. *Random Oracle*) verhält. Diese Annahme nennt man das Random Oracle Model (ROM), oder (selten) Zufallsorakel-Modell; man sagt beispielsweise, „der Beweis wurde im ROM geführt“ oder ein Verfahren sei „im ROM mindestens so sicher wie“ ein anderes [KL07, § 13.1].

Ein Zufallsorakel ist ein Gedankenmodell. Es ist eine zentrale Instanz, die von allen an einem Verfahren Beteiligten befragt werden kann, und zwar derart, dass die anderen Beteiligten nichts darüber erfahren. Das Zufallsorakel hat die selbe Signatur wie eine Hashfunktion, d. h. die Eingabe ist ein Bitstring beliebiger Länge, die Ausgabe einer fester Länge. Wird das Zufallsorakel mit einem Wert befragt, den es vorher nicht erhalten hat, wählt es einen zufälligen Ausgabewert; außerdem merkt es sich die Ein- und Ausgabe. Wird es wiederholt mit dem selben Eingabewert befragt, gibt es den selben Wert zurück wie zuvor. Nicht zuletzt aufgrund des hierfür nötigen unendlich großen Speichers ist ein Zufallsorakel nicht praktisch umsetzbar.

Der Nutzen des Random Oracle Models liegt in zwei Punkten. Zum einen kann man so den Beweis von einer konkreten Hashfunktion abstrahieren. Insbesondere könnten Hashfunktionen gewisse praktische Schwächen haben, d. h. die garantierten Eigenschaften in manchen Fällen nicht erfüllen. Nimmt man ein Zufallsorakel an, kann man die Sicherheit des Verfahrens an sich losgelöst von einer konkreten Hashfunktion beweisen; für konkrete Instanziierungen setzt man eine geeignete Hashfunktion ein.

Der zweite Vorteil des ROM ist, dass man in einem Spiel die Anfragen an das Zufallsorakel abfangen und ggf. modifizieren kann. So sind kleine Einblicke in die Arbeitsweise des Angreifers (der ansonsten eine black box ist) möglich, die manche Beweise erst ermöglichen.

Im ROM geführte Beweise sind daher weniger aussagekräftig als Beweise im Standardmodell. Dennoch sind sie ein hilfreicher Ersatz, wenn ein Beweis im Standardmodell nicht möglich ist. Denn auch wenn es möglich ist, künstliche Verfahren zu konstruieren, die im ROM sicher, aber im Standardmodell unsicher sind (z. B. [KL07, Üb. 13.2]), zeigt ein ROM-Beweis in der Regel, dass das Verfahren selbst keine groben konzeptuellen Schwächen hat.

2.4 Mathematische Grundlagen und Notation

In diesem Abschnitt werde ich die von mir verwendete Notation erläutern. Nicht alle hier erläuterten mathematischen Hintergründe sind für das Verständnis der vorliegenden Arbeit nötig. Ich empfehle, diesen Abschnitt zunächst nur grob zu lesen und bei Bedarf als Nachschlagewerk zu nutzen. Um dies zu erleichtern, sind alle verwendeten Notationen im Seitenrand vermerkt.

Die Definitionen von Gruppen, Ringen und Körpern werden vorausgesetzt, es sei aber knapp an die Grundzüge erinnert: Eine Gruppe ist eine Struktur mit nur einer Operation (meistens als Addition geschrieben). Ein Ring ist eine Struktur mit zwei Operationen (Addition und Multiplikation), die durch Distributivgesetze miteinander verknüpft sind. Gelten einige weitere Regeln, wird ein Ring zum Körper (engl. *field*). Gruppe Ring Körper

$\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ ist der Restklassenring, also der Ring, in dem Addition und Multiplikation Modulo q durchgeführt werden (der Schrägstrich steht für die Bildung des Quotientenrings). Ist q prim, ist diese Struktur sogar ein Körper und wird dann $\mathbb{F}_q$ geschrieben. $\mathbb{Z}_q$ $\mathbb{F}_q$

Mit $\mathbb{B}$ bezeichne ich meist Bits, also die Menge $\{0,1\}$, je nach Kontext werden hier aber auch Rechnungen durchgeführt, die dann als Rechnungen im $\mathbb{F}_2$ aufzufassen sind. Ein Bitstring der Länge n wird $\mathbb{B}^n$ notiert. $\mathbb{B}$

Es gibt noch eine andere Art, einen Ring zu erzeugen, die in dieser Arbeit relevant ist, das sind sogenannte **Polynomringe**. Diese werden $\mathcal{R} = \mathbb{Z}[X]$ notiert, die Ringelemente sind hierbei Polynome, sie werden in Fettschrift notiert: $\boldsymbol{r} \in \mathcal{R} = r_0 + r_1 \cdot X + r_2 \cdot X^2 + \cdots$. Die Addition ist komponentenweise (wie bei Vektoren); die Multiplikation erfolgt wie von Polynomen bekannt. Sie wird $\boldsymbol{r}_1 \times \boldsymbol{r}_2$ geschrieben, und das Produkt ist wieder ein Polynom, dessen Grad im Allgemeinen höher ist als der der Faktoren. $\mathcal{R}$ $\mathbb{Z}[X]$

Ein Polynomring nach dieser Definition hat jedoch im Allgemeinen einen beliebig großen Grad und beliebig große Koeffizienten, womit er sich nicht für Berechnungen in (endlich großen) Computern eignet. Daher definiert man den Polynomring nicht über die ganzen Zahlen, sondern über einen Restklassenring wie oben, also $\mathcal{R}_q = \mathbb{Z}_q[X]$. Die Koeffizienten eines Ringelements sind dann aus $\mathbb{Z}_q$. $\mathcal{R}_q$

$\mathcal{R}/\langle f \rangle$ Um den Grad zu begrenzen, kann man (analog zu den Restklassenringen über den ganzen Zahlen) definieren $\mathcal{R}'_q = \mathcal{R}_q/\langle f \rangle$, für ein Polynom $f \in \mathcal{R}$. Diese Definition entspricht der des Restklassenrings oben: So wie die Zahlen 13 und 23 im Ring $\mathbb{Z}_{10}$ kongruent zueinander sind (weil bei Division durch 10 der selbe Rest bleibt), können auch zwei Polynome kongruent sein, wenn bei Polynomdivision durch f der selbe Rest bleibt. Die Elemente von $\mathcal{R}'_q$ sind also Polynome, deren Grad durch den Grad von f begrenzt ist – ebenso, wie die Elemente des Körpers Z_q immer kleiner als q sind.

Ideal Eine weniger bekannte Struktur ist das **Ideal**. Ein Ideal ist eine Untermenge eines Ringes mit zwei Eigenschaften. Erstens muss das Ideal unter der Addition und Subtraktion abgeschlossen sein, die Summe zweier Idealelemente also auch im Ideal sein. Außerdem muss das Produkt eines Idealelements mit einem Ringelement im Ideal liegen. Ein Beispiel für ein Ideal ist die Menge der geraden Zahlen (als Untermenge der ganzen Zahlen): die Summe zweier gerader Zahlen ist gerade, das Produkt einer geraden mit einer beliebigen ganzen Zahl ebenso.

$\vec{v}$ Vektoren werden mit Pfeil geschrieben: $\vec{v} \in \mathbb{R}^n = (v_1, \ldots, v_n)$. Das Skalarprodukt
$\vec{v} \cdot \vec{w}$ notiere ich $\vec{v} \cdot \vec{w}$; Multiplikation mit Matrizen oder Skalaren mit dem normalen Malpunkt $(\cdot)$ oder Hintereinanderschreibung.

M Matrizen sind in serifenloser Schrift geschrieben: M. Die n-dimensionale Einheitsmatrix wird $\mathbb{1}_n$ notiert. Für zwei Matrizen A, B ist $[\mathsf{A} \mid \mathsf{B}]$ die Matrix, die aus den Spaltenvektoren beider Matrizen besteht („nebeneinander geschrieben“).

Ein Vektorraum ist immer über einen Körper definiert, d. h. die Einträge eines Vektors stammen aus einem Körper. Eine Verallgemeinerung des Vektorraums ist ein
Modul **Modul** (der Modul, gesprochen ['mo:dʊl], Plural *Moduln* [Wik19]). Die Definition ist identisch: ein Modul hat eine natürlichzahlige Dimension n, seine Elemente sind ein Tupel von n Ringelementen. Im Rahmen dieser Arbeit werden insbesondere Moduln
$\vec{v}$ (Fettschrift) über einen Polynomring betrachtet[7]. Diese notiere ich $\vec{v} \in \mathcal{R}^n = (v_1, v_2, \ldots v_n)$. Matrizen mit Einträgen aus einem Polynomring werden fett und serifenlos dargestellt:
A $\mathbf{A} \in \mathcal{R}^{m \times n}$.

$\xleftarrow{\$}$ Mit $\xleftarrow{\$}$ wird die zufällige, gleichverteilte Wahl eines Wertes aus einer Menge symbolisiert. Beispielsweise hat die Variable b nach der Operation $b \xleftarrow{\$} \mathbb{B}$ mit je 50 % Wahrscheinlichkeit den Wert 1 oder 0. Wenn nicht gleichverteilt, sondern aus einer Wahrscheinlichkeitsverteilung gezogen wird, ist diese rechts des Pfeils angegeben, z. B. steht $x \xleftarrow{\$} \mathcal{G}(\mathbb{R})$ für die Wahl nach der Gaußverteilung.

$\lfloor \cdot \rfloor, \lceil \cdot \rceil, \lfloor \cdot \rceil$ Die Gaußklammern $\lfloor \cdot \rfloor$ und $\lceil \cdot \rceil$ bezeichnen Auf- bzw. Abrundung zur nächsthöheren bzw. -niedrigeren Ganzzahl, $\lfloor \cdot \rceil$ die Rundung zur nächstgelegenen ganzen Zahl. Mit
// $\cdot \,//\, \cdot$ wird die Ganzzahldivision bezeichnet, also $\lfloor \cdot / \cdot \rfloor$.

[7] Tatsächlich sind alle Vektorräume auch Moduln, werden dann aber nicht so bezeichnet.

3 Post-Quanten-Kryptografie

In diesem Kapitel stelle ich die Grundlagen zu den im nächsten Kapitel vorgestellten Post-Quanten-Verfahren vor, das sind z. B. mathematische Hintergründe, die zugrundeliegenden Probleme und dergleichen.

Die Verfahren werden üblicherweise nicht nach technischen Kriterien o. ä. eingeteilt, sondern basierend auf den ihrer Sicherheit zugrunde liegenden Problemen. In dieser Arbeit betrachte ich zwei Klassen von Verfahren: Gitter-basierte und Code-basierte; diese beiden Klassen stelle ich im Folgenden vor. Am Ende des Kapitels folgt ein kleiner Überblick über die anderen Klassen.

3.1 Gitter

Ein Gitter (engl. *Lattice*) ist eine Menge von regelmäßig im Raum angeordneten Punkten, anders gesagt, eine diskrete Untermenge eines n-dimensionalen Vektorraums. Wie ein Vektorraum wird ein Gitter von einer Basis, n linear unabhängigen Vektoren, aufgespannt [MR09]:

$$\mathcal{L}(\vec{b}_1, \dots \vec{b}_n) = \left\{ x_1\vec{b}_1 + \cdots + x_n\vec{b}_n \;\middle|\; x_i \in \mathbb{Z} \right\}.$$

Die Definition ist also fast identisch zu der eines Vektorraums, mit dem Unterschied, dass nur *ganzzahlige* Vielfache der Basisvektoren zulässig sind. (Die Basisvektoren selbst sind jedoch aus $\mathbb{R}^n$.) Eine wichtige Eigenschaft ist, dass ein Gitter (wie Vektorräume auch) unendlich viele verschiedene Basen hat.

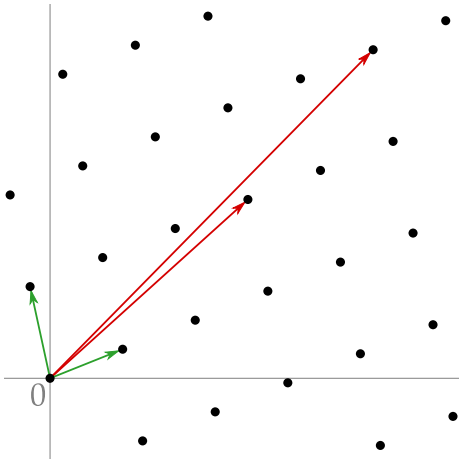

Abbildung 3.1: Ein Gitter in $\mathbb{R}^2$, gegeben durch zwei verschiedene Basen

3.1.1 Zugrundeliegende Probleme

Das *Shortest Vector Problem*

Eines der grundlegenden Probleme in Gittern ist das *Shortest Vector Problem* (SVP) SVP

[MR09]. Es besteht darin, zu einem durch eine Basis gegebenen Gitter den kürzesten Vektor außer $\vec{0}$ in ihm zu finden (d. h. den Gitterpunkt, der am nächsten an $\vec{0}$ liegt – jedes Gitter enthält $\vec{0}$). Für kryptografische Zwecke ist das SVP insbesondere in der sogenannten Approximations-Variante interessant. Hier ist die Aufgabe, statt dem kürzesten Vektor einen beliebigen, maximal γ-mal so langen Vektor zu finden. Dies wird SVP_γ notiert.

SVP_γ

Für SVP_γ mit $\gamma = 2^{\mathcal{O}(n)}$ (n ist die Dimension des Gitters) gibt es mit dem LLL-Algorithmus von Lenstra, Lenstra und Lovász (1982) einen effizienten Algorithmus, für den seitdem nur geringfügige Verbesserungen gefunden wurden. Für eine bessere Approximation (also SVP oder $\mathsf{SVP}_{\mathrm{poly}(n)}$) wurden bis heute nur Algorithmen mit exponentieller Laufzeit gefunden. Auch mit Quantencomputern scheinen keine besseren Lösungen möglich. Es wird daher vermutet, dass das SVP schwer ist (und auch in einer Welt mit Quantencomputern sein wird) [MR09, § 1].

Das Problem ist nur schwer im komplexitätstheoretischen Sinne, also im schlechtesten Fall. Denn wenn die gegebene Basis aus sehr kurzen (und möglichst orthogonalen) Vektoren besteht, ist die Lösung des SVP relativ leicht möglich [MR09, § 5.1]. Die Umwandlung einer „schlechten“ Basis in eine solche „gute“ Basis ist vermutlich schwer (denn sie würde ja die Lösung des SVP ermöglichen). In Abb. 3.1 sind eine „gute“ (grün) und „schlechte“ (rot) Basis beispielhaft dargestellt.

In dieser Diskrepanz liegt der Grund, warum sich Gitter-Probleme für Verschlüsselung eignen: Die „schlechte“ Basis kann veröffentlicht werden, ohne dass daraus Rückschlüsse auf den geheimen Schlüssel (eine „gute“ Basis) möglich sind. Deren Kenntnis ermöglicht es jedoch, Operationen effizient durchzuführen, die sonst schwer sind; das kann z. B. die Entschlüsselung sein.

Die Nutzung von Gitter-Problemen für Kryptographie geht auf eine Idee von Miklós Ajtai und Cynthia Dwork [AD97]. Im Unterschied zu den in dieser Arbeit untersuchten Verfahren wird in ihrer Idee direkt ein Gitter genutzt anstatt der folgenden Abstraktion:

Learning with Errors

Dieses für gitterbasierte Kryptographie grundlegende Problem wurde 2005 von Oded Regev definiert [Reg05]. Die Aufgabe ähnelt der, eine Lösung für ein lineares Gleichungssystem zu finden, allerdings mit der zusätzlichen Schwierigkeit, dass jede Gleichung einen (unbekannten) kleinen Fehler enthält (hierbei sind die $a_{i,j}$ und b_i bekannt, und $\vec{s}$ gesucht[1]):

LWE

$$\begin{aligned} a_{1,1}\,s_1 + a_{1,2}\,s_2 + \cdots + a_{1,n}\,s_n \;+\; e_1 &= b_1 \\ a_{2,1}\,s_1 + a_{2,2}\,s_2 + \cdots + a_{2,n}\,s_n \;+\; e_2 &= b_2 \\ &\vdots \end{aligned} \tag{3.1}$$

[1] es kann auch $\vec{e}$ gesucht sein; diese Formulierung ist äquivalent, da das Gleichungssystem mit Kenntnis von $\vec{e}$ trivial zu lösen ist.

Die Rechnungen finden dabei im Restklassenkörper $\mathbb{Z}_p$ statt, also Modulo einer Primzahl p. Die $a_{i,j}$ und s_j werden (unabhängig und gleichverteilt) zufällig aus $\mathbb{Z}_q$ gezogen, während die Fehler e_i nur klein sein dürfen – sie werden aus einer Fehlerverteilung χ gezogen, i. d. R. einer diskreten Gauß-Verteilung.

Fasst man die $a_{i,j}$ als Matrix A sowie die e_i, b_i und s_j als Vektoren $\vec{e}$, $\vec{b}$ bzw. $\vec{s}$ zusammen, ergibt sich die (äquivalente) Formulierung

$$\mathsf{A} \cdot \vec{s} + \vec{e} = \vec{b}. \tag{3.2}$$

Ich werde derlei Gleichungen im Folgenden gelegentlich farblich markieren, um die Äquivalenz zum LWE-Problem zu verdeutlichen. Dabei werden öffentliche Werte grün, geheime Werte orange markiert, wie im folgenden Beispiel ($\bullet$ ist das Skalarprodukt):

$$\underline{\vec{a}} \bullet \underline{\vec{s}} + \underline{e} = \underline{b}$$

Während diese Aufgabe mit $e = 0$ (also als gewöhnliches lineares Gleichungssystem) effizient (z. B. per Gauß-Elimination) zu lösen wäre, scheint LWE schwer zu sein. Regev bewies dies in [Reg05, Thm. 1.1] unter der Annahme, dass das SVP schwer ist, und für geeignete Fehlerverteilungen χ.

Diese Reduktion ist nur eine Quanten-Reduktion, d. h. der Algorithmus zur Überführung einer SVP- in eine LWE-Instanz ist ein Quantenalgorithmus. Es ist also nicht ausgeschlossen, dass das LWE für klassische Computer leicht ist, während das SVP nur für Quantencomputer leicht, aber für klassische Computer schwer ist. In einer Post-Quanten-Welt (für die diese Verfahren ja entwickelt werden) ist dieser Unterschied aber irrelevant.

Das zugehörige Entscheidungsproblem (dLWE) besteht darin, für gegebene LWE-Samples $(\mathsf{A}, \vec{b})$ zu bestimmen, ob die $\vec{b}$ auf o. g. beschriebene Art berechnet, oder (gleichverteilt) zufällig gezogen wurden. dLWE ist mindestens so schwer wie (Such-) LWE [Reg05, Lemma 4.2] – die umgekehrte Reduktion ist trivial [NewH, § 1.1.3]. Such- und Entscheidungsvariante sind also äquivalent. dLWE

Worst Case vs. Average Case Ein wichtiges Detail habe ich bisher noch nicht erwähnt: In der Komplexitätstheorie werden Probleme anhand der Laufzeit im schlechtesten Fall (*Worst Case*) beurteilt. Ein Problem wird z. B. schon als schwierig angesehen, wenn es Eingaben gibt, für die das Problem schwer ist; die Komplexitätsklasse bemisst sich nach der Laufzeit, die für die schwierigste Eingabe benötigt wird. Das bedeutet, dass ein Problem theoretisch schon als schwer gälte, wenn es nur für eine einzige Eingabe schwer, aber für alle anderen Eingaben leicht wäre.

Für Verschlüsselung reicht das aber nicht: Es muss schwer sein, sämtliche Nachrichten ohne Kenntnis des geheimen Schlüssels zu entschlüsseln, nicht nur manche – anders gesagt: Das Problem muss für alle Eingaben schwer sein. Man begnügt sich in der Praxis damit, dass das Problem für zufällige Eingaben, also im Durchschnittsfall,

schwer ist. Um die Sicherheit eines Verfahren zu beweisen, reicht also eine gewöhnliche Reduktion (wie sonst in der theoretischen Informatik) nicht aus, sondern es muss eine zufällige Eingabe eines Problems auf eine *schwere* Eingabe eines im komplexitätstheoretischen Sinne schweren Problems (oder auf alle möglichen Eingaben) reduziert werden. Eine solche Reduktion wird **worst-case-to-average-case reduction** genannt.

Regev zeigt in [Reg05] einen Spezialfall einer solchen Reduktion, und zwar reduziert er LWE auf sich selbst, genauer: eine Untermenge der LWE-Eingaben auf alle LWE-Eingaben. LWE ist also auch in diesem engeren Sinne schwer.

LWE-Varianten

Auf Standard-LWE basierende Kryptoverfahren haben den Nachteil, relativ ineffizient zu sein, sowohl im Bezug auf die Menge der zu übertragenden Daten als auch auf den Rechenaufwand. Es wurden daher verschiedene Varianten entwickelt, bei denen statt Restklassenkörpern andere algebraische Strukturen (Ringe, ...) verwendet werden.

Die höhere Effizienz wird dabei beispielsweise dadurch erreicht, dass die verwendeten Strukturen intern stärker strukturiert sind. Beispielsweise kann statt einer völlig zufälligen Matrix eine zyklische Matrix verwendet werden, bei der sich aus der ersten Spalte die Einträge aller anderen Spalten berechnen lassen. Derlei Optimierungen werden jedoch immer mit einem Sicherheitsrisiko erkauft, denn es besteht immer die Gefahr, dass die höhere Strukturiertheit auch für Angriffe ausgenutzt werden kann.

Als Beispiel für die Gefahren effizienterer LWE-Varianten sei Compact-LWE [LLKN17] genannt, ein zum NIST-PQC-Standardisierungsprozess eingereichtes Verschlüsselungsverfahren. Die Autoren dachten, durch Verwendung mehrerer Fehler- und weiterer geheimer Werte eine sehr kleine Dimension nutzen zu können, und behaupteten, diese Variante sei dennoch so sicher wie das klassische LWE. Das Verfahren war daher eines der kleinsten und schnellsten unter denen in Kategorie 3. Es zeigte sich jedoch, dass das Verfahren völlig unsicher ist: Nachrichten können von einem Angreifer in weit unter einer Sekunde auf einem normalen Desktop-Computer entschlüsselt werden [BT17; LLPX18].

Ring-LWE Sei $f(x) = x^n + 1$ und n eine Zweierpotenz (f ist dann irreduzibel). Sei der Polynomring R_q definiert als $R_q = \mathbb{Z}_q[X]/\langle f(X)\rangle$. Hierbei muss q eine Primzahl sein und gelten $q \equiv 1 \mod 2n$.

Die Elemente von R_q können dann als Polynome mit einem Grad kleiner n (da die Multiplikation Modulo f ausgeführt wird) und Koeffizienten aus $\mathbb{Z}_q$ repräsentiert werden. (Zum Vergleich: Bei normalem LWE sind es Vektoren mit n Koeffizienten aus $\mathbb{Z}_q$. Die Probleme sind also auf den ersten Blick recht ähnlich, die interne Struktur ist jedoch verschieden.)

Ring-LWE wurde von Vadim Lyubashevsky, Chris Peikert und Oded Regev in [LPR10] definiert. Sie konnten die Schwierigkeit ebenfalls auf das SVP zurückführen, jedoch eingeschränkt auf den Spezialfall der Ideal-Gitter (engl. *ideal lattices*).

Ein Ideal ist eine Untermenge eines Rings mit der Eigenschaft, abgeschlossen bezüglich der Addition in sich sowie bezüglich der Multiplikation mit beliebigen Ringelementen zu sein. Beispielsweise ist die Untermenge der graden Zahlen aus den ganzen Zahlen $2\mathbb{Z} \subset \mathbb{Z}$ ein Ideal: Die Summe zweier gerader Zahlen ist gerade, und die Multiplikation einer geraden mit einer ganzen Zahl ist gerade. Ideal

Ideal-Gitter sind eine Spezialisierung von Gittern analog zu dieser Definition. Die Details sind für diese Arbeit nicht relevant, wichtig ist nur, dass das Gitter dadurch stärker strukturiert ist, mit den oben erwähnten Risiken.

Es wird weithin angenommen, dass das SVP auch für Ideal-Gitter schwer ist. Dennoch gibt es einige potentielle Einschränkungen. So ist dieser Spezialfall weniger gut untersucht als allgemeine Gitter. Außerdem wird befürchtet, dass die zusätzliche Struktur (im Vergleich zu einem völlig zufälligen Gitter) Angriffe erleichtert. Bisher wurden aber noch keine substantiell besseren Angriffe gefunden.

Modul-LWE Adeline Langlois und Damien Stehlé präsentierten 2012 [LS15] einen Mittelweg zwischen diesen beiden Extremen: LWE über Moduln[2].

Statt über Ringe wird das LWE-Problem über Moduln mit kleiner Dimension (z. B. zwischen 2 und 4 bei Kyber [Kyb]) definiert. Werte, die bei Ring-LWE Skalare (eigentlich: Ringelemente, also Polynome) sind, sind bei Modul-LWE Vektoren über Ringelemente; aus Vektoren werden entsprechend Matrizen.

Die Dimension des Rings kann dabei bei gleichbleibender Sicherheit um die Dimension des Moduls verkleinert werden [LS15, S. 3], entsprechend reduziert sich die Gefahr, dass die Ringstruktur ausgenutzt werden kann. Ähnlich wie bei Ring-LWE kann die Sicherheit aber nicht direkt auf allgemeine Gitter zurückgeführt werden, sondern nur auf Modul-Gitter, deren Strukturiertheit zwischen der allgemeiner und der von Idealgittern liegen.

3.1.2 Grundlegende Verfahren

Die später vorgestellten, zum NIST-Standardisierungsprozess eingereichten Kryptoverfahren basieren im Wesentlichen auf zwei Papern, in denen die Verfahren in einer grundlegenden, eher theoretischen Form entworfen wurden; die Einreichungen ergänzen diese, zum einen um technische Verfeinerungen und Effizienzverbesserungen, andererseits um konkrete Parameter und Sicherheitsabschätzungen sowie eine Referenzimplementierung (und oft auch eine erste optimierte Implementierung).

[2] Es sei daran erinnert, das dieser Begriff nichts mit Resten (*Modulo*) zu tun hat, sondern Moduln als Verallgemeinerung von Vektorräumen gemeint sind. Siehe Abschnitt 2.4.

Daher sind sich die Einreichungen oft in manchen Punkten ähnlich. Im Folgenden werde ich deshalb die grundlegenden Verfahren kurz vorstellen und vergleichen. Die Details der Einreichungen und die eher implementierungsrelevanten Details werden im nächsten Kapitel dargestellt.

Regev (2005)

Im selben Paper [Reg05], in dem Regev auch das LWE definiert, entwirft er bereits ein darauf basierendes Verschlüsselungsverfahren. Es verwendet, wie LWE, einen Restklassenkörper $\mathbb{Z}_p$ (p prim). Da das System in exakt dieser Form nicht verwendet wird, stelle ich es nur in den Grundzügen und informell vor.

Als öffentlicher Schlüssel dienen LWE-Samples wie in Gl. 3.2: $\text{pk} = (\mathsf{A}, \vec{b} = \mathsf{A}\cdot\vec{s}+\vec{e})$, der geheime Schlüssel ist $\text{sk} = \vec{s}$. Die Werte A und $\vec{s}$ sind zufällig gleichverteilt gewählt, die Fehler $\vec{e}$ aus einer diskreten Gaußverteilung um 0. Die Sicherheit des geheimen Schlüssels ist also trivial auf LWE zurückführbar.

Das System ist als Beispiel konzipiert – es ermöglicht nur die Übertragung eines einzelnen Bits. Zur Verschlüsselung werden eine zufällige Untermenge der Zeilen von A sowie die entsprechende Untermenge von Einträgen aus $\vec{b}$ jeweils aufsummiert (die Summen werden im Folgenden $\vec{a}$ und β bezeichnet). Anders formuliert: einige der Zeilen von Gleichung 3.1 werden ausgewählt und jeweils die linken Seiten (zu $\vec{a}$) und rechten Seiten (zu β) aufsummiert. Nach der dLWE-Behauptung ist $(\vec{a}, \vec{\beta})$ für Dritte nicht von einer Zufallsverteilung zu unterscheiden [Reg05, Lemma 5.4].

Der Empfänger hingegen kann $\vec{a}$ mit dem geheimen Schlüssel $\vec{s}$ (skalar) multiplizieren, dieses Produkt ist ungefähr β. Konkret: $\beta - \vec{a} \bullet \vec{s}$ ist genau die Summe der Fehler e_i. Wir haben also eine Möglichkeit gefunden, auf zwei Arten *ungefähr* den selben Wert zu übertragen, jedoch für Dritte unbemerkbar – insbesondere kann ein Beobachter nicht herausfinden, wie nahe sich die Werte tatsächlich sind.

Der Kniff des Verfahrens ist nun, in einem dieser Werte die Nachricht msg zu verstecken. Wenn $\text{msg} = 1$ ist, erhöhen[3] wir β um $p/2$, also $\beta' = \beta + \text{msg} \cdot p/2$. Dies ändert nichts an der Pseudozufälligkeit der Verteilung. Der Ciphertext ist $(\vec{a}, \beta')$.

Der Empfänger kann nun (analog zu oben) den Wert $\beta' - \vec{a} \bullet \vec{s} \approx \beta' - \beta$ berechnen. Liegt dieser Wert etwa bei $p/2$, ist die Nachricht eins, liegt er um $0 \pmod p$, ist die Nachricht null.

Die Entschlüsselung ist genau dann erfolgreich, wenn die Summe der Fehler kleiner als $p/4$ ist. Das bedeutet, dass die Fehler nicht zu groß sein dürfen. Andererseits dürfen sie nicht zu klein gewählt werden, da sonst die LWE-Bedingung nicht erfüllt ist und das Verfahren nicht sicher ist. Es ist außerdem nicht möglich, den Fehler gleichverteilt aus einem Intervall z. B. $]-p/4, +p/4[$ zu wählen – LWE verlangt eine Gauß-Verteilung. Die Gauß-Verteilung hat die Eigenschaft, dass die Eintrittswahrscheinlichkeit eines

[3] Es sei daran erinnert, dass wir uns im Restklassenkörper $\mathbb{Z}_p$ befinden.

Ereignisses auch weit vom Mittelpunkt nie ganz auf Null sinkt. In der Praxis haben derartige Verfahren daher immer eine (wenn auch sehr kleine) Wahrscheinlichkeit, dass die Entschlüsselung scheitert. Es ist möglich, diese Wahrscheinlichkeit gegen die Ciphertextlänge auszubalancieren [Kyb, § 1.4].

„duale" Variante (2008–2011)

Im Jahr 2008 veröffentlichten Craig Gentry, Chris Peikert und Vinod Vaikuntanathan [GPV08] eine Variante zu Regevs oben beschriebenem Schema und legten damit einen Grundstein für folgende Entwicklungen – alle im folgenden vorgestellten Verfahren nutzen diese Modifikation. Insbesondere beschreiben Richard Lindner und Chris Peikert in einem Paper aus 2011 [LP11] eine Weiterentwicklung dieses Verfahrens. Die Einreichung *Frodo* (§ 4.4) wiederum ist eine Instanziierung dieser Weiterentwicklung.

Die Notation der folgenden Beschreibung basiert auf der in [LP11], allerdings mit Umbenennung von Variablen und einer Nachrichtenlänge von einem Bit zur Angleichung an das oben beschriebene Schema.

Die Schlüsselgenerierung entspricht der oben beschriebenen, allerdings wird der geheime Schlüssel $\vec{s}$ normalverteilt statt gleichverteilt gewählt; diese Modifikation verringert die Schwierigkeit des Problems aber nicht [ACPS09, § 3.2]. Bei der Verschlüsselung wird der selbe Kniff angewendet – zwei ähnliche Werte werden übertragen, wobei auf einen der Geheimtext addiert wird. Es werden weitere Fehlervektoren $\vec{e}_1$ und $\vec{e}_2$ und ein Fehler e_3 normalverteilt gewählt. Der Ciphertext besteht aus den beiden Teilen

$$\vec{c_1} = \vec{e}_1 \cdot \mathsf{A} + \vec{e}_2$$
$$c_2 = \vec{e}_1 \bullet \vec{b} + e_3 + \mathrm{msg} \cdot p/2.$$

Der Empfänger kann $c_2 - \vec{c}_1 \bullet \vec{s} \approx \mathrm{msg} \cdot p/2$ berechnen und so die Nachricht herausfinden. Für Dritte hingegen sind die ausgetauschten Nachrichten auch hier nicht von einer Zufallsverteilung zu unterscheiden.

Lyubashevsky, Peikert, Regev (2010)

In den Folien [LPR10F, F. 8] zum Paper [LPR10], in dem sie Ring-LWE definieren, beschreiben die Autoren auch beispielhaft ein darauf basierendes Kryptosystem. Alle Berechnungen finden in einem Polynomring, typischerweise $\mathcal{R} = \mathbb{Z}_q[X]/(X^n + 1)$, statt, die Funktionsweise ist ansonsten sehr ähnlich zu dem oben beschriebenen System.

Öffentlicher Schlüssel ist hier ein einziges Ring-LWE-Sample: $\mathrm{pk} = (\boldsymbol{a}, \boldsymbol{b} = \boldsymbol{a} \times \boldsymbol{s} + \boldsymbol{e})$; der private Schlüssel $\mathrm{sk} = \boldsymbol{s}$ und $\boldsymbol{e}$ sind Ringelemente mit kleinen (d. h. aus einer Gauß-Verteilung o. ä. gewählten) Koeffizienten; a ist gleichverteilt aus $\mathcal{R}$ gewählt. Bei

der Verschlüsselung werden ebenfalls drei Fehlerwerte zufällig aus einer Gaußverteilung gezogen. Die Bits der Nachricht werden als Koeffizienten eines Polynoms aus $\mathcal{R}$ betrachtet. Der Ciphertext ist

$$c_1 = a \times e_1 + e_2$$
$$c_2 = b \times e_1 + e_3 + msg \cdot q/2.$$

Wie die Farbmarkierungen zeigen, lässt sich auch hier die Sicherheit von Schlüssel und Nachricht auf Ring-LWE zurückführen.

3.1.3 Weitere gitterbasierte Kryptosysteme

NTRU

Es gibt weitere Kryptosysteme, die der Klasse der gitterbasierten Kryptographie zugeordnet werden können, z. B. das NTRU-Kryptosystem von 1998 und dessen Weiterentwicklungen. NTRU-Verfahren sind für Post-Quanten-Verfahren relativ alt und sehr effizient und daher vielversprechende Kandidaten. Zwei der Verfahren (*NTRU* und *NTRU Prime*) haben es in die zweite Runde des NIST-Standardisierungsprozesses geschafft. Diese Klasse soll aber im Rahmen dieser Arbeit nicht weiter betrachtet werden.

3.2 Codes

Ein fehlererkennender oder -korrigierender Code ermöglicht es, gewisse bei der Datenübertragung auftretende Fehler zu erkennen oder sogar zu korrigieren. Das einfachste Beispiel ist das Anhängen eines Paritätsbits an ein (binäres) Codewort. Dieser Code ermöglicht eine Fehlererkennung, wenn angenommen wird, dass höchstens ein Bit fehlerhaft übertragen wird. Eine Fehlerkorrektur ist jedoch nicht möglich.

Robert McEliece entwarf bereits 1978, also in der Zeit, in der auch RSA und das Diffie-Hellman-Verfahren veröffentlicht wurden, ein auf diesen Codes basierendes, asymmetrisches Cryptoverfahren [McE78]. In der Praxis setzten sich aber, vor allem aufgrund wesentlich kleinerer Schlüsselgrößen, RSA und DH durch.

In der folgenden Zeit wurden durchgehend Forschungsarbeiten zum McEliece-Verfahren veröffentlicht; es gelang dabei nicht, für das zugrundeliegende Problem wesentlich effizientere Algorithmen zu finden [cME, § 4.1]. Die genannten Veröffentlichungen betrachteten das Problem allerdings vorwiegend aus dem Blickwinkel der Codierungstheorie; Erst im Zuge der Suche nach Post-Quanten-Verfahren rückte das Verschlüsselungsverfahren von McEliece wieder stärker in den Fokus.

3.2.1 Mathematische Grundlagen

Wir bezeichnen als $[n, k]$-Code über einen Körper $\mathbb{F}$ einen Code, der Wörter aus $\mathbb{F}^k$ als *Codewörter* $\mathbb{F}^n$ kodiert (es gilt $k < n$). Wir betrachten im Folgenden nur *lineare*, *binäre* Codes, das heißt, die Codewörter bilden einen k-dimensionalen Unterraum von $\mathbb{F}^n$; und $\mathbb{F} = \mathbb{B}$. Die folgende Beschreibung basiert auf [Kri15; OS09, § 6; Hof14], alle Vektoren sind Zeilenvektoren.

$[n, k]$-Code

Zu jedem Code gehört eine Generatormatrix $\mathsf{G} \in \mathbb{B}^{k \times n}$, mit der diese Codewörter berechnet werden können: $\vec{c} = \vec{m} \cdot \mathsf{G}$. Wenn die ersten k Spalten von G die Identitätsmatrix sind, d. h., wenn jedes Codewort mit dem Klartext-Wort beginnt, sprechen wir von einem *systematischen* Code.

G

systematisch

Der Hamming-Abstand $\mathrm{d_H}(\vec{x}, \vec{y})$ zweier Wörter in einem Vektorraum ist die Anzahl der Koordinaten, in denen sich die beiden Wörter unterscheiden. Das Gewicht $\mathrm{wt}(\vec{x}) := \mathrm{d_H}(\vec{x}, \vec{0})$ eines Wortes $\vec{x}$ ist definiert als die Anzahl seiner Koordinaten, die nicht null sind. Der Minimalabstand (oft d) eines Codes ist der kleinste Hamming-Abstand zweier verschiedener Codewörter, wir sprechen dann auch von einem $[n, k, d]$-Code.

$\mathrm{d_H}()$

$\mathrm{wt}()$

Minimalabstand

$[n, k, d]$-Code

Wir betrachten nochmals den 1-bit-Paritätscode. Er hat den Minimalabstand 2, denn unterscheiden sich zwei Wörter nur in einem Bit, unterscheiden sich die entsprechenden Codewörter außerdem im Paritätsbit. Außerdem ist er systematisch: das Klartext-Wort wird nicht verändert, es wird nur ein Bit angehangen. Seine Generatormatrix ist die Einheitsmatrix gefolgt von einer Spalte Einsen.

Ein Code ist t-fehlerkorrigierend, wenn ein Decodierer $\mathcal{D}_C$ existiert, so dass für alle Wörter $\vec{m} \in \mathbb{F}^k$ und alle Fehlervektoren $\vec{e} \in \mathbb{F}^n$ gilt:

t-fehlerkorrigierend

$$\mathrm{wt}(\vec{e}) \leq t \quad \Rightarrow \quad \mathcal{D}_C(\vec{m} + \vec{e}) = \vec{m}.$$

Ein Code kann nur t-fehlerkorrigierend sein, wenn $d > 2 \cdot t$.

Ein wichtiges Hilfskonstrukt ist der Orthogonalcode $C^\perp$ eines Codes C. Er ist die Menge aller Vektoren, die orthogonal zu allen Codewörtern $\vec{c} \in C$ sind, und ebenfalls ein Unterraum von $\mathbb{B}^n$. Seine Generatormatrix schreiben wir H, wir nennen sie auch die *Prüfmatrix* von C, sie hat die Größe $(n - k) \times n$. Aus der Orthogonalität folgt folgende wichtige Eigenschaft:

$C^\perp$

H

Prüfmatrix

$$\vec{c} \in C \quad \Leftrightarrow \quad \vec{c} \cdot \mathsf{H}^\top = \vec{0}.$$

Wir können also mit der Prüfmatrix überprüfen, ob ein Codewort korrekt übertragen wurde – das ist genau dann der Fall, wenn $\vec{c} \cdot \mathsf{H}^\top$ der Nullvektor ist.

Das Produkt $\vec{c} \cdot \mathsf{H}^\top$ wird auch *Syndrom* genannt, denn es hat eine sehr hilfreiche Eigenschaft: Alleine das Syndrom genügt, um den Fehlervektor $\vec{e}$ zu berechnen – diese Zuordnung ist unabhängig vom Klartext-Wort. Das *Classic-McEliece*-Verfahren (jedoch nicht das unten beschriebene McEliece-Verfahren) nutzt diese Eigenschaft, um die Nachrichtengröße zu reduzieren.

Syndrom

Wenn ein Code durch seine Generatormatrix G in systematischer Form gegeben ist, kann daraus trivial die zugehörige Prüfmatrix H berechnet werden:

$$\mathsf{G} = \left[\mathbb{1}_k \mid \bar{\mathsf{G}}\right] \quad \Leftrightarrow \quad \mathsf{H} = \left[\bar{\mathsf{G}}^\top \mid \mathbb{1}_{n-k}\right].$$

Die Prüfmatrix des 1-bit-Paritätscodes ist also eine einzeilige Matrix, deren Koeffizienten alle 1 sind.

3.2.2 Grundlegende Verfahren

McEliece (1978)

Das erste Verfahren dieser Art wurde 1978 von Robert McEliece [McE78] veröffentlicht, interessanterweise fast zeitgleich zum RSA-Verfahren.

Als Schlüssel dient ein Goppa-Code $\mathcal{G}$, der t-fehlerkorrigierend ist. Goppa-Codes sind eine Unterklasse von fehlerkorrigierenden Codes, die auf eine bestimmte Art generiert werden; ihr Vorteil ist, dass für sie eine effiziente Decodierungsfunktion existiert. Die Generatormatrix von $\mathcal{G}$ sei $\mathsf{G} \in \mathbb{F}^{k\times n}$. Der Empfänger generiert außerdem eine Permutationsmatrix $\mathsf{P} \in \mathbb{F}^{n\times n}$ und eine invertierbare binäre Matrix $\mathsf{S} \in \mathbb{F}_2^{k\times k}$. Für den öffentlichen Schlüssel wird G verwürfelt: $\mathsf{G}' = \mathsf{SGP}$ (dies ist dann ebenfalls die Generatormatrix eines t-fehlerkorrigierenden Codes), der öffentliche Schlüssel ist (G', t). Der geheime Schlüssel ist $(\mathsf{S}, \mathcal{D}_\mathcal{G}, \mathsf{P})$; $\mathcal{D}_\mathcal{G}$ ist eine Dekodierungsfunktion für $\mathcal{G}$.

Für die Verschlüsselung einer Nachricht $\vec{m}$ wird ein Fehlervektor $\vec{e}$ mit $\mathrm{wt}(\vec{e}) = t$ zufällig gewählt. Der Ciphertext ist $\vec{c} = \vec{m} \cdot \mathsf{G}' + \vec{e}$.

Für die Entschlüsselung wird zunächst die Permutation P rückgängig gemacht ($\vec{c} \cdot \mathsf{P}^{-1}$), auf diesen Wert kann der Dekodierungsalgorithmus $\mathcal{D}_\mathcal{G}$ angewendet werden. Er gibt $\vec{m}' = \vec{m} \cdot \mathsf{S}$ zurück (da $\mathrm{wt}(\vec{e}) \leq t$) und wir können die Nachricht berechnen: $\vec{m} = \mathsf{S}^{-1} \cdot \vec{m}'$.

Niederreiter (1986)

Von Harald Niederreiter [Nie86] stammt eine Weiterentwicklung, in der die Nachricht nicht als Klartext-Wort verwendet, sondern als Fehlervektor codiert wird. Dadurch reicht es aus, statt dem Codewort das Syndrom zu übertragen.

Der Empfänger generiert eine Prüfmatrix $\mathsf{H} \in \mathbb{B}^{(n-k)\times n}$ für einen t-fehlerkorrigierenden $[n, k]$-Code $\mathcal{G}$, und eine zufällige Permutationsmatrix $\mathsf{P} \in \mathbb{B}^{n\times n}$. Er formt HP in die systematische Form um, d. h. so, dass die ersten $n - k$ Spalten die Einheitsmatrix sind. Die Umformungsabbildung sei M, d. h. es gilt $\mathsf{MHP} = \left[\mathbb{1}_{n-k} \mid \overline{\mathsf{H}}\right]$. Der geheime Schlüssel ist $(\mathsf{P}, \mathcal{D}_\mathcal{G}, \mathsf{M})$, der öffentliche Schlüssel ist $(\mathsf{H}' = \mathsf{MHP},\ t)$. Es genügt jedoch, statt H' nur das deutlich kleinere $\overline{\mathsf{H}}$ zu übertragen.

Der Nachrichtentext ist ein Vektor $\vec{e} \in \mathbb{B}^n$ mit $\mathrm{wt}(\vec{e}) = t$. Der Ciphertext ist das Syndrom[4] $\vec{s} = \mathsf{H}' \cdot \vec{e}^\top$. (Das Codewort von $\vec{0}$ ist immer $\vec{0}$ (Linearität), daher entspricht dies der Übertragung von $\vec{0}$ mit einem Übertragungsfehler $\vec{e}$. Anders als oben, wo ein Fehler zufällig ausgewählt werden musste, der auf das Codewort addiert wurde, ist hier kein zufälliges Klartextwort nötig, da das Syndrom unabhängig von der Nachricht bei einem bestimmten Fehler immer gleich ist.)

Der Empfänger kann zunächst die durch M erzeugte Verwürfelung rückgängig machen ($\mathsf{M}^{-1} \cdot \vec{s}$) und dieses Wort dekodieren. Da $\mathrm{wt}(\vec{e}) \leq t$, erhält er $\vec{e}' = \mathsf{P} \cdot \vec{e}$ und kann $\vec{e} = \mathsf{P}^{-1} \cdot \vec{e}'$ berechnen.

3.2.3 Sicherheit

Das Problem *Computational Syndrome Decoding* (CSD) besteht in folgender Aufgabe [OS09, § 3.2]: Gegeben ist ein Code in Form seiner Prüfmatrix $\mathsf{H} \in \mathbb{B}^{r \times n}$, ein Syndrom $\vec{s} \in \mathbb{B}^r$ und ein Gewicht $w > 0$. Gesucht ist ein Wort $\vec{e} \in \mathbb{B}^n$ mit $\mathrm{wt}(\vec{e}) < w$, für das gilt $\vec{e} \cdot \mathsf{H}^\top = \vec{s}$. CSD

Die Schwere des CSD im *Average Case* ist nicht bewiesen, sie wird aber vermutet, unter anderem, weil seit 1978 trotz intensiver Forschung am McEliece-Verfahren keine effizienten Verfahren gefunden wurden [cME, § 4.1]. Für den *Worst Case* ist die Schwere gesichert: In der Entscheidungsform ist das CSD NP-vollständig [BMvT78], die Suchform lässt sich mit linearem Faktor auf die Entscheidungsform reduzieren [OS09, § 3.2].

Das Problem, die Nachricht zu einem McEliece-Ciphertext zu finden, wird gelegentlich McEliece-Problem (MEP) genannt. Gegeben sind $\mathsf{G}', t, \vec{c}$ wie oben, gesucht ist $\vec{m}$. Es ist ähnlich zu CSD. Es gilt $\mathsf{MEP} \leq_p \mathsf{CSD}$ [OS09, § 2.1], die (hier wichtigere) umgekehrte Reduktion kann nur vermutet werden. Sie gälte für allgemeine, zufällige Codes, im McEliece-Verfahren wird aber eine bestimmte (und dem Angreifer bekannte) Klasse von Codes verwendet. Die Vermutung scheint aber wahr zu sein – wie gesagt wurden seit 1978 kaum Fortschritte bekannt.

Die Niederreiter-Variante unterscheidet sich letztlich nur in zwei Details vom ursprünglichen McEliece-Verfahren.

- Die Nachricht wird als Übertragungsfehler codiert anstatt als Nachricht. Dies ist äquivalent: Wer den Fehler kennt, kann die Nachricht berechnen und umgekehrt.
- Es wird eine systematische Prüfmatrix H verwendet. Eine Prüfmatrix lässt sich nicht immer in systematische Form bringen, sondern nur etwa 29 % der Fälle[5]. Auch dies ist unerheblich: Ein Angriff auf das Niederreiter-System ließe sich auch für Angriffe

[4] Dies scheint auf den ersten Blick der obigen Definition des Syndroms als $\vec{c} \cdot \mathsf{H}^\top$ zu widersprechen. Es gilt aber $\vec{s} = \vec{c} \cdot \mathsf{H}^\top \Leftrightarrow \vec{s}^\top = \mathsf{H} \cdot \vec{c}^\top$.

[5] Bei der Schlüsselgenerierung wird in einem solchen Fall eine neue Matrix generiert, sie dauert also bei Niederreiter etwa drei mal so lang wie bei McEliece.

auf etwa jeden dritten McEliece-Schlüssel verwenden. (Beim McEliece-Verfahren werden keine weiteren Bedingungen an den Code gestellt, so dass 29 % der Codes die Niederreiter-Kriterien erfüllen.)

Das Niederreiter-Verfahren ist also mindestens so sicher wie das McEliece-Verfahren [LDW94] und erbt dessen vermutlich sehr starke Sicherheit.

3.3 Weitere Verfahrensklassen

Neben den Gitter- und den Code-basierten Verfahren gibt es noch weitere Klassen von Post-Quanten-Verfahren, die ich der Vollständigkeit halber kurz vorstelle.

Hashbasierte Verfahren

Hashfunktionen, selbst etwas ältere wie SHA-2, sind auch gegen Angriffe mit Quantencomputern sicher (allerdings reduziert der Grover-Algorithmus das Sicherheitsniveau auf die Hälfte; es sind für gleichbleibende Sicherheit also größere Parameter zu wählen). Es ist vermutlich nicht möglich, aus Hashfunktionen asymmetrische Verschlüsselungsverfahren zu konstruieren (da Hashfunktionen sich ja gerade dadurch auszeichnen, nicht invertierbar zu sein), aber für Signaturverfahren eignen sie sich sehr gut.

Bereits von 1979 stammt das Lamport-Diffie-Einmalsignaturverfahren, ein extrem einfaches Verfahren, dessen Sicherheit einzig auf die Urbildresistenz einer Hashfunktion zurückgeführt werden kann (d. h. darauf, dass es schwer ist, ein Urbild zu einem Hashwert zu finden). Darauf aufbauend entwarf Ralph Merkle im selben Jahr ein Baum-artiges Verfahren, mit dem dieses Verfahren zu einem Mehrfachsignaturverfahren erweitert werden kann [BDS09], das ebenfalls beweisbar sicher ist.

Dringend nötig sind Post-Quanten-Signaturen heute nur für die Signatur von Updates, für die Signierung ausgetauschter Daten kann, solange es keine Quantencomputer gibt, ein klassisches Verfahren verwendet werden (eine Vorlaufzeit wie bei der Verschlüsselung ist nicht nötig). Hierfür reichen diese einfachen Verfahren sogar aus, auch wenn sie Nachteile haben wie z. B. große Signaturen und eine begrenzte Zahl von Signaturvorgängen. Es gibt inzwischen aber auch jüngere Weiterentwicklungen mit weniger Einschränkungen, zum Beispiel das Extended Merkle Signature Scheme (XMSS), das kürzlich in [RFC8391] standardisiert wurde.

Multivariate Gleichungssysteme

Diese Verfahren beruhen auf der Schwierigkeit, multivariate (d. h. von mehreren Variablen abhängige), quadratische Gleichungssysteme zu lösen. Für Signaturverfahren ist dieses Problem vielversprechend (mehrere solcher Verfahren haben es in die

zweite Runde des NIST-Standardisierungsprozesses geschafft [NIS19]), aber die Entwicklung eines guten asymmetrischen Verschlüsselungsverfahrens ist bisher nicht gelungen [NW17].

Isogenien zwischen supersingulären elliptischen Kurven

Anders als bei der klassischen Kryptografie über elliptische Kurven, bei denen mit Punkten auf den Kurven gerechnet wird, werden hier bestimmte Abbildungen (Isogenien) zwischen verschiedenen supersingulären elliptischen Kurven betrachtet.

Das Feld ist noch vergleichsweise jung (das erste grundlegende Verfahren ist aus 2011), die Verfahren werden daher von manchen Autor_innen als noch nicht vertrauenswürdig genug eingestuft [NW17]. Nichtsdestotrotz hat sich ein solches Verfahren (SIKE) für die zweite Runde des NIST-Standardisierungsprozesses qualifiziert.

4 Vergleich der Verfahren

In diesem Kapitel werde ich vier ausgewählte quantencomputerresistente Verfahren vorstellen und vergleichen; außerdem stelle ich kurz RSA vor, zum einen zum Vergleich, zum anderen, weil es später für die Konstruktion des Hybrid-Verfahrens benötigt wird.

Zum Auswahlprozess

Die Gründe für die Auswahl von RSA erläutere ich später (§ 5.1.1). Die vier PQ-Verfahren wurden aus allen in der ersten Runde des NIST-Standardisierungsprozesses eingereichten Verfahren ausgewählt. Kriterien waren dabei:

- Sicherheit: für manche Verfahren wurden bereits Schwächen (bis hin zum vollständigen Bruch) bekannt. Hierbei habe ich auch mittlere Schwächen (z. B. geringeres Sicherheitsniveau als behauptet) als Anzeichen für Unreife gewertet und die Einreichung ausgeschlossen.
- Effizienz: bei sehr ähnlichen Verfahren schied das weniger effiziente aus.
- Reife: Viele Einreichungen erweckten den Eindruck, nicht zu Ende entwickelt zu sein (z. B. fehlende Details, kritische Hinweise in Kommentaren) oder eher ein Forschungsbeitrag als ein konkreter, implementierbarer Entwurf zu sein; während bei anderen schon sehr genaue Überlegungen zur Implementierung angestellt wurden.
- Detailliertheit: Einige Spezifikationen sind nur sehr kurz und wenig detailliert, während andere sehr detaillierte Ausführungen zu Hintergründen und der Umsetzung enthalten. Da in den meisten Fällen die verschiedenen Spezifikationen meine Hauptquelle waren, konnte ich Einreichungen mit sehr knappen Spezifikationen nicht berücksichtigen.
- Kommentare: Zu einigen Einreichungen gab es wenig oder kein Feedback z. B. im Diskussionforum des NIST. In einigen Fällen führte dies in Verbindung mit den vorigen beiden Punkten zum Ausschluss.

  Dieses Kriterium ist sehr kritisch zu sehen, da so bestehende Bias verstärkt werden (z. B. ist zu erwarten, dass Einreichungen bekannter Autor_innen stärker beachtet werden); da ich aber mangels vertieftem Vorwissen in diesem Bereich auf die Paper und Kommentare als Quellen angewiesen war, war dies im Rahmen dieser Arbeit nicht anders möglich.
- Organisatorisches: Einige ähnliche Verfahren wurden im Laufe des Prozesses zurückgezogen und als gemeinsame Einreichungen neu eingereicht. Dies geschah

überwiegend erst während der Bearbeitungszeit dieser Arbeit, die betreffenden Verfahren schieden daher auch aus.

- Diversität: Um in dieser Arbeit verschiedene Klassen von Verfahren abzudecken, habe ich ein Code-basiertes sowie drei Verfahren aus verschiedenen LWE-Untergruppen (LWE, Ring-LWE, Modul-LWE) ausgewählt.
- Zeitbedarf: Um den Arbeitsaufwand zu begrenzen, habe ich entgegen dem vorigen Punkt die Klasse der NTRU-basierten Verfahren nicht mit einbezogen.

Es ist anzumerken, dass der Auswahlprozess auch subjektiv war und nicht den Anspruch erhebt, neutral und objektiv zu sein. Positiv ist zu sehen, dass alle vier Verfahren unter denjenigen sind, die vom NIST für die zweite Runde ausgewählt wurden. Es hat sich allerdings später herausgestellt, dass ich das Ziel der Diversität nicht in gewünschtem Maße erreicht habe: Die drei LWE-Verfahren nutzen zwar verschiedene Strukturen, sind sich aber ansonsten in vielen Punkten sehr ähnlich, was sich auch an den Autor_innen zeigt (z. B. sind Joppe Bos und Léo Ducas an allen dreien beteiligt). Insbesondere letzteres hätte ich schon früh erkennen und eine andere Auswahl treffen können.

Zu den Verfahrensbeschreibungen

Alle beschriebenen Verfahren bauen intern auf einem Grundverfahren auf, das lediglich Geheimhaltung (OW, s. § 2.3.1) garantiert; die weitergehende Sicherheit wird durch darauf aufbauende Konstruktionen (sogenannte Transformationen, siehe später in Abschnitt 5.2) erreicht. Ich beschreibe in diesem Kapitel weitestgehend nur die Grundverfahren; Details zu den Transformationen sind mit angegeben, für Grundlagen hierzu sei auf das nächste Kapitel verwiesen.

Der Fokus der Beschreibungen liegt darin, einen Überblick und ein grundlegendes Verständnis der Verfahren zu vermitteln, um einen Vergleich der Verfahren zu ermöglichen. Insbesondere die Algorithmen sind daher hier nur auf einer abstrahierten Ebene angegeben. Für Details wird auf die jeweiligen Spezifikationen verwiesen. Ich habe, um dies zu erleichtern, die Notation weitgehend aus der jeweiligen Spezifikation übernommen.

Steckbriefe Zu jedem Verfahren gibt es einen Steckbrief, in dem die wichtigsten Kennwerte des Verfahrens aufgelistet sind, außerdem findet sich dort eine Beschreibung der Algorithmen als Pseudocode. Nachfolgend einige Erläuterungen zu den Daten:

- Die angegebenen Laufzeiten sind diejenigen der KEMs, da bei den meisten Verfahren nur hierfür Werte angegeben sind. Insbesondere die Werte für die Entschlüsselung dürften bei den KTMs wesentlich kleiner sein[1]. Es sind, wenn nicht anders

[1] siehe z. B. den Unterschied zwischen dem CCA- und dem CPA-KEM von NewHope [NewH, Tab. 5].

angegeben, die Werte des unoptimierten Codes (z. B. ohne Nutzung von Vektorinstruktionen; bei Frodo wird jedoch die AES-Hardwarebeschleunigung genutzt). Die Laufzeiten sind in Tausenden von Prozessorzyklen angegeben.

- Da in der vom NIST vorgegebenen API bei der Entschlüsselung nur der private Schlüssel übergeben wird, bei den KEMs aber auch der öffentliche Schlüssel benötigt wird[2], ist die für den privaten Schlüssel angegebene Größe tatsächlich die von Summe von geheimem und öffentlichem Schlüssel. Die Schlüsselgrößen sind aus historischen Gründen (Kompatibilität mit RSA) in kib, also 1024 bits, angegeben.
- $\Pr \perp$ bezeichnet die Fehlerwahrscheinlichkeit, also die Wahrscheinlichkeit, dass eine korrekt erzeugte Nachricht vom Empfänger nicht entschlüsselt werden kann.
- Die Algorithmen sind möglichst abstrakt gehalten, Details wie die Expansion des eigentlich kommunizierten Seeds zum verwendeten Wert A sind nicht dargestellt.

  Für die Notation siehe Abschnitt 2.4. Insbesondere steht $\xleftarrow{\$}$ für das gleichverteilte Ziehen eines Wertes aus einer Menge; $\xleftarrow{\$} \mathcal{G}()$ steht für das Ziehen aus einer Gauß-artigen Verteilung als Gegensatz zur Gleichverteilung, auch wenn konkret eine leicht andere Verteilung genutzt wird (z. B. Binomialverteilung). Die tatsächlich genutzte Verteilung ist mit angegeben. Sind mehrere Werte genannt ($a, b, c \xleftarrow{\$} \mathbb{N}$), so werden diese jeweils einzeln und unabhängig voneinander aus der Menge gezogen. In vielen Fällen wird zur Optimierung nur ein Seed zufällig gezogen und die restlichen Werte pseudozufällig davon abgeleitet; solche Details sind nicht dargestellt.

4.1 RSA

Das RSA-Verfahren wurde 1978 von Ronald Rivest, Adi Shamir und Leonard Adleman veröffentlicht [RSA78]. Es ist eines der dominierenden asymmetrischen Verschlüsselungsverfahren und ist bis heute ungebrochen, wenn gewisse Randbedingungen beachtet werden. Allerdings ist RSA nicht gegen Quantencomputer resistent, da die Faktorisierung mit dem Shor-Algorithmus effizient durchgeführt werden kann.

Tatsächlich konnte in einer satirischen Einreichung [pqRSA] zum Standardisierungsprozess gezeigt werden, dass RSA für Schlüsselgrößen von 1 Gbit auch Angriffen mit dem Shor-Algorithmus standhält. Dies ist möglich, weil die Rechenzeit für RSA langsamer wächst als die des Shor-Algorithmus. Die Schlüsselgenerierung dauert etwa drei Tage, die Verschlüsselung 12 Minuten und die Entschlüsselung $2^1/_2$ Stunden.

RSA ist in seiner hier beschriebenen Rohform (genannt Lehrbuch-RSA, engl. *textbook RSA*) nur OW-sicher und muss daher immer als KEM (s. § 5.5) oder mit einer Transformation wie OAEP (§ 5.2.1) verwendet werden.

[2] Wegen der im Rahmen der Transformation nötigen *re-encryption* (siehe § 5.2)

RSA

Quantencomputer-Resistenz: Keine

Problem: RSA-Problem (Nachricht), Primfaktorzerlegung (Schlüssel)

Struktur: $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$ $(N \approx 2^k)$

Parameter: Sicherheitsparameter k

Laufzeiten: (kCyc)	**Gen**	**Enc**	**Dec**
$k = 1024$	57 500	130	1070
$k = 2048$	280 000	425	4240
$k = 3072$	810 000	850	11 100
$k = 4096$	2 100 000	1442	23 300

Größen: (kib)	***sk***	***pk***	***ct***	**Pr** $\perp$
(allgemein)	$2k$	k	k	0
$k = 1024$	2	1	1	0
$k = 4096$	8	4	4	0

Transformation: z. B. OAEP (§ 5.2.1) für KTM; RSA-KEM ist IND-CCA (s. § 5.5)

Schlüsselgenerierung:

1: Wähle zufällig zwei Primzahlen p, q, so dass $2^{k-1} < p \cdot q < 2^k$
2: $N \leftarrow p \cdot q$
3: $\phi(N) \leftarrow (p-1)(q-1)$
4: Wähle ein e, so dass $1 < e < \phi(N)$ und $\gcd(e, \phi(N)) = 1$ (oft: $e = 65537$)
5: $d \leftarrow e^{-1} \mod \varphi(N)$
6: **return** $\mathrm{pk} = (N, e)$, $\mathrm{sk} = (N, d)$

Verschlüsselung: $\mathrm{msg} \in \mathbb{Z}_N \mapsto \mathrm{ct} \in \mathbb{Z}_N$

1: **return** $\mathrm{ct} \leftarrow \mathrm{msg}^e \mod N$

Entschlüsselung: $\mathrm{ct} \in \mathbb{Z}_N \mapsto \mathrm{msg} \in \mathbb{Z}_N$

1: **return** $\mathrm{msg} \leftarrow \mathrm{ct}^d \mod N$

Korrektheit: Längerer Beweis, siehe z. B. [RSA78, § VI]

Tabelle 4.1: Steckbrief RSA

Die Sicherheit des Schlüssels lässt sich auf das Faktorisierungsproblem reduzieren [Buc16, § 8.3.4], ein sehr altes und universelles Problem, dessen Schwere deshalb sehr stark angenommen werden kann. Die Aufgabe, zu einem RSA-Ciphertext den Klartext zu berechnen, wird auch RSA-Problem genannt. Dessen Schwere lässt sich nicht auf das Faktorisierungsproblem (oder andere Probleme) reduzieren, sondern wird nur vermutet – aufgrund der langen Geschichte und hohen Verbreitung von RSA kann man sich aber relativ sicher sein, dass es auch hierzu keine effiziente Methode gibt.

RSA-Problem

Die Schlüsselgrößen gelten für einen kleinen öffentlichen Exponenten. Die Laufzeiten wurden durch mich selbst ermittelt und anschließend in Zyklenzahlen umgerechnet. Hierbei wurde die Python-Bibliothek `pyCrypto` genutzt, da sie Zugriff auf unsichere Modi (in diesem Fall: RSA ohne Padding) erlaubt und die mathematischen Operationen durch Auslagerung in eine C-Bibliothek beschleunigt sind. Die Werte sind vermutlich nicht sehr genau, sie sind hier nur zur Orientierung angegeben.

4.2 Classic McEliece

Classic McEliece [cME] ist das einzige codebasierte Verfahren, das ich im Rahmen dieser Arbeit untersuche (es ist aber nicht das einzige zum NIST-Standardisierungsprozess eingereichte codebasierte Verfahren). Classic McEliece sticht insbesondere durch seine lange Geschichte hervor, in der kaum Fortschritte bei der Kryptoanalyse erreicht werden konnten.

Ein großer praktischer Nachteil ist der etwa ein Megabyte große öffentliche Schlüssel und die sehr langsame Schlüsselerzeugung, die auf einem modernen Prozessor um die zwei Sekunden benötigt (die Zeit kann aber mit FPGAs auf 10 bis 20 ms verringert werden). Dieser große Anfangs-Overhead wird jedoch ausgeglichen durch den kleinsten Ciphertext aller Verfahren sowie sehr schnelle Ver-/Entkapselung. Die in Tabelle 4.2 angegeben Laufzeiten sind allerdings (anders als bei den anderen Verfahren) die der optimierten Fassung, mit Nutzung von Vektorinstruktionen.

Trotz des Namens ist Classic McEliece keine Implementierung des Original-Verfahrens von McEliece aus 1978, sondern der Niederreiter-Variante. Die Sicherheit ist, wie bereits in Abschnitt 3.2.3 ausgeführt, äquivalent zu der des McEliece-Verfahrens.

Es werden zwei Parametersätze geboten, die beide Kategorie 5 erreichen. Der Unterschied ist, dass beim größeren Parametersatz $n = q$ gilt, während beim kleineren Satz n kleiner als q ist, was zwei mögliche Angriffsarten erschwere [cME, § 4.1, 8.3]

Classic McEliece ist ein zweistufiges Verfahren, intern wird ein nur IND-CPA-sicheres KTM verwendet, das anschließend in ein IND-CCA-sicheres KEM transformiert wird (zu Transformationen siehe im Detail § 5.2). Eine Besonderheit ist, dass das KTM als Eingabe nicht beliebige Bitstrings akzeptiert, sondern einen Fehlervektor mit vorgegebenem Gewicht. Das KTM ist also in dieser Form nicht universell einsetzbar (eine

Classic McEliece

Problem: CSD

Struktur: Goppa-Code über $\mathbb{B}$

Parameter:	n	k	t
McEliece6960119	6960	5413	119
McEliece8192128	8192	6528	128

Laufzeiten: (kCyc)	**Gen**	**Enc**	**Dec**
McEliece6960119	(*nicht angegeben*)		
McEliece8192128	4 674 803	296	458

Größen: (kib)	**sk**	**pk**	**ct**	**Pr** $\perp$
McEliece6960119	109	8182	1,8	0
McEliece8192128	110	10 608	1,9	0

Transformation: selbst entwickelt; MAC-and-encrypt-artig, mit re-encryption und implicit rejection ($\not\perp$)

Schlüsselgenerierung:

1: $g(X) \xleftarrow{\$} \mathbb{F}_q[X]$ (Grad t, $q = 8192$)
2: $\alpha_1, \ldots, \alpha_n \xleftarrow{\$} \mathbb{F}_q$ (α_i paarw. versch.)
3: Erzeuge aus $(g, \vec{\alpha})$ die Prüfmatrix H eines binären $[n, k]$-Goppa-Codes (für Details siehe [cME, § 2.3])
4: Forme H in die systematische Form um, so dass $\mathsf{H} = [\mathbb{1}_{n-k} \mid \mathsf{T}]$. Ist dies nicht möglich, gehe zu Schritt 1.
5: **return** $\mathrm{pk} = \mathsf{T},\ \mathrm{sk} = \Gamma = (g, \vec{\alpha})$

Verschlüsselung:

$\mathrm{pk}, \vec{e} \in \mathbb{B}^n \mapsto \mathrm{ct} \in \mathbb{B}^{n-k} \qquad (\mathrm{wt}(\vec{e}) = t)$

1: **return** $\mathrm{ct} = \vec{w} = \mathsf{H} \cdot \vec{e}$

Entschlüsselung:

1: $\vec{v} \leftarrow (w_1, \ldots, w_{n-k}, 0, \ldots, 0) \in \mathbb{B}^n$
2: Finde Codewort $\vec{c}$ mit $\mathrm{d_H}(\vec{w}, \vec{c}) \leq t$. Existiert kein solches $\vec{c}$: **return** $\perp$
3: $\vec{e}' \leftarrow \vec{v} - \vec{c}$
4: **if** $\mathrm{wt}(\vec{e}') \neq t$: **return** $\perp$
5: **if** $\mathsf{H} \cdot \vec{e}' \neq \vec{w}$: **return** $\perp$
6: **return** $\vec{e}'$

Korrektheit: Da $\mathrm{wt}(\vec{e}) \leq t$, existiert das gesuchte $\vec{c}$ und es gilt $\vec{e} = \vec{e}'$.

Tabelle 4.2: Steckbrief Classic McEliece

deterministische und umkehrbare Umwandlung eines Bitstrings in einen solchen Fehlervektor ist allerdings möglich, jedoch langsam). Für die Nutzung als KEM ist dies jedoch nicht nötig – bei der Verkapselung kann einfach statt einem zufälligen Bitstring direkt ein zufälliger Fehlervektor gezogen und der Umrechnungsaufwand eingespart werden.

Für die Transformation wird kein etabliertes, bereits veröffentlichtes Verfahren genutzt, sondern ein selbst entworfenes. Die Autor_innen begründen in [cME, § 4.3] die Wahl ihres Verfahrens, bieten aber keinen Beweis.

4.3 Gemeinsamkeiten der gitterbasierten Verfahren

Einige Eigenschaften sind den drei gitterbasierten Verfahren gemein, ich werde sie daher hier gesammelt vorstellen und in den Abschnitten zu den jeweiligen Verfahren nur die Besonderheiten erwähnen.

Der Wert *A* Es ist prinzipiell möglich, den Wert A einmalig (als Verfahrensparameter) festzulegen und für sämtliche Schlüssel diesen Wert zu nutzen – die Reduktion auf das LWE-Problem hängt davon nicht ab. Allerdings hätte dieser Ansatz zwei Nachteile. Zum einen müsste sichergestellt sein, dass der Wert tatsächlich zufällig ist und nicht ein Wert standardisiert wird, zu dem jemand, beispielsweise ein Geheimdienst, eine Hintertür kennt. Außerdem erleichtert ein solch einheitlicher Wert möglicherweise Angriffe, denn so können langwierige Berechnungen für viele Schlüssel genutzt werden statt nur für einen, und sich so amortisieren. Eine ausführlichere Diskussion über verschiedene Alternativen, A zu erzeugen, findet sich in der NewHope-Spezifikation [NewH, S. 17].

In allen vorgestellten LWE-Verfahren wird für jeden Schlüssel ein neuer Wert für A zufällig erzeugt und in den öffentlichen Schlüssel integriert. Um Bandbreite zu sparen, wird nicht A selbst, sondern ein Seed, aus dem A pseudozufällig deterministisch erzeugt werden kann, veröffentlicht.

Rundung Die folgenden Verfahren verwenden das selbe Prinzip wie die in Abschnitt 3.1.2 vorgestellten Basis-Verfahren, nämlich, zwei Werte zu übertragen, in deren (Nicht-)Ähnlichkeit der Klartext codiert ist. Es gibt hier eine Optimierungsmöglichkeit, denn um zu erkennen, ob die beiden Werte nahe beieinander liegen oder weit auseinander, sind nicht die exakten Werte nötig, es genügt, die ungefähren Werte zu kennen.

Reconciliation

Ein besonders effizientes Verfahren hierzu ist als *Reconciliation* bekannt und geht zurück auf Jintai Ding, Xiang Xie und Xiaodong Lin [DXL12] mit einer Verbesserung von Chris Peikert [Pei14]. Hierdurch kann die Ciphertextgröße um fast die Hälfte reduziert werden, da statt des zweiten Werts nur ein einziges Bit gesendet wird. Reconciliation wurde beispielsweise in den ursprünglichen Varianten von NewHope

[ADPS16a] und Frodo [Fro16] verwendet. In den Einreichungsfassungen aller drei LWE-Verfahren wird die Reconciliation jedoch nicht verwendet, da sie zu kompliziert sei und den Ciphertext nur geringfügig kleiner mache als er bei der zweiten Alternative ist:

Bit-Dropping

Bit-Dropping ist die zweite Art, eine solche Optimierung umzusetzen, und wesentlich simpler: Es werden einfach von zu übertragenden Werten nur eine gewisse Zahl (höherwertiger) Bits übertragen. In Experimenten konnten Thomas Pöppelmann und Tim Güneysu zeigen, dass die Fehlerrate erst ab 8 fehlenden Bits (von 13) deutlich ansteigt [PG14, Tab. 1].

Fehlerwahrscheinlichkeit Wie bereits erwähnt (§ 3.1.2), gibt es bei allen LWE-Verfahren eine (sehr geringe) Wahrscheinlichkeit, dass ein korrekt erzeugter Ciphertext nicht entschlüsselt werden kann. Auch wenn es prinzipiell möglich ist, die Parameter so zu wählen, dass keine Fehler auftreten [Fro, § 2.4.3], haben alle drei Verfahren die Parameter so gewählt, dass eine sehr kleine Fehlerwahrscheinlichkeit bleibt.

Diese liegt in allen Fällen weit unter 2^{-128}, ist also für die Praxis irrelevant; es ist wahrscheinlicher, dass ein Angreifer im ersten Versuch den Schlüssel errät (bei Verwendung von AES-128, was als sicher angesehen wird). Die Möglichkeit eines Fehlschlags ist jedoch beim Entwurf der Algorithmen zu berücksichtigen, denn ein Angreifer könnte gezielt entsprechende Ciphertexte erzeugen und aus der Reaktion des Empfängers etwas über den Schlüssel erfahren. Außerdem ist die Fehlerwahrscheinlichkeit bei der Beweisführung zu berücksichtigen.

Coins In bestimmten Fällen ist es erwünscht, dass eine Verschlüsselungsfunktion deterministisch arbeitet. Einer dieser Fälle sind gewisse Transformationen (siehe später, § 5.2), die bei der Entschlüsselung den Klartext erneut verschlüsseln, um Manipulationen am Ciphertext zu erkennen. Andererseits ist es aber auch wichtig (z. B. für die Ununterscheidbarkeit), dass es keine feste Klartext/Ciphertext-Zuordnung gibt.

Diese beiden Anforderungen lassen sich vereinbaren, indem die Funktion zwar Zufallsbits verwendet, aber diese als weiterer Parameter übergeben werden. Es ist dann Aufgabe des Aufrufenden, sicherzustellen, dass geeignete Zufallsbits verwendet werden, damit die Verschlüsselung nach außen hin nicht-deterministisch ist.

Kyber und NewHope sind bereits in der Spezifikation mit einem solchen Parameter spezifiziert, bei Frodo ist eine derartige Anpassung trivial möglich. Der Parameter ist jedoch in den folgenden Algorithmenbeschreibungen der Übersichtlichkeit halber nicht explizit erwähnt.

Sicherheit Die Sicherheit folgt bei allen Verfahren direkt aus der der jeweiligen LWE-Variante. Zur leichteren Erkennbarkeit sind auch hier öffentliche und geheime Werte farblich markiert.

4.4 Frodo

Frodo [Fro] basiert auf klassischem LWE und ist damit eine Reaktion auf die Sorgen, dass die Strukturiertheit von Ring-LWE (oder Modul-LWE) eines Tages zu Angriffen auf darauf basierende Verfahren führen könnte; erreicht aber dennoch eine Schlüsselgröße, die in der Praxis handhabbar ist – von den Autor_innen mit dem Claim *„conservative yet practical“* zusammengefasst. Der Name und der Untertitel *„take off the ring“* sind eine Anspielung auf *Der Herr der Ringe*.

Frodo ist eine zweistufiges Verfahren. Kern ist ein IND-CPA-sicherer KTM (genannt *FrodoPKE*), das mit einer Variante der Transformation $QFO_m^{\perp}$ [HHK17] in einen IND-CCA-sicheren KEM (*FrodoKEM*) umgewandelt wird (für Details zu Transformationen siehe Abschnitt 5.2, im Rest dieses Abschnitts behandele ich nur die Kernfunktion). Nur FrodoKEM ist eine Einreichung zum NIST-Standardisierungsprozess, FrodoPKE hat in diesem Zusammenhang nur den Status einer Hilfsfunktion, ist aber dennoch so spezifiziert, dass es separat verwendet werden kann.

FrodoPKE (im Folgenden: Frodo) ist im Wesentlichen eine Instanziierung des Schemas von Lindner und Peikert [LP11]. Eine erste Version von Frodo wurde 2016 veröffentlicht [Fro16], diese unterscheidet sich aber in einigen Details von der Einreichungsversion.

Die öffentliche Matrix A wird wie bei allen LWE-Verfahren deterministisch aus einem Seed erzeugt. Hierzu wurden zwei Varianten entworfen: Einerseits kann hierfür die XOF cSHAKE() aus der SHA3-Familie verwendet werden. Es kann aber auch AES als Hashfunktion verwendet werden, indem die Ciphertexte bestimmter fester Werte als Matrixelemente gesetzt werden, vergleichbar mit dem Counter Mode; der Seed dient dabei als Schlüssel. Wenn für AES Hardwarebeschleunigung zur Verfügung steht, sind mit dieser Variante sämtliche Funktionen mehr als doppelt so schnell (die Expansion des Seeds zu A ist für die Schlüsselgenerierung, die Ver- und die Entschlüsselung nötig).

Für Frodo sind zwei Parametersätze definiert: **Frodo-640** erlaubt die Verschlüsselung eines $\ell = 128$ bit langen Klartexts und erreicht Kategorie 1 (siehe § 2.1.1). **Frodo-976** kann $\ell = 192$ bit verschlüsseln und erreicht Kategorie 3. Werte, mit denen Kategorie 5 erreicht wird, sind nicht spezifiziert, seien aber möglich.

Bei der Verschlüsselung wird die Nachricht zunächst wie folgt vorbereitet (Frodo.Encode): Jeweils B Bits der Nachricht werden zusammengefasst und in ein Matrixelement codiert. Dabei werden die Werte mit $q/2^B$ multipliziert, um die auftretenden Werte breit gefächert auf den Bereich $[0, q[$ zu verteilen, so dass die Decodierung mit hoher Sicherheit erfolgreich ist. Bei Frodo-640 (wo gilt $B = 2$) treten beispielsweise die Werte $(0, 8192, 16\,384, 24\,576) \in \mathbb{Z}_{32\,768}$ auf. Bei der Entschlüsselung wird die entsprechende Dekodierungsfunktion aufgerufen (Frodo.Decode), hier wird jeder der Eingabewerte auf den nächsten dieser Werte gerundet und so die Variation durch die Fehlerwerte wieder aufgehoben.

Frodo

Problem: LWE

Struktur: $\mathbb{Z}_q$

Parameter:	n	D	$q = 2^D$	B	ℓ
Frodo-640	640	15	32 768	2	128
Frodo-976	976	16	65 536	3	192

Laufzeiten: (kCyc)	**Gen**	**Enc**	**Dec**
Frodo-640-AES	1300	1800	1800
Frodo-976-AES	2700	3600	3600

Größen: (kib)	**sk**	**pk**	**ct**	**Pr** $\perp$
Frodo-640	155	75	76	2^{-149}
Frodo-976	244	122	123	2^{-200}

Fehlerverteilung $\mathcal{G}$**:** aus Wertetabelle, ähnlich gerundeter Gaußverteilung (Koeffizienten $\in [-11 \,..\, +11]$ bzw. $[-10 \,..\, +10]$)

Transformation:
Variante von $\mathsf{QFO}^{\not\perp}_m$ [HHK17]

Schlüsselgenerierung:

1: $\mathsf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times n}$ (via $\text{seed}_{\mathsf{A}} \in \mathbb{B}^{128}$)
2: $\mathsf{S} \xleftarrow{\$} \mathcal{G}(\mathbb{Z}^{n \times 8})$
3: $\mathsf{E} \xleftarrow{\$} \mathcal{G}(\mathbb{Z}^{n \times 8})$
4: $\mathsf{B} \leftarrow \mathsf{A} \cdot \mathsf{S} + \mathsf{E}$ $\quad \mathsf{B} \in \mathbb{Z}_q^{n \times 8}$
5: **return** $\text{pk} = (\text{seed}_{\mathsf{A}}, \mathsf{B}),\ \text{sk} = \mathsf{S}$

Verschlüsselung: $\text{pk}, \text{msg} \in \mathbb{B}^{\ell} \mapsto \text{ct}$

1: $\mathsf{S}' \xleftarrow{\$} \mathcal{G}(\mathbb{Z}^{8 \times n})$
2: $\mathsf{E}' \xleftarrow{\$} \mathcal{G}(\mathbb{Z}^{8 \times n})$
3: $\mathsf{E}'' \xleftarrow{\$} \mathcal{G}(\mathbb{Z}^{8 \times 8})$
4: $\mathsf{M} \leftarrow \mathsf{Frodo.Encode}(\text{msg})$ $\quad \mathsf{M} \in \mathbb{Z}_q^{8 \times 8}$
5: $\mathsf{B}' \leftarrow \mathsf{S}' \cdot \mathsf{A} + \mathsf{E}'$ $\quad \mathsf{B}' \in \mathbb{Z}_q^{8 \times n}$
6: $\mathsf{C} \leftarrow \mathsf{S}' \cdot \mathsf{B} + \mathsf{E}'' + \mathsf{M}$ $\quad \mathsf{C} \in \mathbb{Z}_q^{8 \times 8}$
7: **return** $\text{ct} = (\mathsf{B}', \mathsf{C})$

Entschlüsselung:

1: $\mathsf{M}' \leftarrow \mathsf{C} - \mathsf{B}' \cdot \mathsf{S}$
2: **return** $\text{msg} = \mathsf{Frodo.Decode}(\mathsf{M}')$

Korrektheit:

$$\begin{aligned}\mathsf{M}' &= \mathsf{C} - \mathsf{B}' \cdot \mathsf{S} \\ &= \mathsf{S}'\mathsf{B} + \mathsf{E}'' + \mathsf{M} - \mathsf{S}'\mathsf{A}\mathsf{S} - \mathsf{E}'\mathsf{S} \\ &= \mathsf{S}'\mathsf{A}\mathsf{S} + \mathsf{S}\mathsf{E} + \mathsf{E}'' + \mathsf{M} - \mathsf{S}'\mathsf{A}\mathsf{S} - \mathsf{E}'\mathsf{S} \\ &= \mathsf{S}\mathsf{E} + \mathsf{E}'' + \mathsf{M} - \mathsf{E}'\mathsf{S} \approx_{\mathsf{E},\mathsf{E}',\mathsf{E}''} \mathsf{M}\end{aligned}$$

Tabelle 4.3: Steckbrief Frodo

Das Grundprinzip der Verschlüsselung selbst entspricht vom Prinzip her der dualen Variante (§ 3.1.2), außer dass (wie eben beschrieben) mehrere Nachrichtenbits in ein Matrixelement codiert werden.

Als Fehlerverteilung wird eine Verteilung verwendet, die angenähert einer gerundeten kontinuierlichen Gauß-Verteilung entspricht; sie ist in einer Wertetabelle gespeichert. Es treten Werte von $-10 \ldots +10$ (Frodo-640) bzw $-11 \ldots +11$ (Frodo-976) auf. Die Fehlerwahrscheinlichkeit beträgt mit den gewählten Parametern etwa 2^{-149} bzw 2^{-200}.

Die Sicherheit lässt sich direkt auf LWE zurückführen, wie an der farblichen Hervorhebung im Steckbrief einzusehen ist. Frodo basiert dem Schema von Lindner und Peikert, was wiederum eine Weiterentwicklung der dualen Variante des Regev-Schemas ist, einige weitere Anmerkungen zur Sicherheit finden sich im Abschnitt zu diesen Schemata (§ 3.1.2, S. 19).

4.5 NewHope

NewHope [NewH] benutzt Ring-LWE und dürfte eines der bekannteren PQ-Verfahren sein, da es bereits 2016 experimentell in Google Chrome eingesetzt wurde [Böc16].

In der ursprünglichen Veröffentlichung (*NewHope-Usenix*, [ADPS16a]) wurde Reconciliation [DXL12; Pei14] genutzt. Die Reconciliation sei aber relativ kompliziert, ohne nennenswert Bandbreite zu sparen [Fro, S. 16], daher veröffentlichten die Autoren etwas später mit *NewHope-Simple* eine Variante ohne Reconciliation [ADPS16b]. Hier wird stattdessen der zweite Teils des Ciphertexts komprimiert: $\mathsf{Compress}_3(v)$ bildet jeden Koeffizienten v_i auf $\lfloor v_i \cdot 8/q \rceil$ ab, speichert also nur die $d = 3$ höchstwertigen Bits jedes Koeffizienten[3]. Die Einreichungsvariante basiert auf NewHope-Simple, nutzt also dieses Bit-Dropping.

Auch NewHope besteht aus einem (nur intern genutzten) IND-CPA-sicheren KTM und einem darauf aufbauenden IND-CCA-sicheren KEM. Die genutzte Transformation ist exakt die selbe wie bei Frodo (eine Variante von $QFO^{\not\perp}_m$ [HHK17]). Es ist außerdem ein zweites, nur IND-CPA-sicheres, KEM spezifiziert, dieses ist aber für diese Arbeit nicht relevant.

Der verwendete Polynomring ist $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$, dabei ist $n = 512$ bzw. 1024 (je nach Parametersatz) und $q = 12289$. Alle Werte sind Polynome aus diesem Polynomring, repräsentiert durch n Koeffizienten aus $\mathbb{Z}_q$, sie werden fett geschrieben: $\boldsymbol{r} \in \mathcal{R}_q$.

[3] Jedoch mit Rundung statt simplem Verwerfen der Bits. In der NewHope-Spezifikation wird dieser Wert nicht benannt, da er durchgehend 3 ist, der Name d ist von Kyber übernommen.

NewHope

Problem: Ring-LWE

Struktur: $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$

Parameter:	n	q	d
NewHope512	512	12 289	3
NewHope1024	1024	12 289	3

Laufzeiten: (kCyc)	**Gen**	**Enc**	**Dec**
NewHope512	117	181	206
NewHope1024	245	377	437

Größen: (kib)	**sk**	**pk**	**ct**	**Pr** $\perp$
NewHope512	14,8	7,3	8,8	2^{-213}
NewHope1024	28,8	14,3	17,3	2^{-216}

Fehlerverteilung $\mathcal{G}$**:**
Binomialvert. ψ_8^n (Koeff. $\in [-8 \dots +8]$)

Transformation:
Variante von $\mathsf{QFO}_m^{\not\perp}$ [HHK17]

Schlüsselgenerierung:

1: $\hat{a} \xleftarrow{\$} \mathcal{R}_q$ (via $\text{seed}_a \in \mathbb{B}^{256}$)
2: $s, e \xleftarrow{\$} \mathcal{G}(\mathcal{R}_q)$
3: $\hat{b} \leftarrow \hat{s} \times \hat{a} + \hat{e}$
4: **return** $\text{pk} = (\text{seed}_a, \hat{b}),\ \text{sk} = \hat{s}$

Verschlüsselung:
$\text{pk}, \text{msg} \in \mathbb{B}^{256} \mapsto \text{ct}$

1: $s', e', e'' \xleftarrow{\$} \mathcal{G}(\mathcal{R}_q)$
2: $m \leftarrow \mathsf{Encode}(\text{msg})$
3: $\hat{u} \leftarrow \hat{s'} \times \hat{a} + \hat{e'}$
4: $v \leftarrow \mathsf{NTT}^{-1}(\hat{s'} \times \hat{b}) + e'' + m$
5: **return** $\text{ct} = (\hat{u}, \mathsf{Compress}_d(v))$

Entschlüsselung:

1: $v' = \mathsf{Decompress}_d(v)$
2: $m' = v' - \mathsf{NTT}^{-1}(\hat{u} \times \hat{s})$
3: **return** $\text{msg} = \mathsf{Decode}(m')$

Korrektheit:

$$\begin{aligned} m' &= v' - u \times s \approx_d v - u \times s \\ &= s' \times b + e'' + m - s' \times a \times s + e' \times s \\ &\approx_{e,e',e''} s' \times s \times a + m - s' \times a \times s \\ &= m \end{aligned}$$

Tabelle 4.4: Steckbrief NewHope

Die Addition erfolgt komponentenweise, die Multiplikation ist jedoch recht aufwändig, sie erfordert n^2 Skalarmultiplikationen. In NewHope werden die Polynome daher zur Multiplikation mit der NTT (engl. *Number Theoretic Transform*), einem Spezialfall der schnellen Fouriertransformation, in die NTT-Domäne transformiert, in der die Multiplikation schneller ausgeführt werden kann. Insgesamt (inklusive der Transformationen) reduziert sich die Komplexität auf $\mathcal{O}(n \log n)$. Diese Optimierung ist mehr als nur ein Implementationsdetail, denn um Transformationen einzusparen, werden manche Werte in der NTT-Domäne übertragen; die NTT ist also Teil des Protokolls.

Auch bei der Wahl der Parameter musste hierauf Rücksicht genommen werden: Damit die NTT genutzt werden kann, muss n eine Zweierpotenz und q prim sein, außerdem muss gelten $q \equiv 1 \pmod{2n}$. Aus diesen Einschränkungen folgend gibt es nur zwei Parametersätze: NewHope512 (Ringdimension $n = 512$, erreicht Kategorie 1) und NewHope1024 ($n = 1024$, Kategorie 5). Eine dazwischen liegende Dimension, womit Kategorie 3 erreicht werden könnte, ist wegen der NTT nicht möglich.

In den Algorithmenbeschreibungen (Tab. 4.4) werden Werte aus der NTT-Domäne mit Dach dargestellt: $\hat{\boldsymbol{a}} = \mathsf{NTT}(\boldsymbol{a})$. Für das Verständnis des Algorithmus' ist dies aber nicht relevant. Die Transformationen werden nur explizit angegeben, wenn es für das Verständnis nötig ist. Um die NTT-Berechnungen in-place durchführen zu können, sind außerdem noch Bitumkehrungen nötig. Diese sind in den Algorithmen in [NewH] explizit mit angegeben, werden aber hier als Implementierungsdetail betrachtet und der Übersichtlichkeit wegen nicht explizit mit angegeben. Gleichverteilte Zufallswerte können direkt in der NTT-Domäne gezogen werden, damit wird eine Transformation eingespart.

Die verwendete Fehlerverteilung ist eine Binomialverteilung ψ_k. Diese entspricht nur angenähert einer (gerundeten) Gaußverteilung, wie LWE eigentlich verlangt, die Sicherheit ist aber trotzdem etwa äquivalent [NewH, Thm. 4.1]. Vorteil der Binomialverteilung ist insbesondere, dass sie sehr effizient und zuverlässig[4] aus gleichverteilten Zufallsbits berechnet werden kann ($\psi_k = \sum_{i=0}^{k} b - b';\ b, b' \xleftarrow{\$} \mathbb{B}$). Für alle Parametersätze ist $k = 8$, die Fehler streuen also im Bereich $[-8 \ldots +8]$. Die Fehlerwahrscheinlichkeit liegt bei beiden Parametersätzen weit unter 2^{-200}.

Der öffentliche Parameter $\vec{\boldsymbol{a}}$ wird auch hier deterministisch aus einem Seed erzeugt, der öffentliche Schlüssel enthält nur den Seed.

Die Verschlüsselung funktioniert ansonsten exakt wie bei LPR10 (§ 3.1.2), allerdings mit mehrfacher Übertragung jedes Nachrichtenbits: Da die Nachrichtenlänge 256 bit ist, die Polynomdimension aber 512 bzw. 1024, wird jedes Nachrichtenbit in zwei bzw. vier Koeffizienten kodiert; beispielsweise ist (für $n = 512$) $\mathsf{Encode}(0^{254}11) = (X^{257} + X^{256} + X + 1) \cdot q/2$. Bei der Dekodierung wird die durchschnittliche Abweichung der zwei bzw. vier Koeffizienten von $q/2$ betrachtet. Hierdurch sinkt die Wahrscheinlichkeit einer fehlerhaften Übertragung.

[4] Bezogen auf Seitenkanalangriffe, [Kyb, S. 11]

Die Bandbreiten- und Rechenzeit-Werte im Steckbrief sind zur besseren Vergleichbarkeit die des CCA-KEM; auch wenn in der Spezifikation Werte des CPA-KEM angegeben sind, die näher an denen des CPA-KTM liegen dürften. Die Zeitwerte stammen von der Laufzeit der Referenzimplementierung ohne Vektorinstruktionen [NewH, Tab. 5].

4.6 Kyber

Kyber [Kyb] basiert auf Modul-LWE und hat etwa die selben Nachrichtengrößen und Laufzeiten wie NewHope (Ring-LWE, s. o.), aber möglicherweise weniger Angriffsvektoren (siehe den Abschnitt zu Modul-LWE auf Seite 19). Kyber ist Teil von CRYSTALS („*Cryptographic Suite for Algebraic Lattices*“), einer Suite von PQ-Verfahren, zu der mit CRYSTALS-Dilithium auch ein Signaturverfahren gehört, das ebenfalls auf Modul-Gittern basiert.

Der genutzte Polynomring ist $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$. Die Ringdimension n ist gleichzeitig auch die Nachrichtenlänge, die Moduldimension k beträgt je nach Parametersatz 2, 3 oder 4. Das Produkt $n \cdot k$ liegt also zwischen 512 und 1024, dieser Wert bestimmt den Namen des Parametersatzes. (Hier sieht man auch die Ähnlichkeit zu NewHope: Ring-LWE kann als Modul-LWE mit Moduldimension $k = 1$ betrachtet werden; bei NewHope ist $n = 512$ bzw. 1024, die Werte für $n \cdot k$ sind also bei selbem Sicherheitsniveau identisch.)

Die Werte, mit denen gerechnet sind, sind Vektoren (mit Dimension $k = 2$ bis 4, geschrieben $\vec{v}$), deren Elemente Polynome mit einem Grad von höchstens n sind ($\boldsymbol{s} \in \mathcal{R}_q = s_0 + s_1 X + \cdots + s_{n-1} X^{n-1}$).

Kyber verwendet wie NewHope die schnelle Fouriertransformation (NTT) zur effizienten Berechnung von Produkten im Ring (für Details siehe dort). Die NTT schränkt die Parameterwahl auch hier ein, so muss n eine Zweierpotenz sein und gelten, dass $q \equiv 1 \pmod{2n}$. Der gewählte Wert für q ist der (für das gewählte n) kleinstmögliche, er liegt außerdem knapp unter 2^{13}, so dass die Koeffizienten recht effizient in 13 Bits gespeichert werden können. Trotz dieser Einschränkungen kann Kyber (im Gegensatz zu NewHope) drei Parametersätze für alle drei NIST-Kategorien bieten [Kyb, Tab. 1], da die Moduldimension k frei gewählt werden kann.

Im Gegensatz zu NewHope werden aber alle Werte, die im Ciphertext enthalten sind, in der normalen (nicht-NTT-) Darstellung übertragen, so dass die NTT hier als Implementierungsdetail betrachtet werden kann und in der Algorithmenbeschreibung nicht dargestellt wird. Lediglich der Wert $\vec{s}$ wird in der NTT-Domäne im geheimen Schlüssel gespeichert, was aber nur wichtig ist, wenn geheime Schlüssel zwischen Implementierungen ausgetauscht werden können sollen.

Der Wert für $\mathbf{A}$ wird aus einem (für jeden Schlüssel neu generierten) Seed berechnet. Der öffentliche Schlüssel enthält nur diesen Seed.

Kyber

Problem: Modul-LWE

Struktur: $\mathcal{R}_q^k = (\mathbb{Z}_q[X]/(X^n+1))^k$
$n = 256,\ q = 7681$ (prim), $k \in \{2,3,4\}$

Parameter:	n	k	$n \cdot k$	η
Kyber512	256	2	512	5
Kyber768	256	3	768	4
Kyber1024	256	4	1024	3

Laufzeiten: (kCyc)	**Gen**	**Enc**	**Dec**
Kyber512	142	205	246
Kyber768	243	333	394
Kyber1024	369	481	559

Größen: (kib)	sk	pk	ct	**Pr** $\perp$
Kyber512	12,8	5,8	6,3	2^{-145}
Kyber768	18,8	8,5	9,0	2^{-142}
Kyber1024	24,8	11,3	11,8	2^{-169}

Fehlerverteilung $\mathcal{G}$:
Binomialverteilung $B_\eta^{n\times k}$
(Koeffizienten $\in [-\eta \dots +\eta]$)

Schlüsselgenerierung:

1: $\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k\times k}$
2: $\vec{s}, \vec{e} \xleftarrow{\$} \mathcal{G}(R_q^k)$
3: $\vec{t} \leftarrow \mathbf{A} \cdot \vec{s} + \vec{e}$
4: $\text{pk} \leftarrow (\text{Compress}_{11}(\vec{t}),\ \text{seed}_{\mathbf{A}})$
5: $\text{sk} \leftarrow \hat{\vec{s}}$

Verschlüsselung:
$\text{pk}, \text{msg} \in \mathbb{B}^{256} \mapsto \text{ct}$

1: $\vec{r}, \vec{e}_1 \xleftarrow{\$} \mathcal{R}_q^k$
2: $e_2 \xleftarrow{\$} \mathcal{R}_q$
3: $m \leftarrow \text{Encode}(\text{msg})$
4: $\vec{u} \leftarrow \mathbf{A} \cdot \vec{r} + \vec{e}_1$
5: $v \leftarrow \vec{t} \bullet \vec{r} + e_2 + m$
6: $\text{ct} \leftarrow (\text{Compress}_3(\vec{u}), \text{Compress}_{11}(v))$

Entschlüsselung:

1: $v' = \text{Decompress}_3(\text{ct}_1)$
2: $\vec{u}' = \text{Decompress}_{11}(\text{ct}_2)$
3: $m' \leftarrow v' - \vec{s} \bullet \vec{u}'$
4: $\text{msg} \leftarrow \text{Decode}(m')$

Korrektheit:

$$\begin{aligned} m' &\approx \vec{t} \bullet \vec{r} + e_2 + m - \vec{s} \bullet \vec{u} \\ &= \mathbf{A} \cdot \vec{s} \bullet \vec{r} + \vec{e} \bullet \vec{r} + e_2 + m \\ &\quad - \vec{s} \cdot \mathbf{A} \bullet \vec{r} - \vec{s} \bullet \vec{e}_1 \\ &= \vec{e} \bullet \vec{r} + \vec{e}_2 + m - \vec{s} \bullet \vec{e}_1 \\ &\approx m \end{aligned}$$

Tabelle 4.5: Steckbrief Kyber

Die Rundung (*bit-dropping*) wird von der Funktion $\mathsf{Compress}_d$ implementiert. Diese bildet einen Wert (bzw. bei Übergabe eines Ring-/Modulelements dessen Koeffizienten) aus $\mathbb{Z}_q$ auf einen d-Bit-Wert ab. Die Funktion $\mathsf{Decompress}_d$ berechnet daraus wieder einen Wert aus $\mathbb{Z}_q$, der in der Nähe des ursprünglichen Wertes liegt. Beispielsweise ist $\mathsf{Decompress}_2(\mathsf{Compress}_2(0{,}65 \cdot q)) = 0{,}75 \cdot q$. (Anmerkung: In [Kyb] werden die Funktionen $\mathsf{Compress}_q(\cdot, d)$ geschrieben.)

Diese Funktionen können auch unverändert für die Codierung der Nachricht als Ringelement genutzt werden. Es ist $\mathsf{Encode} = \mathsf{Decompress}_1$ und $\mathsf{Decode} = \mathsf{Compress}_1$. Es wird dabei jeweils ein Bit der Nachricht zu einem Koeffizient des Polynoms codiert (0 zu 0 und 1 zu $q/2$). Bei der Entschlüsselung werden Werte um 0 bzw q wieder zu einem Nullbit und Werte um $q/2$ zu einem Einsbit.

Als Transformation wird $U^{\perp}_m[T]$ aus [HHK17] genutzt, jedoch werden zur Erhöhung der Robustheit wesentlich mehr Werte in die Hashes einbezogen.

Die verwendete Fehlerverteilung ist eine Binomialverteilung B_η. Diese entspricht nur angenähert einer (gerundeten) Gaußverteilung, wie LWE eigentlich verlangt, die Sicherheit ist aber trotzdem etwa äquivalent (ein Beweis fehlt in der Kyber-Spezifikation, findet sich aber in [NewH, Thm. 4.1]). Vorteil der Binomialverteilung ist insbesondere, dass sie sehr effizient und zuverlässig[5] aus Zufallsbits berechnet werden kann ($B_\eta = \sum_{i=0}^{\eta} b - b';\ b, b' \xleftarrow{\$} \mathbb{B}$). Die Fehler streuen also im Bereich $[-\eta \ldots +\eta]$, der Wert für η variiert zwischen den verschiedenen Parametersätzen. Die Funktion CBD erweitert diese Verteilung auf Ringelemente: $\mathsf{CBD}_\eta = f_0 + f_1 X + \cdots + f_{255} X^{255}$ (mit $f_i \xleftarrow{\$} B_\eta$).

Für die Expansion von Zufallswerten wird SHA3 bzw. dessen XOF SHAKE benutzt.

4.7 Vergleich

Ein Vergleich der grundsätzlichen Unterschiede (insbesondere der zugrundeliegenden Probleme) habe ich bereits in der Einführung zu den Problemen gezogen (§ 3.1.1, S. 18 und § 3.1.2, S. 19). Die restlichen Implementierungsdetails sind in vielen Fällen austauschbar (z. B. die Transformationen; auch könnte der Ansatz von Frodo, AES als Hashfunktion für die Expansion der Seeds einzusetzen, bei den anderen Verfahren angewendet werden). Ich stelle daher in Tabelle 4.6 nur nocheinmal die Performancedaten gut vergleichbar nebeneinander.

Die Größen sind auf ganze kibit gerundet, die Laufzeiten auf zwei bis drei signifikante Stellen. Von Frodo sind nur die cSHAKE-Werte gut mit denen der anderen Verfahren vergleichbar, da bei der AES-Variante AES-Hardwareinstruktionen genutzt werden. Die Einordnung von RSA in die NIST-Kategorien folgt [Sma+18].

[5] Bezogen auf Seitenkanalangriffe, [Kyb, S. 11]

Verfahren	**Struktur**	**Laufzeiten** (kCyc)			**Größen** (kibit)			**Pr** $\perp$
		Gen	**Enc**	**Dec**	**sk**	**pk**	**ct**	
Kategorie 1								
RSA-3072		810 000	850	11 100	6	3	3	0
Frodo-640-AES	LWE	1 300	1 800	1 800	155	75	76	2^{-149}
Frodo-640-cSHAKE	LWE	8 300	9 100	9 100	155	75	76	2^{-149}
NewHope512	Ring-LWE	117	181	206	15	7	9	2^{-213}
Kyber512	Modul-LWE	142	205	246	13	6	6	2^{-145}
Kategorie 3								
Frodo-976-AES	LWE	2 700	3 600	3 600	244	122	123	2^{-200}
Frodo-976-cSHAKE	LWE	17 800	19 300	19 300	244	122	123	2^{-200}
Kyber768	Modul-LWE	243	333	394	19	9	9	2^{-142}
Kategorie 5								
RSA-15360		260 000 000	19 800	91 800	30	15	15	0
McEliece8192128	Code	4 700 000	296	458	109	10 600	2	0
NewHope1024	Ring-LWE	245	377	437	29	14	17	2^{-216}
Kyber1024	Modul-LWE	369	481	559	25	11	12	2^{-169}

Tabelle 4.6: Vergleich der wichtigsten Kennzahlen der Verfahren

5 Entwurf eines Hybrid-Verfahrens

Wie in der Einleitung bereits erwähnt, sind viele der Post-Quanten-Verfahren noch nicht gut erprobt und wenig beforscht, so dass sie weniger Vertrauen genießen als klassische Verfahren. Es ist jedoch aufgrund des Fortschritts in der Entwicklung von Quantencomputern dringend nötig, zeitnah auf Post-Quanten-Verfahren umzusteigen. Andernfalls besteht das Risiko, dass Daten, die heute verschlüsselt werden, schon in wenigen Jahren von Dritten entschlüsselt werden können.

Eine Möglichkeit, diesen Konflikt zu lösen, besteht darin, die Daten sowohl mit einem etablierten als auch einem gegen Quantencomputer resistenten Verfahren zu verschlüsseln. Ein Angreifer muss dann beide Verfahren angreifen, um an den Klartext zu gelangen. Hat er nur einen genügend großen Quantencomputer zur Verfügung, kann er zwar das klassische Verfahren brechen, scheitert aber am Post-Quanten-Verfahren. Falls hingegen gravierende Schwächen im Post-Quanten-Verfahren entdeckt werden, ist das Hybrid-Verfahren zwar nicht mehr gegen Quantencomputer resistent, aber es bleibt zumindest die Verschlüsselung mit dem klassischen Verfahren. Verglichen mit dem weiteren ausschließlichen Einsatz von klassischen Verfahren erreicht man also mindestens die selbe Sicherheit.

Begriffsklärung Ich verwende den Begriff „hybrid“ in dieser Arbeit in der Regel für die Kombination eines klassischen mit einem Post-Quanten-Verfahren. Traditionell wird darunter die Kombination eines symmetrischen und eines asymmetrischen Verfahrens verstanden. Der Begriff hat sich trotz der Mehrdeutigkeit im PQC-Kontext etabliert [NW17; Moo17; NIS16]. „hybrid“

5.1 Variantendiskussion

Es gibt eine Vielzahl von Möglichkeiten, ein solches hybrides Verfahren zu entwerfen. Im Folgenden werde ich die verschiedenen Varianten diskutieren und bewerten. Anschließend wähle ich eine der Möglichkeiten aus, um an ihrem Beispiel ein hybrides Verfahren konkret umzusetzen.

Hinsichtlich des Einsatzzwecks lassen sich die folgenden beiden Kategorien unterscheiden:

- **interaktive** oder **online**-Verschlüsselung: Die Kommunikationspartner interagieren in Echtzeit miteinander, d. h. sie können, wenn nötig, mehrere Nachrichten austauschen, deren Inhalt auch von vorigen Nachrichten abhängen kann. Meist ist wichtig, dass Nachrichten kurz sind und der Rechenaufwand gering ist. Typische Beispiele: TLS, SSH.
- **nicht-interaktive** oder **offline**-Verschlüsselung: Der Datenaustausch zwischen den Kommunikationspartnern erfolgt ohne zeitlichen Zusammenhang. In der Regel müssen zwei Nachrichten genügen: Zunächst die Veröffentlichung eines (generischen, d. h. nicht für den Kommunikationspartner spezifischen) öffentlichen Schlüssels durch den Empfänger; dann Kommunikation des Ciphertexts vom Absender zum Empfänger. Rechenzeit und Nachrichtengröße sind oft weniger stark beschränkt. Typische Beispiele: E-Mail-Verschlüsselung.

Bei der Auswahl lege ich den Fokus auf flexible Varianten, die sowohl für eine offline- als auch für eine interaktive Verschlüsselung geeignet sind.

5.1.1 Klassisches Verfahren

Von den klassischen Verfahren sind die folgenden Verfahren verbreitet und erprobt [BSI2102]. Da das Hybrid-Verfahren möglichst lange sicher sein soll, aber das klassische Verfahren die Sicherheit alleine bieten muss, falls Schwächen im PQ-Verfahren entdeckt werden, ist der Einsatz neuerer, weniger erprobter klassischer Verfahren nicht sinnvoll.

- **RSA** (Siehe Beschreibung in Abschnitt 4.1.)
- **Diffie-Hellman-Schlüsselaustausch (DH)** Der DH-Schlüsselaustausch ist besonders für seine kurzlebige Variante (DHE, von engl. *ephemeral*: kurzlebig, flüchtig) bekannt (die beispielsweise bei TLS inzwischen stark verbreitet ist, weil sie Perfect Forward Secrecy bietet), kann aber auch mit statischen Schlüsseln verwendet werden [BSI2102, § 3.4]. Für manche Einsatzzwecke (insbesondere interaktive Verschlüsselung) kann DHE benutzt werden, für offline-Verschlüsselung muss stattdessen die statische Variante verwendet werden.

  DH kann statt über eine Gruppe wie $\mathbb{Z}_q$ über beliebige Gruppen implementiert werden, insbesondere über elliptische Kurven (**ECDH / ECDHE**). Da hierbei die Nachrichten- und Schlüsselgröße bei gleichbleibendem Sicherheitslevel wesentlich kleiner ist als bei klassischem DH über $\mathbb{Z}_q$ bzw. bei RSA [BSI2102], ist dies die wesentlich weiter verbreitete Variante.

RSA hat den Nachteil größerer Schlüssel und einer recht langsamen Schlüsselerzeugung.

Für den vorliegenden Einsatzzweck ist aber noch ein anderes Kriterium wichtig: Falls Schwächen im PQ-Verfahren entdeckt werden, basiert die Sicherheit des hybriden Ver-

fahrens nur auf der des klassischen Verfahrens. Es ist also sinnvoll, das klassische Verfahren so zu wählen, dass es möglichst gut gegen Quantencomputer resistent ist, also nur von Quantencomputern mit einer hohen Anzahl von Qubits gebrochen werden kann.

Genaue Qubit-Zahlen sind hier nicht verfügbar – da es noch keine Quantencomputer in dieser Größe gibt, gibt es nur Schätzungen und Simulationen; außerdem hängt die genaue Zahl von der Implementierung ab. Sicher ist, dass bei sehr großen Sicherheitsleveln die Lösung des Faktorisierungsproblems (RSA) mehr Qubits benötigt als die die Berechnung des diskreten Logarithmus auf elliptischen Kurven (ECDH) [Lan17, Folie 16; RNSL17].

Da die restlichen Fragen (Ist Perfect Forward Secrecy erwünscht? Wie wichtig sind Schlüsselgröße und Schlüsselerzeugungszeit?) vom Einzelfall abhängen, es hier aber keinen konkreten Anwendungsfall gibt, habe ich entschieden, den letzten Punkt den Ausschlag geben zu lassen und RSA einzusetzen. Diese Entscheidung liegt vor allem in dem Wunsch begründet, ein konkretes und vollständig spezifiziertes hybrides Verfahren entwerfen zu können, soll aber keine Einzelfallentscheidung vorwegnehmen.

Schlüsselgröße Betrachtet man nur die klassische Sicherheit für etwa die nächsten zehn Jahre, werden Schlüsselgrößen (bei RSA zu verstehen als die Größe des Moduls N) ab 3000 bit empfohlen [BSI2102; Sma+18]. Zieht man auch die Gefahr von Schwächen im PQ-Verfahren mit in Betracht, ist ein weitaus größerer RSA-Modul zu empfehlen. Da nicht vorhersagbar ist, wann Quantencomputer mit einer bestimmten Anzahl an Qubits verfügbar sind, kann hier keine genaue Zahl angegeben werden, er sollte im Zweifelsfall so groß wie möglich gewählt werden. Da die Schlüsselgröße aber nur ein Parameter des Verfahrens ist, muss sie hier nicht abschließend festgelegt werden.

5.1.2 Post-Quanten-Verfahren

Die in dieser Arbeit untersuchten Post-Quanten-Verfahren sind absichtlich recht verschieden, um für den konkreten Anwendungsfall ein passendes Verfahren auswählen zu können.

Um diese Möglichkeit auch für das Hybrid-Verfahren zu erhalten, soll hier keine Vorauswahl getroffen werden. Stattdessen werde ich das Hybrid-Verfahren möglichst generisch gestalten, so dass für die konkrete Implementierung möglichst jedes der Verfahren eingesetzt werden kann. Es gibt dabei jedoch teilweise praktische Einschränkungen, weshalb nicht in jedem Fall jedes Verfahren verwendet werden kann, Details dazu folgen weiter unten.

5.1.3 Kombination der Verfahren

Hat man sich für zwei Verfahren entschieden, bleibt noch die Entscheidung, wie die beiden Verfahren zu einem zusammengesetzt werden sollen. Die Kombination soll derart gestaltet sein, dass auch falls eines der beiden Verfahren gebrochen ist (sei es durch Entwicklung von Quantencomputern oder weil im Post-Quanten-Verfahren Schwächen bekannt werden), die Nachricht trotzdem noch durch das zweite Verfahren geschützt wird.

Verschachtelung

Eine Möglichkeit ist, den Klartext zunächst mit dem Post-Quanten-Verfahren zu verschlüsseln und anschließend den gesamten entstandenen Ciphertext mit dem klassichen Verfahren zu verschlüsseln. Als Formel ausgedrückt: $\mathsf{enc}_{\mathrm{hy}}(m) := \mathsf{enc}_{\mathrm{kl}}(\mathsf{enc}_{\mathrm{pq}}(m))$. Eine umgekehrte Schachtelung ist natürlich genauso möglich.

Dieses Vorgehen hat offensichtlich den Vorteil, dass ein Angreifer, der nicht über einen Quantencomputer verfügt, schon an der „äußersten Schale“ scheitert – es ist ihm nicht möglich, den PQ-Ciphertext zu analysieren (und eventuelle Schwächen auszunutzen); bei umgekehrter Schachtelung kann ein Angreifer, selbst wenn er über einen Quantencomputer verfügt, diesen nur einsetzen, falls eine Schwäche im PQ-Verfahren bekannt geworden ist, mit der er Zugriff auf den klassischen Ciphertext erhält. Wir werden jedoch sehen, dass auch andere Kombinationsverfahren diesen Vorteil bieten.

Es gibt jedoch einige Nachteile. Ein praktischer Nachteil kann sein, dass die beiden Ver- oder Entschlüsselungsoperationen jeweils nicht parallelisierbar sind. Gravierender ist jedoch, dass der Ciphertext des inneren Verfahrens in vielen Fällen größer ist als die vom äußeren Verfahren unterstützte Klartextgröße. (Von den in Kapitel 4 beschriebenen Verfahren unterstützt keins längere Klartexte als 256 bit. Die Ciphertexte aller LWE-Verfahren haben Größen, die weit über typischen RSA-Moduli liegen.) Der Ciphertext des inneren Verfahrens müsste also ggf. in mehrere Blöcke unterteilt werden und das äußere Verfahren mehrfach angewandt werden; dabei entsteht weiterer Aufwand, weil ein Betriebsmodus verwendet werden muss.

Schlüsselaustausch

Asymmetrische Verschlüsselungsverfahren sind in Bezug auf die Größe des Klartextes wesentlich weniger effizient als symmetrische Verfahren. Sie werden daher in der Praxis selten alleine, sondern meist in Kombination mit einem symmetrischen Verfahren benutzt. Dies wird üblicherweise *hybride Verschlüsselung* genannt [1]. Hierbei transportiert das asymmetrische Verfahren statt der Nachricht nur einen kurzen Schlüssel

[1] siehe den Hinweis zur Begriffsverwendung am Beginn dieses Kapitels

(in diesem Kontext *gemeinsames Geheimnis* genannt). Die Nachricht selbst wird symmetrisch verschlüsselt, als Schlüssel dient das Geheimnis.

Diesen Fakt kann man bei der Konstruktion ausnutzen. Das hybride (i. s. V. Post-Quanten/klassisch) Verfahren muss dann nicht beliebige Nachrichten, sondern nur das (kurze) Geheimnis transportieren.

In dem hier vorliegenden Fall gibt es zwei asymmetrische Verfahren. Es stellt sich daher bei Wahl dieser Option noch eine weitere Frage, nämlich, wie aus den beiden Nachrichtentexten das gemeinsame Geheimnis gebildet wird. Auch hier gibt es verschiedene Varianten. Die Nachrichtentexte der asymmetrischen Verfahren werden mit μ bezeichnet, sie werden gleichverteilt zufällig gewählt.

- Die beiden asymmetrischen Nachrichten könnten so gewählt werden, dass sie beide die Länge des Geheimnisses haben. Dieses würde durch (bitweises) exklusives Oder der beiden Nachrichten berechnet ($k_{sym} = \mu_{pq} \oplus \mu_{kl}$).
- Die beiden asymmetrischen Schlüssel könnten jeweils halb so lang wie der symmetrische Schlüssel gewählt werden und dieser durch Konkatenation abgeleitet werden ($k_{sym} = \mu_{pq} \parallel \mu_{kl}$).
- Der symmetrische Schlüssel könnte durch eine Schlüsselableitungsfunktion (engl. *Key Derivation Function*, KDF) aus den beiden asymmetrischen Nachrichten abgeleitet werden ($k_{sym} = \mathrm{KDF}(\mu_{pq}, \mu_{kl})$).

In einer vereinfachten Betrachtung, d. h. unter der Annahme, dass im symmetrischen Verfahren keine Schwächen gefunden werden, die sich durch Kenntnis einzelner Schlüsselbits ausnutzen lassen und dass in den asymmetrischen Chiffren keine Schwächen gefunden werden, mit denen bestimmte Klartextbits entschlüsselt werden können, sind alle drei Varianten sicher (für einen Beweis für die erste Variante siehe [GHP18, Lemma 1]). Um das System robuster zu machen und solcherlei Angriffe zu erschweren, ist es sinnvoll, die dritte Option zu wählen – Schlüsselableitungsfunktionen dienen genau dem Zweck, aus eventuell nicht gut zufallsverteilten Daten einen guten Schlüssel abzuleiten.

Es sei der Vollständigkeit halber bereits hier angemerkt, dass die genannten Kombinationsmöglichkeiten nur die IND-CPA-Sicherheit erhalten, für CCA-Sicherheit sind andere Verfahren nötig. Wir werden später darauf zurückkommen.

Nachträglicher zweiter Schlüsselaustausch

Für interaktive Verschlüsselung gibt es noch eine weitere Option, die zwar weniger Sicherheit bietet, aber für die Übergangszeit bis zur Entwicklung von großen Quantencomputern völlig hinreichend ist: Zunächst wird auf klassischem Weg ein verschlüsselter und authentifizierter Kanal erstellt. Anschließend kann in diesem Kanal ein zweiter Schlüsselaustausch mit einem Post-Quanten-Verfahren durchgeführt werden und der Kanal fortan mit diesem Schlüssel verschlüsselt werden.

Ein Vorteil dieser Option ist, dass letzterer Schlüsselaustausch nur gegen passive Angreifer sicher sein muss, denn die Sicherheit des äußeren Kanals kann erst gebrochen werden, wenn es hinreichend große Quantencomputer gibt. Bewahrt ein Angreifer einen Mitschnitt der Konversation auf und entschlüsselt diese später mit einem Quantencomputer, ist es zu spät für einen aktiven Angriff auf den Post-Quanten-Schlüsselaustausch.

Da dieses Verfahren so nicht für offline-Verschlüsselung eingesetzt werden kann, werde ich es hier nicht weiter vertiefen.

5.1.4 Symmetrisches Verfahren

Die Wahl des symmetrischen Verfahrens ist nicht sonderlich relevant, da dessen Eigenschaften für die Konstruktion nicht wichtig sind. Um die Konstruktion möglichst genau spezifizieren zu können, ist es trotzdem sinnvoll, ein Verfahren festzulegen. Für symmetrische Verschlüsselung ist AES der etablierte Standard[BSI2102], und es sind keine gravierenden Schwächen oder Gefahren bekannt, ich wähle daher als symmetrische Komponente AES aus.

Da mit dem Grover-Algorithmus auf Quantencomputern die Schwierigkeit, symmetrische Verfahren zu brechen, (bezogen auf das Sicherheitsniveau) halbiert werden kann, sollte AES mit einer Schlüssellänge von 256 bit verwendet werden, um ein Sicherheitsniveau von 128 bit zu erreichen [NW17]. Ein Sicherheitsniveau von 256 bit ist aktuell nicht möglich, da AES für 512 bit bisher nicht spezifiziert ist.

Um längere Nachrichten verschlüsseln zu können, ist die Nutzung von AES in einem Betriebsmodus nötig; um für die Gesamtkonstruktion IND-CCA-Sicherheit zu erreichen, muss dieser ebenfalls IND-CCA-sicher sein. Eine vertiefte Analyse in Frage kommender Verfahren ist nicht Teil dieser Arbeit. Der Galois/Counter Mode (GCM, [NIST38d]) erfüllt die Vorraussetzungen und ist weit verbreitet und erprobt [BSI2102]; er wird daher im Folgenden verwendet.

5.1.5 Weitere Optimierungen

Eine weitere Möglichkeit, das hybride Verfahren effizienter zu gestalten, bietet sich bei den geheimen Schlüsseln: Wenn die geheimen Schlüssel beider Verfahren jeweils von einem gemeinsamen Seed abgeleitet werden, kann statt der beiden Schlüssel nur der Seed gespeichert werden statt beider Schlüssel.

Insbesondere für die gitterbasierten Verfahren, deren Schlüsselerzeugung sehr effizient ist, sind hier Einsparungen gut möglich. Bei RSA ist hingegen die Schlüsselerzeugung durch die nötigen Primzahltests sehr aufwändig. Daher würde die beschriebene Speicherplatzersparnis mit einem sehr viel größeren Rechenaufwand erkauft.

Eine solche Optimierung ist daher (zumindest beim Einsatz von RSA als klassischem Verfahren) nur in wenigen Spezialfällen, wo kleine private Schlüssel extrem wichtig sind, sinnvoll. Dies sollte dann im Einzelfall auf die konkreten Anforderungen hin optimiert werden und wird daher in dieser allgemein gehaltenen Arbeit nicht weiterverfolgt.

Eine ähnliche Optimierung bei den öffentlichen Schlüsseln ist generell nicht möglich, da diese auf eine verfahrensspezifische Art aus dem jeweiligen geheimen Schlüssel abgeleitet werden.

5.2 Transformationen

Viele Verschlüsselungsalgorithmen bieten in ihrer Grundform nur recht schwache Sicherheitsgarantien. So erfüllt RSA in der Lehrbuch-Variante beispielsweise offensichtlich nicht die Ununterscheidbarkeit, da es deterministisch ist; auch alle in dieser Arbeit untersuchten Verfahren erreichen in ihrer Grundform höchstens IND-CPA. Es ist jedoch mittels sogenannter *Transformationen* möglich, solche Verfahren umzuwandeln in stärkere Verfahren, die starke Sicherheitsdefinitionen wie IND-CCA erfüllen.

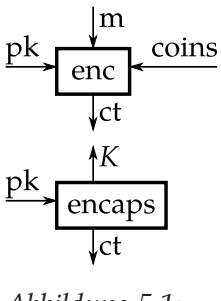

Abbildung 5.1: „Syntax" der im Folgenden verwendeten schematischen Darstellungen. Doppelte Ein-/Ausgänge stehen für Konkatenation.

Diese Zweiteilung – schwache Kernfunktion plus Transformation zu starkem Verfahren – ist aus zweierlei Gründen sinnvoll: Der Hauptvorteil ist, dass man so die Kernfunktion allein darauf optimieren kann, verlässlich und mit möglichst guter Effizienz die Einwegeigenschaft zu erfüllen. Außerdem ist es auf diese Weise möglich, mit der selben Kernfunktion sowohl sichere (im hier betrachteten Sinne) als auch homomorphe Verschlüsselung [KL07, § 11.1.3] zu implementieren. (Ein Verfahren mit homomorphen Eigenschaften kann prinzipiell nicht ununterscheidbar sein, denn die Homomorphie verletzt offensichtlich die Unverfälschbarkeit.)

Ich werde im Folgenden verschiedene dieser Transformationen vorstellen. Dabei werden die Verschlüsselungsfunktionen grafisch dargestellt, entsprechend dem Schema in Abbildung 5.1. Für die Definitionen der Primitiven (enc, encaps, etc.) siehe Abschnitt 2.3.2.

Das Grundprinzip ist bei allen Verfahren gleich, weshalb ich es hier bereits in allgemeiner Form vorstelle. Es gilt, zwei Hauptangriffsvektoren zu verhindern:

- Wenn das verwendete Grundverfahren nur Einweg-Garantien bietet, ist nur garantiert, dass ein Angreifer nicht die *komplette* Nachricht herausfinden kann – dass einzelne Bits leaken, ist aber nicht ausgeschlossen. Davor schützen sogenannte

Alles-oder-Nichts-Mechanismen: Das Ergebnis der Entschlüsselung muss zunächst gehasht werden, um an den Klartext zu gelangen. Eine Hashfunktion ist so entworfen, dass eine Veränderung nur eines Eingabebits bewirkt, dass sich jedes Ausgabebit mit 50 % Wahrscheinlichkeit ändert (Lawineneffekt, engl. *avalanche effect* [KL07, S. 166]), so dass solche Angriffe unmöglich sind.

- Um gegen Chosen-Ciphertext-Angriffe zu schützen, müssen veränderte Ciphertexte erkannt werden können[2], d. h. es muss schwer sein, einen gültigen Ciphertext zu erzeugen, ohne den zugehörigen Klartext zu kennen (abgesehen von Modifikationen, die einen unverhersehbaren, zufälligen Klartext ergeben, diese sind unkritisch).

5.2.1 Optimal Asymmetric Encryption Padding (OAEP)

RSA ist in seiner „reinen" Form (Lehrbuch-RSA) unsicher (es bietet nur OW-Sicherheit), und zwar aus vielerlei Gründen, zwei seien hier genannt:

- Es ist stark (multiplikativ) homomorph: Zur Erinnerung, die Verschlüsselung einer Nachricht m ist $\mathsf{enc}(m) = m^e \pmod N$. Es gilt also $\mathsf{enc}(m \cdot \mu) = m^e \cdot \mu^e \pmod N = \mathsf{enc}(m) \cdot \mathsf{enc}(\mu)$. Das ermöglicht es zum Beispiel, über Chosen-Ciphertext-Angriffe beliebige Nachrichten zu entschlüsseln [Buc16, § 8.3.9].
- Es ist deterministisch und daher nicht ununterscheidbar.

Mihir Bellare und Phillip Rogaway erfanden 1994 das OAEP-Verfahren [BR95], das RSA IND-CCA-sicher macht und somit vor solcherlei Angriffen schützt. Es ist prinzipiell auch mit anderen OW-sicheren Verschlüsselungsverfahren nutzbar, ist aber stark auf RSA zugeschnitten, z. B. wird angenommen, dass das Verfahren eine Permutation ist (also CSP = MSP), und dass es deterministisch ist; im Allgemeinen erreicht OAEP auch nur IND-CPA-Sicherheit, nur für den Spezialfall RSA wird IND-CCA erreicht [RFC8017; Sho01b]. Der Beweis nutzt das Random Oracle Model.

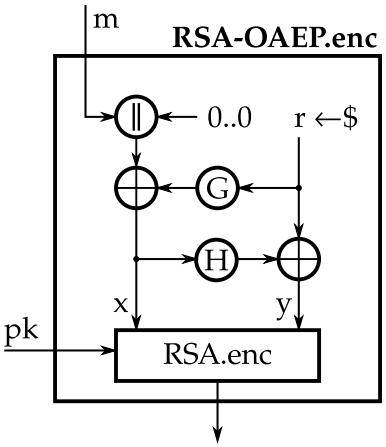

Abbildung 5.2: OAEP-Verschlüsselung, schematisch

Die folgende Beschreibung lehnt sich an [BR95] an. Die Variante in [RFC8017] ist leicht modifiziert und erlaubt zusätzlich die Parametrisierung mittels eines Labels.

[2] Bei symmetrischen Verfahren werden hierzu in der Regel MACs benutzt, mangels gemeinsamem Geheimnis fällt diese Option hier aus.

Parameter des Verfahrens sind k, die Anzahl der Bits, die mit der zugehörigen RSA-Funktion verschlüsselt werden können, sowie zwei Hashfunktionen G und H, letztere mit der Ausgabelänge l. Diese muss so groß sein, dass 2^{-l} vernachlässigbar klein ist. G ist i. d. R. eine XOF, es wird eine Ausgabelänge von $k - 2 \cdot h$ Bit benötigt. In der Praxis wird für beide die selbe Hashfunktion verwendet und für G geeignet mehrfach aufgerufen.

An die Nachricht werden zunächst l Nullbits angehängt. Nach der Entschlüsselung kann so überprüft werden, ob der Ciphertext manipuliert wurde. Die Wahrscheinlichkeit, dass nach einer solchen Manipulation wieder ein Klartext, der mit l Nullbits endet, ist vernachlässigbar gering. Die Verkettung wird im Folgenden mit $\overline{m}$ bezeichnet.

Anschließend wird ein zufälliger Wert $r \xleftarrow{\$} \mathbb{B}^l$ gewählt. Die beiden Teile des letztendlich zu verschlüsselnden Klartexts sind:

$$x := \overline{m} \oplus G(r)$$
$$y := r \oplus H(x)$$

Dieses „überkreuzte“ Hashen schützt als Alles-oder-Nichts-Mechanismus den Klartext, wie oben beschrieben. Außerdem verhindert es, dass bekannte, zu auf 0^l endenden Klartexten entschlüsselnde Ciphertexte eingeschleust werden können, um den weiter oben beschriebenen Schutzmechanismus zu umgehen.

Die Entschlüsselung ist trivial: Erst wird $r = H(x) \oplus y$ berechnet, daraus kann $\overline{m} = x \oplus G(r)$ berechnet werden und die angehängten Nullen verifiziert werden.

5.2.2 Fujisaki-Okamoto-Transformation

Eiichiro Fujisaki und Tatsuaki Okamoto entwickelten diese Transformation 1995 [FO13]. Die FO-Transformation ermöglicht es, ein symmetrisches und ein asymmetrisches Verschlüsselungsverfahren, die beide nur relativ schwache Sicherheitsdefinitionen erfüllen, zu einem sehr starken (IND-CCA-sicheren) Hybrid-Verfahren (i. s. V. sym./asym.) zu kombinieren. Der Beweis dazu ist im Random Oracle Model geführt. Konkret werden folgende Anforderungen gestellt:

- Das asymmetrisches Verfahren muss lediglich die Einwegeigenschaft (OW) erfüllen, also garantieren, dass nicht die vollständige Nachricht entschlüsselt werden kann. Außerdem darf das Verfahren nicht deterministisch sein, genauer: zu jedem Klartext muss es $2^{\omega(\log k)}$ mögliche Ciphertexte geben (üblicherweise „γ-spread“ genannt, Details siehe [FO13, § 5.2]). k ist der Sicherheitsparameter des Verfahrens.

  Die Verschlüsselungsfunktion des Verfahrens muss jedoch unterstützen, sämtlichen benötigten Zufall als dritten Parameter (Coins) übergeben zu bekommen. Die

Funktion selbst ist somit (bei gleichbleibenden coins) deterministisch – es ist Aufgabe des Aufrufenden, sicherzustellen, dass geeignete Zufallsbits verwendet werden, damit die Verschlüsselung nach außen hin nicht-deterministisch ist.

- Das symmetrische Verfahren muss ununterscheidbar sein, wenn jeder Schlüssel nur für eine Nachricht verwendet wird. Das ist eine schwächere Form von IND-CPA (§ 2.3.1), die Definition ist bis auf das fehlende Verschlüsselungsorakel identisch (siehe [CS03, § 7.2.1] für eine vollständige Definition).

Eine – wie wir noch sehen werden – für Post-Quanten-Kryptografie recht starke Einschränkung ist, dass beide Verfahren fehlerfrei sein müssen.

Funktionsweise Die Nachricht wird mit dem symmetrischen Verfahren verschlüsselt, dieser Ciphertext ist der zweite Teil des resultierenden Ciphertexts. Für den Schlüssel wird ein Zufallswert σ erzeugt, der später mit dem asymmetrischen Verfahren verschlüsselt wird. Da letzteres nur OW-Sicherheit garantiert, wird nicht σ selbst, sondern ein Hash $G(\sigma)$ als Schlüssel für das symmetrische Verfahren verwendet (Alles-oder-Nichts-Mechanismus). Hiermit ist der Schutz vor einer teilweisen Entschlüsselung bereits erreicht (denn das symmetrische Verfahren bietet Ununterscheidbarkeit).

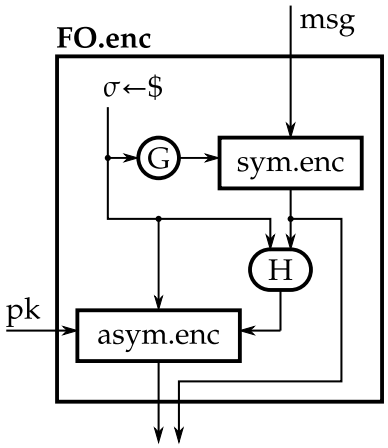

Abbildung 5.3: Fujisaki-Okamoto-Verschlüsselung, schematisch

Zum Schutz vor Manipulationen wird die Determiniertheit der (asymmetrischen) Verschlüsselung ausgenutzt, als coins wird ein Hash von σ und dem symmetrischen Ciphertext übergeben. Bei der Entschlüsselung wird die *Ver*schlüsselungsfunktion erneut mit den selben Parametern erneut aufgerufen (*re-encryption*). Wenn dabei ein anderer Ciphertext entsteht, wird ein Fehler zurückgegeben. Jede Manipulation am symmetrischen Ciphertext verändert die coins und kann daher erkannt werden. Ein Angreifer kann auch keinen gültigen asymmetrischen Ciphertext erzeugen, denn er kennt den geheimen Wert σ nicht. Würde er σ selbst wählen, entstünde bei der symmetrischen Entschlüsselung ein anderer, unvorhersehbarer Klartext. Es ist wichtig, dass ein Angreifer nicht zwischen verschiedenen Fehlern unterscheiden kann.

Im Folgenden wird die Funktionsweise der Verschlüsselung als Algorithmus dargestellt.

1: $\sigma \xleftarrow{\$} \mathsf{MSP}_{\text{asym}}$
2: $\text{ct}_{\text{sym}} \leftarrow \mathsf{sym.enc}(\text{msg}, G(\sigma))$
3: $\text{coins} \leftarrow H(\sigma \parallel \text{ct}_{\text{sym}})$
4: $\text{ct}_{\text{asym}} \leftarrow \mathsf{asym.enc}(\sigma, \text{pk}, \text{coins})$
5: **return** $\text{ct}_{\text{asym}}, \text{ct}_{\text{sym}}$

5.2.3 HHK-Transformationen

Die FO-Tranformation hat einige Schwächen. Zum einen müssen die verwendeten Verfahren fehlerfrei sein (was LWE-Verfahren i. d. R. nicht sind); andererseits werden nur sehr geringe Anforderungen an die zugrundeliegenden Verfahren gestellt, was die Transformation aufwändiger macht als oft nötig ist.

In einem recht jungen Paper [HHK17] entwarfen Dennis Hofheinz, Kathrin Hövelmanns und Eike Kiltz daher eine Sammlung von Transformationen, die zum einen auch Verfahren mit einer gewissen Fehlerwahrscheinlichkeit unterstützen, und zum anderen modular miteinander kombiniert werden können (d. h. erfüllt das grundlegende Verfahren eine recht starke Sicherheitsdefinition, kann ein Teil der Transformationen weggelassen werden, so dass die Gesamttransformation einfacher wird).

Ein weiterer praktischer Unterschied zur FO-Transformation ist, dass die Transformationen nur das asymmetrische Verfahren in sich zu einem IND-CCA-sicheren umwandeln; dieses kann danach mit einem symmetrischen Verfahren kombiniert werden, das sei immer möglich [HHK17, S. 3]. Außerdem sind die Sicherheitsreduktionen der HHK-Transformationen mit einer Ausnahme (s. u.) enger (engl. *tighter*) als in den Beweisen von [FO13] (d. h. in der FO-Transformation ist die Sicherheit des transformierten Verfahrens um einen Faktor niedriger als die des Ursprungsverfahrens, bei HHK fehlt dieser Faktor). Auch die Beweise der HHK-Transformationen sind jedoch im Random Oracle Model geführt.

Für diese Arbeit sind die beiden Transformationen T und $U_m^{\perp}$ relevant, die verschiedene Zwecke erfüllen und nacheinander angewendet werden; das resultierende Verfahren ist CCA2-sicher.

Die Transformation *T*

Die Transformation T wandelt ein OW-CPA- oder IND-CPA-Verfahren in ein OW-PCVA-sicheres Verfahren um, wenn ersteres γ-spread (s. o.) ist [3]. PCVA ist eine weniger verbreitete Sicherheitsdefinition. Sie dient nur als Zwischenschritt für die nächste Transformation, ich stelle sie deshalb nur knapp vor, für die Details siehe [HHK17, Def. 2.1]. OW-PCVA entspricht OW-CPA, ist jedoch auch gegen einen Angreifer resistent, der zwei Orakel zur Verfügung hat: ein *Plaintext Checking Oracle*, mit dem er herausfinden kann, ob ein Ciphertext zu einer Nachricht entschlüsselt, sowie ein *Ciphertext Validity Oracle*, das ihm sagt, ob ein Ciphertext gültig ist.

Die Autor_innen liefern zwei getrennte Sicherheitsbeweise für die beiden Fälle. Ist das Ausgangsverfahren nur OW-CPA, ist die Reduktion nicht *tight*: Die Erfolgswahrscheinlichkeit des Angreifers auf OW-PCVA ist um die Anzahl der Auswertungen

[3] Die nach der Transformation erfüllte Sicherheitsdefinition variiert in Abhängigkeit der Sicherheit des Ursprungsverfahrens. Die hier genannten Sicherheitsdefinitionen sind tatsächlich nur eine von mehreren Möglichkeiten.

der Hashfunktion höher als die des Angreifers auf OW-CPA, was bei der Auswahl der Sicherheitsparameter berücksichtigt werden sollte.

Die Transformation ist sehr simpel: Bei der Verschlüsselung wird ein Hash der Nachricht als coins benutzt – als Formel: $T.\mathsf{enc}(\mathrm{msg}, \mathrm{pk}) = \mathsf{enc}(\mathrm{msg}, \mathrm{pk}, H(\mathrm{msg}))$. Die Verschlüsselung wird also deterministisch (und ist damit nicht mehr ununterscheidbar, auch wenn es das zu Grunde liegende Verfahren ist). Das ermöglicht, nach der Entschlüsselung den Klartext nochmals zu verschlüsseln und die beiden Ciphertexte zu vergleichen – sind sie verschieden, wird ein Fehler ($\perp$) zurückgegeben. Die Schlüsselgenerierung bleibt unverändert.

Die Transformation $U_m^{\perp}$

In einem zweiten Schritt wird das nach der Anwendung von T entstehende deterministische Verfahren in ein CCA2-sicheres KEM umgewandelt. Hier bietet [HHK17] eine Familie von vier Transformationen an: $U^{\perp}$, $U^{\not\perp}$, $U_m^{\perp}$ und $U_m^{\not\perp}$, die sich in zwei Details unterscheiden.

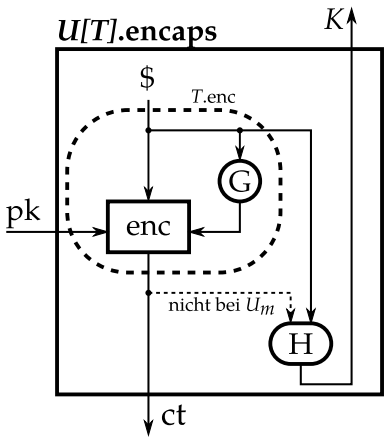

Abbildung 5.4: Die beiden Transformationen U und T (gemeinsam angewendet), schematisch. Der gestrichelt umrandete Bereich ist der Transformation T zuzuordnen, der äußere U.

Bei den beiden $\not\perp$-Transformationen gibt die Entschlüsselungsfunktion bei einem Fehler statt dem Fehlersymbol $\perp$ einen (deterministisch) zufälligen Wert zurück („implicit rejection"). Das bietet den Vorteil, dass das Ursprungsverfahren (vor Anwendung von T) nicht γ-spread sein muss. Der Nachteil ist aber, dass ein Seed für die Generierung dieses Fehlerwerts mit im geheimen Schlüssel gespeichert werden muss.

Die beiden Varianten ohne Index m sind für nichtdeterministische Ursprungsverfahren gedacht, daher ist hier der ausgegebene Sitzungsschlüssel ein Hash von Nachricht und Ciphertext. Im anderen Fall genügt ein Hash der Nachricht.

Wir können hier die effizienteste und einfachste Transformation, $U_m^{\perp}$, benutzen. Die verwendeten Verfahren sind, wenn coins als Parameter übergeben werden, alle deterministisch. Bei den LWE-Verfahren[4] werden bei der Verschlüsselung verschiedene Werte pseudozufällig erzeugt, es ist also stark anzunehmen, dass sie γ-spread selbst für große Werte von γ sind (dies ist jedoch für konkrete Verfahren noch detailliert zu untersuchen). RSA kann durch einfaches Hinzufügen von (deterministisch aus der Nachricht erzeugten) Zufallsbits γ-spread gemacht werden für fast beliebige γ. Dafür

[4] Classic McEliece kommt für das Szenario, in dem diese Transformation verwendet wird, nicht in Frage, wie wir später sehen werden.

muss die Schlüsselgröße nicht erhöht werden, da nur 256 bit Nutzdaten benötigt werden und für einen längerfristigen Einsatz ohnehin RSA-Moduli weit über 1024 bit empfohlen werden.

Bei der Verkapselung wird ein Zufallswert m erzeugt. Dessen Verschlüsselung (n. b. : als Verschlüsselungsverfahren wird das der Transformation T benutzt) ist der Ciphertext, der Sitzungsschlüssel ist ein Hash $H(m)$. Die Entkapselung ist trivial – eine Gültigkeitsprüfung ist nicht nötig, das übernimmt die Entschlüsselungsfunktion von T. Gibt diese $\perp$ zurück, tut das auch die Entkapselung.

Anmerkung zur Robustheit Die Transformationen sind tatsächlich eher simpel – der Hauptbeitrag des Papers [HHK17] sind nicht die Transformationen selbst, sondern die Beweise ihrer Sicherheit. Die hier angegebene Form der Transformationen sind die Minimalvarianten, für die die Sicherheit bewiesen werden kann – das heißt nicht, dass die Transformation genau so verwendet werden sollte. Beispielsweise kann auch bei einem deterministischen Verfahren der Ciphertext mitgehasht werden (anders gesagt: U statt U_m genutzt werden), um die Robustheit zu erhöhen. Siehe auch die Anmerkungen in Abschnitt 5.4, Seite 62.

5.3 Weitere Variantendiskussion

Mein ursprünglicher Plan war, mittels der FO-Transformation die PKEs der Post-Quanten-Verfahren (statt den bereits transformierten KEMs) und RSA in der Lehrbuch-Variante zu einem IND-CCA-sicheren Verfahren zu kombinieren.

Das Verfahren hätte etwa folgende Gestalt gehabt:

- Von den PQ-Verfahren würden die PKE-Varianten genutzt. Diese sind alle (mindestens) OW-CPA.
- RSA würde in der Lehrbuch-Variante genutzt, jedoch derart, dass der Klartext um zufällige Bits ergänzt wird, um es γ-spread zu machen, wie von der FO-Transformation verlangt.
- Die beiden Verfahren würden wie oben (§ 5.1.3, dort insbesondere Abschnitt „Schlüsselaustausch“ auf Seite 50) beschrieben kombiniert.
- Dieses hybride (i. s. V. PQ/klassisch) Verfahren wäre dann mittels der FO-Transformation gemeinsam mit AES zu einem hybriden (i. s. V. asymmetrisch/symmetrisch) Datenverschlüsselungsverfahren kombiniert worden. Dieses wäre (nach [FO13]) IND-CCA-sicher.

Dies hätte folgende Vorteile geboten:

- Es wäre nur eine Transformation notwendig gewesen, statt jedes Verfahren getrennt zu transformieren. Der Ciphertext wäre damit etwas kleiner, und die Ver- und Entschlüsselung etwas schneller gewesen.

- Die Kombination der beiden asymmetrischen Verfahren zu einem hätte nur OW-CPA-sicher sein müssen, da die Transformation zu IND-CCA erst danach erfolgt wäre.
- Die Konstruktion wäre sehr einfach gewesen und ihre Sicherheit hätte zu großen Teilen direkt auf der der FO-Transformation beruht, die schon relativ alt und etabliert ist.

Es hat sich bei der Entwicklung jedoch herausgestellt, dass die FO-Transformation gewisse Anforderungen stellt, die von den verwendeten Verfahren nicht erfüllt werden können.

Zum einen unterstützt die FO-Transformation nur Verfahren mit perfekter Korrektheit. Dies liegt wohl nicht daran, dass das prinzipiell unmöglich wäre (wie die ähnlichen HHK-Transformationen zeigen), sondern ist vermutlich der Tatsache geschuldet, dass zur Zeit ihrer Entwicklung nur fehlerfreie Verfahren verbreitet waren und die Autoren deshalb keine Notwendigkeit dafür sahen. Eine Anpassung der FO-Transformation (oder evtl. sogar nur eine Anpassung des Sicherheitsbeweises) an fehlerbehaftete Verfahren wäre also vermutlich möglich, sprengt aber den Rahmen dieser Arbeit. Das oben beschriebene Vorhaben ist also mit den LWE-basierten Verfahren nicht umsetzbar.

Zum anderen verlangt die FO-Transformation (wie auch die HHK-Transformationen) als asymmetrisches Verfahren ein Schlüsseltransportverfahren (KTM; § 2.3.2, S. 10), ein Schlüsselkapselungsverfahren (KEM) wird nicht unterstützt. Classic McEliece implementiert aber nur ein KEM (NIST verlangt auch nur, eines der beiden zu implementieren [NIS16]). Es gibt zwar eine intern verwendete KEM-ähnliche Funktionalität, die aber aus Effizienzgründen nicht das Verschlüsseln beliebiger Bitstrings erlaubt, sondern nur eines Bitstrings mit einem bestimmten Gewicht. Die (deterministische) Umwandlung eines beliebigen Bitstrings in einen (längeren) Bitstring mit festgelegtem Gewicht ist zwar möglich, wenn auch recht ineffizient [OS09, S. 98]. Allerdings könnte eine derartige Anpassung des Verfahrens die Sicherheitseigenschaften des Systems verändern und müsste daher tiefgreifend analysiert werden; ist also im Rahmen dieser Arbeit nicht möglich. Damit scheidet das oben beschriebene Vorhaben auch für Classic McEliece aus.

Es bleiben damit zwei weitere Optionen:

- Umsetzung des ursprünglichen Plans, jedoch mit Nutzung der HHK-Transformationen anstatt der FO-Transformation. Dies schließt jedoch Classic McEliece aus. Auch wird die Konstruktion leicht komplexer. Die restlichen Vorteile der ursprünglich geplanten Lösung bleiben aber bestehen.
- Nutzung der transformierten, IND-CCA-sicheren KEM-Form der Einreichungen bzw. RSA-KEM und Nutzung einer IND-CCA-erhaltenden Kombination.

  Diese Konstruktion ist in der Umsetzung komplexer als die oben genannte und hat auch etwas größere Ciphertexte und benötigt etwas mehr Rechenleistung. Sie hat

aber den Vorteil, dass ausschließlich erprobte oder bewiesen sichere Komponenten verwendet werden. Betrachtet man die einzelnen Komponenten als abgeschlossene Einheiten, ist die Variante aber weniger komplex, sie ist also leichter zu verstehen und zu analysieren. Hinzu kommt, dass die KEM-Transformationen der PQ-Verfahren selbst verwendet werden, die z. T. im Vergleich zu den Transformationen aus [FO13] oder [HHK17] weiter verbessert sind und etwas besser an die Eigenschaften des jeweiligen Verfahrens angepasst sind.

Da beide Optionen ihre Vor- und Nachteile haben, werde ich beide Verfahren umsetzen und in den beiden folgenden Abschnitten vorstellen.

5.4 Variante mit einmaliger Transformation

Das Verfahren besteht aus folgenden Teilen:

- PKE-Variante eines Post-Quanten-Verfahrens.

  Hier kommen nur die LWE-Verfahren in Betracht, da Classic McEliece nur einen KEM bietet. Alle drei LWE-Verfahren bieten eine deterministische PKE-Funktionalität.[5]

  Frodo unterstützt als einziges Verfahren nur Nachrichtenlängen von 128 bzw. 192 Bit. Soll Frodo zum Einsatz kommen, müsste die folgende Beschreibung also leicht angepasst werden. Alternativ könnte, falls in Zukunft ein Parametersatz für Kategorie 5 spezifiziert wird, der eine Nachrichtenlänge von 256 Bit unterstützt, dieser genutzt werden.
- RSA in der Lehrbuch-Variante mit γ-spread-Anpassung.

  Da wir nach der Kombination zu IND-CCA transformieren, genügt es, wenn RSA one-way ist, der Einsatz eines Paddingverfahrens ist also nicht nötig. Um die Transformation $U_m^{\perp}$ nutzen zu können, muss RSA γ-spread sein. Da nur 256 bit verschlüsselt werden müssen, kann das leicht durch voranstellen von Zufallsbits erreicht werden. Diese Zufallsbits werden der Verschlüsselungsfunktion als weiterer Parameter `coins` übergeben, so dass sie deterministisch bleibt.

  Da die Transformation T für die coins einen Hash der Nachricht verwendet, entspricht diese Konstruktion letztendlich der Verschlüsselung eines Hashes der Nachricht und der Nachricht selbst (siehe unten). Bei der Entschlüsselung kann daher statt der eigentlich nötigen Neuverschlüsselung einfach dieser Präfix verglichen werden.
- Kombination mittels KDF

[5] Bei Frodo ist dies nicht explizit vorgesehen, aber trivial implementierbar, indem der innerhalb der Verschlüsselungsfunktion gezogene Zufallswert als weiterer Parameter definiert wird [Fro, Alg. 10, Z. 2].

- IND-CCA mittels HHK-Transformationen T und $U_m^{\perp}$.

  Für die Gründe der Auswahl dieser Transformationen siehe Abschnitt 5.2.3. Ihre Anforderungen (γ-spread, deterministisch) werden von den verwendeten Verfahren erfüllt, s. o.
- Datentransport mittels AES-GCM mit 256-bit-Schlüsseln. Die Nutzung eines zufälligen Initialisierungsvektors (IV) ist nicht nötig, da bei jeder Verschlüsselung ein neuer Schlüssel verwendet wird.
- Eine Hashfunktion $H : \mathbb{B}^* \to \mathbb{B}^{256}$ und eine Schlüsselableitungsfunktion $KDF : \mathbb{B}^* \to \mathbb{B}^{256}$. Letztere sollte möglichst auf H basieren, um die Anzahl verwendeter Primitiven gering zu halten.

Die Schlüsselgeneration ist die triviale Kombination der Schlüsselgeneration von RSA bzw. dem PQ-Verfahren. Die beiden öffentlichen bzw. die beiden privaten Schlüssel können einfach konkateniert werden, solange sie eine feste Länge haben (dies ist bei allen verwendeten Verfahren der Fall).

Die Verschlüsselung ist in Algorithmus 5.1 beschrieben, die Entschlüsselung in Algorithmus 5.2. Bei der Implementierung der Entschlüsselung ist darauf zu achten, dass ein Angreifer nicht (z. B. durch Laufzeitmessungen) zwischen verschiedenen Fehlerursachen unterscheiden kann.

Robustheit

Es ist festzuhalten, dass dieses Verfahren dasjenige ist, das die Mindestanforderungen der beweisbaren Sicherheit erfüllt. Es ist möglich und zu empfehlen, die Robustheit durch weitere Maßnahmen zu erhöhen, zum Beispiel (die in Kapitel 4 vorgestellten Einreichungen setzen in ihren Transformationen einige dieser Ideen um):

- Um Angriffe auf den System-Zufallszahlengenerator sicher auszuschließen, können die Zufallswerte (μ_{kl}, μ_{pq}) vor der Verwendung gehasht werden. (Umgesetzt in Kyber)
- Auch wenn $U_m[T]$ es für deterministische Verfahren nicht verlangt, können zur Erzeugung des Geheimnisses (Alg. 5.1, Zeile 5) dennoch auch die Ciphertexte mit gehasht werden. (Alle)
- In der Transformation T entsteht bei identischen Zufallswerten, aber verschiedenen öffentlichen Schlüsseln dennoch der selbe Ciphertext. Dies kann verhindert werden, indem bei der Berechnung der Coins der öffentliche Schlüssel mitgehasht wird (Zeilen 3 und 4 von Algorithmus 5.1; Kyber)
- AES-GCM bietet tatsächlich mehr als nur IND-CCA-Sicherheit, nämlich AEAD (*authenticated encryption with associated data*). Das ließe sich dazu nutzen, die Robustheit des Verfahrens weiter zu erhöhen, indem ct_{kl} und ct_{pq} mit authentifiziert werden.

Algorithmus 5.1 Verschlüsselung

Eingabe: $\mathrm{pk}_{\mathrm{hy}} = (\mathrm{pk}_{\mathrm{kl}}, \mathrm{pk}_{\mathrm{pq}})$ öffentlicher Schlüssel
Eingabe: $\mathrm{msg} \in \mathsf{MSP}_{\mathrm{sym}}$ Nachricht
Ausgabe: $\mathrm{ct} \in \mathsf{CSP}_{\mathrm{hy}} = \mathsf{CSP}_{\mathrm{kl}} \times \mathsf{CSP}_{\mathrm{pq}} \times \mathsf{CSP}_{\mathrm{sym}}$ Ciphertext

1: $\mu_{\mathrm{kl}} \xleftarrow{\$} \mathbb{B}^{256}$
2: $\mu_{\mathrm{pq}} \xleftarrow{\$} \mathbb{B}^{256}$
3: $\mathrm{ct}_{\mathrm{kl}} \leftarrow \texttt{RSA.enc}(\mathrm{pk}_{\mathrm{kl}}, H(\mu_{\mathrm{kl}}) \parallel \mu_{\mathrm{kl}})$
4: $\mathrm{ct}_{\mathrm{pq}} \leftarrow \texttt{pq.enc}(\mathrm{pk}_{\mathrm{pq}}, \mu_{\mathrm{pq}}, H(\mu_{\mathrm{pq}}))$
5: $\mathrm{k}_{\mathrm{sym}} \leftarrow \texttt{KDF}(\mu_{\mathrm{kl}} \parallel \mu_{\mathrm{pq}})$
6: $\mathrm{ct}_{\mathrm{sym}} \leftarrow \texttt{AES-GCM.enc}(\mathrm{k}_{\mathrm{sym}}, \mathrm{msg}, \mathrm{IV} = \vec{0})$
7: **return** $\mathrm{ct}_{\mathrm{kl}}, \mathrm{ct}_{\mathrm{pq}}, \mathrm{ct}_{\mathrm{sym}}$

Algorithmus 5.2 Entschlüsselung

Eingabe: $\mathrm{sk}_{\mathrm{hy}} = (\mathrm{sk}_{\mathrm{kl}}, \mathrm{sk}_{\mathrm{pq}})$ geheimer Schlüssel
Eingabe: $\mathrm{ct} = (\mathrm{ct}_{\mathrm{kl}}, \mathrm{ct}_{\mathrm{pq}}, \mathrm{ct}_{\mathrm{sym}}) \in \mathsf{CSP}_{\mathrm{hy}}$ Ciphertext
Ausgabe: $\mathrm{msg} \in \mathsf{MSP}_{\mathrm{sym}} \cup \{\perp\}$ Nachricht oder Fehlersymbol

1: `failure` $\leftarrow$ `false`
2: $\mu_{\mathrm{kl}} \leftarrow \texttt{RSA.dec}(\mathrm{sk}_{\mathrm{kl}}, \mathrm{ct}_{\mathrm{kl}})$ RSA entschlüsseln
3: **if** $\mu_{\mathrm{kl}} \mathbin{//} 2^{512} \neq 0$ **or** $(\mu_{\mathrm{kl}} \mathbin{//} 2^{256}) \neq H(\mu_{\mathrm{kl}} \bmod 2^{256})$**:** RSA verifizieren
4: `failure` $\leftarrow$ `true`
5: $\mu_{\mathrm{pq}} \leftarrow \texttt{pq.dec}(\mathrm{sk}_{\mathrm{pq}}, \mathrm{ct}_{\mathrm{pq}})$ PQ entschlüsseln
6: **if** $\mu_{\mathrm{pq}} = \perp$ **or** $\texttt{pq.enc}(\mathrm{pk}_{\mathrm{pq}}, \mu_{\mathrm{pq}}, H(\mu_{\mathrm{pq}})) \neq \mathrm{ct}_{\mathrm{pq}}$**:** PQ verifizieren
7: `failure` $\leftarrow$ `true`
8: **if** `failure`**:**
9: $\mathrm{k}_{\mathrm{sym}} \leftarrow \texttt{KDF}(\mathrm{ct}_{\mathrm{kl}} \bmod 2^{256} \parallel \mathrm{ct}_{\mathrm{pq}} \bmod 2^{256})$ Schutz gegen Timing-Angriffe
10: **else:**
11: $\mathrm{k}_{\mathrm{sym}} \leftarrow \texttt{KDF}(\mu_{\mathrm{kl}} \parallel \mu_{\mathrm{pq}})$
12: $\mathrm{msg} \leftarrow \texttt{AES-GCM.dec}(\mathrm{k}_{\mathrm{sym}}, \mathrm{ct}_{\mathrm{sym}})$ AES entschlüsseln
13: **if** $\mathrm{k}_{\mathrm{sym}} = \perp$**:**
14: `failure` $\leftarrow$ `true`
15: **if** `failure`**:**
16: **return** $\perp$
17: **else:**
18: **return** msg

5.4.1 Sicherheit

Das Verfahren besteht aus mehreren einzelnen Teilen, deren Sicherheit getrennt analyisert werden kann: Den Kern-Verfahren, der Kombination dieser beiden Verfahren, der Transformation und der Kombination mit AES.

Wie in Kapitel 4 beschrieben, sind die Kern-Verfahren (mindestens) OW-CPA. Das Ziel der Kombination der beiden Verfahren ist, dass diese Kombination auch dann noch sicher ist, wenn eines der Verfahren gebrochen ist. Wir müssen daher zeigen, dass das Hybrid-Verfahren auch gegen einen Angreifer, der eines der beiden Verfahren entschlüsseln kann, weiterhin ausreichende Sicherheit bietet. Der Beweis folgt im nächsten Abschnitt, da er etwas länger ist. Wir werden zunächst die Sicherheit der restlichen Konstruktion betrachten.

Wir nehmen dafür an: Die Kombination der beiden Verfahren ist OW-CPA-sicher, selbst wenn eines der Verfahren gebrochen (d. h. nicht OW-CPA) ist. Durch die Kombination bleiben die Eigenschaften der beiden Verfahren erhalten, insbesondere ist auch die Kombination deterministisch und γ-spread. Die in § 5.2.3 genannten Bedingungen für T und $U_m^{\perp}$ sind also erfüllt. Es folgt, dass das transformierte Verfahren ein IND-CCA-sicheres KEM ist.

AES-GCM erfüllt das Kriterium der authentifizierten Verschlüsselung [MV05][6], was stärker ist als IND-CCA [KL15, § 4.5.4]. Die Kombination eines IND-CCA-sicheren KEM und eines IND-CCA-sicheren symmetrischen Verfahrens ist ebenfalls IND-CCA-sicher [CS03, Thm. 5]. Das Gesamtverfahren bietet also IND-CCA-Sicherheit.

Sicherheit der Kombination

Ich werde die Sicherheit einer etwas schwächeren Variante als oben verwendet zeigen. Die Sicherheit der verwendeten (stärkeren) Variante folgt daraus.

Folgende Vereinfachungen nehme ich an:

- Statt einer KDF wird nur ein einfacher Hash der temporären Schlüssel verwendet. Tatsächlich entspricht eine KDF nach dem Extraktions-Expansions-Verfahren (das z. B. vom BSI [BSI2102] empfohlen wird) bei kurzen Ein- und Ausgabelängen und wenn kein Salt oder IV verwendet wird, zweifachem Hashen der Eingabewerte. Diese Vereinfachung verändert die Struktur des Verfahrens nicht grundlegend.
- Die Nachrichtenlängen der symmetrischen Verfahren werden nur mit je 128 bit angenommen, die Ausgabelänge der Hashfunktion ist 256 bit.

Wir müssen zeigen: Auch wenn ein Angreifer einen der asymmetrischen Klartext (komplett) kennt, ist das Sicherheitsniveau der Kombination 128 bit (d. h. es ist ein

[6] Im zitierten Beweis wird angenommen, dass AES eine pseudozufällige Permutation ist. Dies ist nicht bewiesen, wird aber sehr stark angenommen, z. B. [KL15, S. 225].

Zeitaufwand in der Größenordnung von 2^{128} nötig, um den Klartext zu erhalten. Wir nutzen für den Beweis folgendes Lemma:

Lemma 5.1 (Unmöglichkeit der Annäherung an Hashwerte)**.** *Sei $H : \mathbb{B}^* \to \mathbb{B}^{m+n}$ eine Hashfunktion. H sei 2^{m+n}-urbildresistent, d. h. jeder Angreifer auf H benötigt die Zeit 2^{m+n}, um zu einem gegebenen Bild y ein Urbild x zu finden, so dass $H(x) = y$.*

Angenommen, eine Angreiferin kennt die ersten m Bit eines Eingabewertes $x \in \mathbb{B}^{m+n}$ und sie habe ein Orakel zur Verfügung, mit dem sie überprüfen kann, ob sie ein gültiges Urbild x' mit $H(x') = H(x)$ gefunden hat. Dann benötigt sie noch mindestens die Zeit 2^n, um ein gültiges Urbild zu finden.

Wir definieren die Beteiligten exakt:

- Der Angreifer auf die Hashfunktion ist $\texttt{FindPreimage}_H : y \in \mathbb{B}^{m+n} \mapsto x \in \mathbb{B}^*$. Er ist erfolgreich, wenn $H(x) = y$.
- Das Orakel $\mathcal{O}_{H,x} : x' \in \mathbb{B}^* \mapsto H(x) \stackrel{?}{=} H(x')$.
- Die Angreiferin $\texttt{FindOtherBits}_{H,\mathcal{O}_{H,x}} : \bar{x} \in \mathbb{B}^m \mapsto x' \in \mathbb{B}^{m+n} \cup \{\bot\}$. Sie ist erfolgreich, wenn $x'_{0..m} = \bar{x}$ und $\mathcal{O}_{H,x}(x') = \texttt{true}$.

Beweis (*Widerspruch*). Angenommen, es existierte eine Angreiferin, die nur die Zeit 2^l mit $l < n$ benötigt. Dann existiert folgender Algorithmus für $\texttt{FindPreimage}_H(y)$:

Führe für alle $\bar{x} \in \mathbb{B}^m$ aus: Rufe die Angreiferin $\texttt{FindOtherBits}(\bar{x})$ auf. Befragt sie das Orakel mit einem Wert a, gebe zurück, ob $H(a) \stackrel{?}{=} y$. Gibt die Angreiferin ein gültiges x' aus, breche die Schleife ab und gebe dieses x' zurück.

Wir können annehmen, dass ein $x \in \mathbb{B}^{m+n}$ mit $H(x) = y$ existiert. Daher entspricht die Erfolgswahrscheinlichkeit des Algorithmus der der Angreiferin, da eines der $\bar{x}$ gleich den ersten m Bits von x ist. Der angegebene Algorithmus benötigt die Zeit $2^{m+l} < 2^{m+n}$, was ein Widerspruch zur angenommenen Kollisionsresistenz ist. □

Die Anforderungen von Lemma 5.1 sind hier gegeben, es gilt $m = |\mu_{\text{pq}}| = 128$ und $n = |\mu_{\text{kl}}| = 128$. Das Lemma lässt sich identisch auch für den Fall beweisen, dass die letzten n Bits bekannt und die ersten m gesucht sind. Daraus folgt, dass auch noch ein Sicherheitsniveau von 128 bit bleibt, wenn eines der beiden Verfahren gebrochen ist.

5.5 Variante mit später Kombination

Das Verfahren besteht aus folgenden Teilen:

- KEM eines Post-Quanten-Verfahrens. Alle beschriebenen Verfahren stellen einen KEM zur Verfügung.

Algorithmus 5.3 Verschlüsselung

Eingabe: $pk_{\text{hy}} = (pk_{\text{kl}} = (N, e),\ pk_{\text{pq}})$ öffentlicher Schlüssel
Eingabe: $\text{msg} \in \mathsf{MSP}_{\text{sym}}$ Nachricht
Ausgabe: $ct \in \mathsf{CSP}_{\text{hy}} = \mathsf{CSP}_{\text{kl}} \times \mathsf{CSP}_{\text{pq}} \times \mathsf{CSP}_{\text{sym}}$ Ciphertext

1: $r_{\text{kl}} \xleftarrow{\$} \{0 \ldots N-1\}$ RSA-Verkapselung
2: $\mu_{\text{kl}} \leftarrow \texttt{KDF}(r_{\text{kl}})$
3: $\text{ct}_{\text{kl}} \leftarrow \texttt{RSA.enc}(\text{pk}_{\text{kl}}, r_{\text{kl}})$
4: $\mu_{\text{pq}}, ct_{\text{pq}} \leftarrow \texttt{PQ.encaps}(\text{pk}_{\text{pq}})$ Verkapselung des PQ-Verfahrens
5: $\text{k}_{\text{sym}} \leftarrow \text{H}(\mu_{\text{kl}}, \text{ct}_{\text{pq}}) \oplus \text{H}(\mu_{\text{pq}}, \text{ct}_{\text{kl}})$ KEM-Combiner $[\![..]\!]$
6: $\text{ct}_{\text{sym}} \leftarrow \texttt{AES-GCM.enc}(\text{k}_{\text{sym}}, \text{msg}, \text{IV} = \vec{0})$ symmetrische Verschlüsselung
7: **return** $\text{ct}_{\text{kl}}, \text{ct}_{\text{pq}}, \text{ct}_{\text{sym}}$

Algorithmus 5.4 Entschlüsselung

Eingabe: $\text{sk}_{\text{hy}} = (\text{sk}_{\text{kl}} = (N, d),\ \text{sk}_{\text{pq}})$ geheimer Schlüssel
Eingabe: $\text{ct} = (\text{ct}_{\text{kl}}, \text{ct}_{\text{pq}}, \text{ct}_{\text{sym}}) \in \mathsf{CSP}_{\text{hy}}$ Ciphertext
Ausgabe: $\text{msg} \in \mathsf{MSP}_{\text{sym}} \cup \{\perp\}$ Nachricht oder Fehlersymbol

1: `failure` $\leftarrow$ `false` RSA-Entkapselung
2: $r_{\text{kl}} \leftarrow \texttt{RSA.dec}(\text{sk}_{\text{kl}}, \text{ct}_{\text{kl}})$
3: **if** $r_{\text{kl}} \notin \{0 \ldots N-1\}$**:**
4: `failure` $\leftarrow$ `true`
5: $\mu_{\text{kl}} \leftarrow \texttt{KDF}(r_{\text{kl}})$
6: $\mu_{\text{pq}} \leftarrow \texttt{PQ.decaps}(\text{ct}_{\text{pq}})$ PQ-Entkapselung
7: **if** $\mu_{\text{pq}} = \perp$**:**
8: `failure` $\leftarrow$ `true`
9: $\text{k}_{\text{sym}} \leftarrow \text{H}(\mu_{\text{kl}}, \text{ct}_{\text{pq}}) \oplus \text{H}(\mu_{\text{pq}}, \text{ct}_{\text{kl}})$ KEM-Combiner $[\![..]\!]$
10: $\text{msg} \leftarrow \texttt{AES-GCM.dec}(\text{k}_{\text{sym}}, \text{ct}_{\text{sym}})$ symmetrische Entschlüsselung
11: **if** $k_{\text{sym}} = \perp$**:**
12: `failure` $\leftarrow$ `true`
13: **if** `failure`**:**
14: **return** $\perp$
15: **else:**
16: **return** msg

- RSA-KEM [Sho01a, § 20; RFC5990, Appendix A]. Diese Methode ist, obwohl nur die Lehrbuch-Form von RSA benutzt wird, IND-CCA-sicher [Sho01a, § 3, § 20]. (In der Beschreibung in [RFC5990] wird der bei der Verkapselung zurückgegebene Schlüssel auch direkt für symmetrische Verschlüsselung verwendet, was hier nicht nötig ist, wir verwenden nur den RSA-bezogenen Teil der Beschreibung.)

  Eine generisches Verfahren, aus einem PKE einen KEM zu bauen, findet sich in [Den03, Table 5], RSA-KEM ist aber direkt auf RSA angepasst (insbesondere wird der gesamte RSA-Nachrichtenraum ausgenutzt) und eignet sich daher besser.
- KEM-Combiner

  Federico Giacon, Felix Heuer und Bertram Poettering beschreiben in [GHP18] verschiedene Wege („Combiner"), mehrere KEMs derart zu einem zusammenzufassen, dass das Gesamt-KEM noch sicher ist, wenn alle bis auf eines gebrochen sind. Im Gegensatz zu den in § 5.1.3 beschriebenen Verfahren erhalten diese Combiner die IND-CCA-Sicherheit der Basis-KEMs. Dies ist wichtig, da bei dieser Variante nach der Kombination keine Transformation mehr stattfindet.

  Ich nutze das in Abschnitt 5.2 von [GHP18] beschriebene Verfahren: den *parallel combiner* mit $[\![H, H]\!]$ als *core function* (dabei ist H eine Hashfunktion, s. u.). Auch wenn hier eine pseudozufällige Funktion genügen würde, verwende ich eine Hashfunktion, da in den restlichen Teilen des Verfahrens (RSA-KEM, Transformationen) sowieso eine Hashfunktion benötigt wird und so die Anzahl verwendeter Primitiven reduziert werden kann.
- Datentransport mittels AES-GCM mit 256-bit-Schlüsseln. Die Nutzung eines zufälligen Initialisierungsvektors (IV) ist nicht nötig, da bei jeder Verschlüsselung ein neuer Schlüssel verwendet wird.
- Eine Hashfunktion $H : \mathbb{B}^* \to \mathbb{B}^{256}$ und eine Schlüsselableitungsfunktion $KDF : \mathbb{B}^* \to \mathbb{B}^{256}$. Letztere sollte möglichst auf H basieren, um die Anzahl verwendeter Primitiven gering zu halten.

Die Schlüsselgeneration ist die triviale Kombination der Schlüsselgeneration von RSA bzw. dem PQ-Verfahren. Die beiden öffentlichen bzw. die beiden privaten Schlüssel können einfach konkateniert werden, solange sie eine feste Länge haben (dies ist bei allen verwendeten Verfahren der Fall).

Die Verschlüsselung ist in Algorithmus 5.3 beschrieben, die Entschlüsselung in Algorithmus 5.4. Bei der Implementierung der Entschlüsselung ist darauf zu achten, dass ein Angreifer nicht (z. B. durch Laufzeitmessungen) zwischen verschiedenen Fehlerursachen unterscheiden kann.

Robustheit

Hier gibt es weniger Variationsmöglichkeiten als beim oben beschriebenen Verfahren (siehe Seite 62), da keine Transformation verwendet wird; die Robustheit wird von

der Transformation des verwendeten Verfahrens geerbt. Die AEAD-Ausnutzung ist aber auch hier möglich (siehe dort).

5.5.1 Sicherheit

Die Sicherheit dieses Verfahrens lässt sich direkt aus der der verwendeten Komponenten ableiten.

- Beide KEMs sind IND-CCA-sicher. Das gilt sowohl für die PQ-KEMs (siehe Kapitel 4) als auch für RSA-KEM (siehe oben).
- Die Kombination ist sicher, wenn mindestens ein KEM IND-CCA-sicher ist. Das ist genau die Anforderung, die wir benötigen.

  Es ist zu beachten, dass das Paper [GHP18] noch recht jung ist, sich also noch Schwächen in der Beweisführung und ähnliches herausstellen könnten und Modifikationen nötig sind. Es ist aber wahrscheinlich, dass diese Bedenken nur auf die IND-CCA-Sicherheit zutreffen – die Sicherheit gegen passive Angriffe ist vergleichsweise trivial zu erreichen, es ist also unwahrscheinlich, dass diese nicht erreicht wird.
- AES-GCM erfüllt das Kriterium der authentifizierten Verschlüsselung [MV05][7], was stärker ist als IND-CCA [KL15, § 4.5.4]. Die Kombination eines IND-CCA-sicheren KEM und eines IND-CCA-sicheren symmetrischen Verfahrens ist ebenfalls IND-CCA-sicher [CS03, Thm. 5]. Das Gesamtverfahren bietet also IND-CCA-Sicherheit.

[7] Im zitierten Beweis wird angenommen, dass AES eine pseudozufällige Permutation ist. Dies ist nicht bewiesen, wird aber sehr stark angenommen, z. B. [KL15, S. 225].

6 Zusammenfassung und Ausblick

Ich habe in der vorliegenden Arbeit vier relativ weit entwickelte und praktisch einsetzbare Post-Quanten-Verfahren vorgestellt und verglichen.

Drei der Verfahren basieren auf dem LWE-Problem (*Learning With Errors*, § 3.1.1, S. 16), das auf Probleme in Gittern zurückzuführen ist, und sind damit der Klasse der gitterbasierten Verfahren zuzuordnen. Dieses Feld ist vergleichsweise jung: Die Verwendung von Gittern für kryptografische Zwecke wurde 1996 erstmals vorgeschlagen, das LWE-Problem wurde erst 2005 spezifiziert. Die Verfahren sind:

- **Frodo**, basierend auf dem LWE-Problem, das direkt auf Gitterprobleme zurückführbar ist. Der Preis hierfür ist Effizienz: Die Größe der ausgetauschten Daten liegt etwa eine Größenordnung höher als bei den beiden folgenden Verfahren, die Rechenzeit liegt sogar knapp zwei Größenordnungen darüber. Frodo ist in Abschnitt 4.4 beschrieben.
- **NewHope** und **Kyber**, basierend auf Varianten von LWE über einen Polynomring bzw ein Polynomring-Modul (§ 3.1.1, S. 18); die Sicherheit kann nur auf bestimte Spezialisierungen der Gitterprobleme zurückgeführt werden, deren Sicherheit weniger gut erforscht ist. Beide Verfahren sind sehr effizient, sie unterscheiden sich hierin kaum (Kyber ist etwas langsamer, NewHope benötigt leicht mehr Bandbreite). Siehe Abschnitte 4.5 und 4.6.

Das vierte Verfahren ist **Classic McEliece**, es basiert auf fehlerkorrigierenden Codes. Nachteil des Verfahrens sind hohe Einmalkosten (großer öffentlicher Schlüssel, langsame Schlüsselerzeugung); Ver- und Entschlüsselung sind jedoch extrem schnell und der Ciphertext ist mit Abstand der kleinste sämtlicher Verfahren. Es ist auch in Bezug auf die Sicherheit ungeschlagen – diese lässt sich direkt auf die des seit 1973 ungebrochenen McEliece-Verfahrens zurückführen.

Welches der Verfahren für einen bestimmten Zweck eingesetzt werden sollte, hängt von den genauen Anforderungen ab und muss im Einzelfall entschieden werden. Der in dieser Arbeit vorgenommene Vergleich kann dabei eine Hilfe sein.

Wegen des laufenden PQ-Standardisierungsverfahrens des NIST ist zu erwarten, dass die Verfahren in der nächsten Zeit von vielen Expert_innen analysiert werden. Spätestens wenn hierbei keine Schwächen gefunden werden, sollte begonnen werden, einige der Verfahren in etablierte Kryptobibliotheken zu integrieren, um ihren Einsatz im

Alltag möglich zu machen. Auch hierfür ist Zeit einzuplanen: Eine sichere Implementierung ist nicht trivial, beispielsweise ist an vielen Stellen (z. B. bei den Transformationen) eine sorgfältige Programmierung nötig, um Timing-Angriffe zu verhindern.

Um den Umstieg auf Post-Quanten-Kryptografie zu beschleunigen, können Hybridverfahren eingesetzt werden. Hierbei wird ein klassisches mit einem quantencomputerresistenten Verfahren so kombiniert, dass die Verschlüsselung sicher ist, solange keines der Verfahren gebrochen ist. Würden Schwächen im PQ-Verfahren bekannt, sind die Daten weiterhin noch durch das klassische Verfahren geschützt; die Sicherheit ist also in jedem Fall mindestens so hoch wie bei weiterer ausschließlicher Nutzung von klassischen Verfahren.

In der vorliegenden Arbeit habe ich hierzu zwei Varianten zur Kombination zweier Verfahren zu einem Hybrid-Verfahren entwickelt und verglichen (§ 5.3), beide sind praktisch umsetzbar. Die erste (in Abschnitt 5.4 vorgestellte) Variante ist etwas effizienter, da nur eine Transformation[1] nötig ist. Bei der zweiten Variante (§ 5.5) werden beide Verfahren getrennt transformiert, was etwas aufwändiger ist. Es können aber dadurch besser an das jeweilige Verfahren angepasste Transformationen verwendet werden; Außerdem werden nur bewiesene und veröffentlichte Techniken genutzt, die direkt miteinander verknüpft sind, während bei der ersten Variante ein (eher trivialer) Zwischenschritt erst in dieser Arbeit entwickelt und bewiesen wurde. Die zweite Variante ist also aus Sicherheitserwägungen tendenziell zu bevorzugen.

Die beiden Varianten sind nicht konkret für ein bestimmtes Verfahren implementiert, sondern allgemein definiert, wobei ich auf die Eigenschaften der verschiedenen Verfahren Rücksicht genommen habe. Eine endgültige Entscheidung zwischen den beiden Varianten wird daher hier nicht vorgenommen, hierfür ist eine genauere Abwägung unter Berücksichtigung des verwendeten PQ-Verfahrens nötig, die erst im Rahmen der konkreten Implementierung erfolgen kann.

[1] Transformationen dienen dazu, aus einem nur schwach (z. B. nur gegen passive Angriffe) sicheren Verfahren ein aktiv sicheres zu konstruieren. Siehe Abschnitt 5.2.

Literatur

Die Spezifikationen der Einreichungen zum NIST-Standardisierungsprozess sind wegen der Vielzahl der Autor_innen und zur besseren Erkennbarkeit nicht mit Autor-Jahr-Kürzeln, sondern mit einer Kurzform des Verfahrens bezeichnet. Für Einreichungen, bei denen ein Link fehlt, ist die Spezifikation auf der Webseite des NIST herunterladbar: https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions .

Ich habe mich darum bemüht, zu jeder Quelle – abgesehen von Lehrbüchern u. dgl. – einen Link zu einer Open-Access-Downloadmöglichkeit zu finden, meistens ist dies das sehr hilfreiche ePrint-Archiv der IACR (*International Association for Cryptologic Research*). Für Quellen, die nicht offen verfügbar sind, ist die DOI angegeben, damit eine einfache Suche in alternativen Open-Access-Datenbanken möglich ist.

[AD97] Miklós Ajtai und Cynthia Dwork: *A Public-key Cryptosystem with Worst-case/Average-case Equivalence*. In: Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing. STOC '97. ACM, 1997, Seiten 284–293. DOI: 10.1145/258533.258604.

[NewH] Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Thomas Pöppelmann, Peter Schwabe und Douglas Stebila: *NewHope. Algorithm Specifications and Supporting Documentation*. Version 1.01. Einreichung zur NIST-PQC-Standardisierung. 02.12.2018. https://newhopecrypto.org/data/NewHope_2018_12_02.pdf.

[ADPS16a] Erdem Alkim, Léo Ducas, Thomas Pöppelmann und Peter Schwabe: *Post-quantum Key Exchange - A New Hope*. In: USENIX Security 2016. USENIX Association, 08/2016, Seiten 327–343. https://eprint.iacr.org/2015/1092.

[ADPS16b] Erdem Alkim, Léo Ducas, Thomas Pöppelmann und Peter Schwabe: *NewHope without reconciliation*. 12/2016. https://eprint.iacr.org/2016/1157.

[Fro] Erdem Alkım, Joppe W. Bos, Léo Ducas, Patrick Longa, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Chris Peikert, Ananth Raghunathan und Douglas Stebila: *FrodoKEM: Learning With Errors Key Encapsulation. Algorithm Specifications And Supporting Documentation*. Einreichung zur NIST-PQC-Standardisierung. 30.11.2017. https://frodokem.org/files/FrodoKEM-specification-20171130.pdf.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert und Amit Sahai: *Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems*. In: CRYPTO 2009. Band 5677. LNCS. Springer, Heidelberg, 08/2009, Seiten 595–618. DOI: 10.1007/978-3-642-03356-8_35.

[Arm18] Lucian Armasu: *IonQ Quantum Computer Delivers More Processing Power Than Google's*. 17.12.2018. https://www.tomshardware.com/news/ionq-trapped-ion-quantum-computer-google,38255.html (besucht am 21.04.2019).

[Kyb] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler und Damien Stehlé: *CRYSTALS-Kyber. Algorithm Specifications And Supporting Documentation*. Einreichung zur NIST-PQC-Standardisierung. 30.11.2017. https://pq-crystals.org/kyber/data/kyber-specification.pdf.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval und Phillip Rogaway: *Relations Among Notions of Security for Public-Key Encryption Schemes*. In: CRYPTO'98. Band 1462. LNCS. Springer, Heidelberg, 08/1998, Seiten 26–45. DOI: 10.1007/BFb0055718.

[BR95] Mihir Bellare und Phillip Rogaway: *Optimal asymmetric encryption*. In: EUROCRYPT'94. Springer, 1995, Seiten 92–111. https://cseweb.ucsd.edu/~mihir/papers/oae.pdf.

[BMvT78] Elwyn Berlekamp, Robert McEliece und Henk van Tilborg: *On the inherent intractability of certain coding problems*. In: IEEE Transactions on Information Theory 24.3 (05/1978), Seiten 384–386. https://authors.library.caltech.edu/5607/1/BERieeetit78.pdf.

[cME] Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer und Wen Wang: *Classic McEliece: conservative code-based cryptography*. Einreichung zur NIST-PQC-Standardisierung. 29.11.2017. https://classic.mceliece.org/nist/mceliece-20171129.pdf.

[pqRSA] Daniel J. Bernstein, Josh Fried, Nadia Heninger, Paul Lou und Luke Valenta: *Post-quantum RSA-Encryption*. Einreichung zur NIST-PQC-Standardisierung. 2017.

[BBD09] Daniel Bernstein, Johannes Buchmann und Erik Dahmen, Herausgeber: *Post-Quantum Cryptography*. Springer, 2009. ISBN: 978-3-540-88701-0. https://www.springer.com/de/book/9783540887010.

[Ble98] Daniel Bleichenbacher: *Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1*. In: CRYPTO'98. Band 1462. LNCS. Springer, Heidelberg, 08/1998, Seiten 1–12. DOI: 10.1007/BFb0055716.

[Böc16] Hanno Böck: *New Hope: Google testet Post-Quanten-Algorithmus*. In: Golem.de (08.07.2016). https://www.golem.de/news/new-hope-google-testet-post-quanten-algorithmus-1607-121989.html (besucht am 02.04.2018).

[BT17] Jonathan Bootle und Mehdi Tibouchi: *Cryptanalysis of Compact-LWE*. Cryptology ePrint Archive. 2017. https://eprint.iacr.org/2017/742.

[Fro16] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan und Douglas Stebila: *Frodo: Take off the Ring! Practical, Quantum-Secure Key Exchange from LWE*. In: ACM CCS 2016. ACM Press, 10/2016, Seiten 1006–1018. https://eprint.iacr.org/2016/659.

[Buc16] Johannes Buchmann: *Einführung in die Kryptographie*. Springer, 2016. ISBN: 978-3-642-39774-5. https://www.springer.com/de/book/9783642397745.

[BDS09] Johannes Buchmann, Erik Dahmen und Michael Szydlo: *Hash-based Digital Signature Schemes*. In: Post-Quantum Cryptography. Herausgegeben von Daniel Bernstein, Johannes Buchmann und Erik Dahmen. Springer, 2009, Seiten 35–93. ISBN: 978-3-540-88701-0. https://www.springer.com/de/book/9783540887010.

[CS03] Ronald Cramer und Victor Shoup: *Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack*. In: SIAM Journal on Computing 33.1 (2003), Seiten 167–226. https://www.shoup.net/papers/cca2.pdf.

[Den03] Alexander W. Dent: *A Designer's Guide to KEMs*. In: 9th IMA International Conference on Cryptography and Coding. Band 2898. LNCS. Springer, Heidelberg, 12/2003, Seiten 133–151. https://eprint.iacr.org/2002/174.

[DH76] Whitfield Diffie und Martin Hellman: *New Directions in Cryptography*. In: IEEE Transactions on Information Theory 22.6 (11/1976), Seiten 644–654. https://ee.stanford.edu/~hellman/publications/24.pdf.

[DXL12] Jintai Ding, Xiang Xie und Xiaodong Lin: *A Simple Provably Secure Key Exchange Scheme Based on the Learning with Errors Problem*. Cryptology ePrint Archive, Report 2012/688, Version 20140729:180116. 2012. https://eprint.iacr.org/2012/688.

[NIST38d] Morris Dworking: *NIST Special Publication 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. National Institute of Standards und Technology, 11/2007. https://csrc.nist.gov/publications/detail/sp/800-38d/final.

[FO13] Eiichiro Fujisaki und Tatsuaki Okamoto: *Secure Integration of Asymmetric and Symmetric Encryption Schemes*. In: Journal of Cryptology 26.1 (01/2013), Seiten 80–101. DOI: 10.1007/s00145-011-9114-1. Leicht verbesserte Version des Original-Papers aus CRYPTO99.

[GPV08] Craig Gentry, Chris Peikert und Vinod Vaikuntanathan: *Trapdoors for hard lattices and new cryptographic constructions*. In: 40th ACM STOC. ACM Press, 05/2008, Seiten 197–206. https://eprint.iacr.org/2007/432.

[GHP18] Federico Giacon, Felix Heuer und Bertram Poettering: *KEM Combiners*. In: PKC 2018. Springer, 2018, Seiten 190–218. https://eprint.iacr.org/2018/024.

[GM84] Shafi Goldwasser und Silvio Micali: *Probabilistic encryption*. In: Journal of Computer and System Sciences 28.2 (1984), Seiten 270–299. https://www.sciencedirect.com/science/article/pii/0022000084900709.

[GLRS16] Markus Grassl, Brandon Langenberg, Martin Roetteler und Rainer Steinwandt: *Applying Grover's Algorithm to AES: Quantum Resource Estimates*. In: Post-Quantum Cryptography. Springer, 2016, Seiten 29–43. https://arxiv.org/abs/1512.04965.

[Gro96] Lov K. Grover: *A Fast Quantum Mechanical Algorithm for Database Search*. In: Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing. STOC '96. ACM, 1996, Seiten 212–219. https://arxiv.org/abs/quant-ph/9605043.

[HV09] Sean Hallgren und Ulrich Vollmer: *Quantum computing*. In: Post-Quantum Cryptography. Herausgegeben von Daniel Bernstein, Johannes Buchmann und Erik Dahmen. Springer, 2009, Seiten 15–34. ISBN: 978-3-540-88701-0. https://www.springer.com/de/book/9783540887010.

[Hof14] Dirk Hoffmann: *Einführung in die Informations- und Codierungstheorie*. Springer, 2014. ISBN: 978-3-642-54003-5. https://www.springer.com/de/book/9783642540028.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns und Eike Kiltz: *A Modular Analysis of the Fujisaki-Okamoto Transformation*. In: Theory of Cryptography. Springer, 2017, Seiten 341–371. https://eprint.iacr.org/2017/604.

[RFC8391] A. Huelsing, D. Butin, S. Gazdag, J. Rijneveld und A. Mohaisen: *XMSS: eXtended Merkle Signature Scheme*. RFC 8391. IETF, 05/2018. https://tools.ietf.org/html/rfc8391.

[KL07] Jonathan Katz und Yehuda Lindell: *Introduction to Modern Cryptography*. Erste Ausgabe, 2007. ISBN: 978-1-58488-551-1.

[KL15] Jonathan Katz und Yehuda Lindell: *Introduction to Modern Cryptography*. Zweite Ausgabe, 2015. ISBN: 978-1-4665-7027-6.

[Kob87] Neal Koblitz: *Elliptic Curve Cryptosystems*. In: Mathematics of Computation 48.177 (01/1987), Seiten 203–209.

[Kri15] Klaus Kriegel: *Mathematik für Informatiker II: Lineare Algebra*. Vorlesungsskript, Freie Universität Berlin. Sommersemester/2015.

[BSI2102] *Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. Version 2019-01. Bundesamt für Sicherheit in der Informationstechnik, 22.02.2019. https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr02102/index_htm.html.

[Lan17] Tanja Lange: *Quantum cryptanalysis – the catastrophe we know and don't know*. Vortrag bei „Catacrypt". 29.04.2017. https://hyperelliptic.org/tanja/vortraege/catacrypt.pdf.

[LS15] Adeline Langlois und Damien Stehlé: *Worst-case to average-case reductions for module lattices*. In: Designs, Codes and Cryptography 75.3 (06/2015), Seiten 565–599. https://eprint.iacr.org/2012/090.

[LLPX18] Haoyu Li, Renzhang Liu, Yanbin Pan und Tianyuan Xie: *Ciphertext-Only Attacks against Compact-LWE Submitted to NIST PQC Project*. Cryptology ePrint Archive. 2018. https://eprint.iacr.org/2018/020.

[LDW94] Yuanxing Li, Robert H. Deng und Xinmei Wang: *On the equivalence of McEliece's and Niederreiter's public-key cryptosystems*. In: IEEE Trans. Information Theory 40.1 (1994), Seiten 271–273. DOI: 10.1109/18.272496.

[LP11] Richard Lindner und Chris Peikert: *Better Key Sizes (and Attacks) for LWE-Based Encryption*. In: CT-RSA 2011. Springer, Heidelberg, 02/2011, Seiten 319–339. https://eprint.iacr.org/2010/613.

[LLKN17] Dongxi Liu, Nan Li, Jongkil Kim und Surya Nepal: *Compact-LWE: a Public Key Encryption Scheme*. Einreichung zur NIST-PQC-Standardisierung. 2017.

[LPR10] Vadim Lyubashevsky, Chris Peikert und Oded Regev: *On Ideal Lattices and Learning with Errors over Rings*. In: EUROCRYPT 2010. Band 6110. LNCS. Springer, Heidelberg, 2010, Seiten 1–23. DOI: 10.1007/978-3-642-13190-5_1.

[LPR10F] Vadim Lyubashevsky, Chris Peikert und Oded Regev: *On Ideal Lattices and Learning with Errors over Rings*. Vortrag auf der EUROCRYPT 2010 (Folien zu [LPR10]). https://crypto.orange-labs.fr/events/eurocrypt2010/talks/slides-ideal-lwe.pdf.

[McE78] Robert J. McEliece: *A Public-Key Cryptosystem Based on Algebraic Coding Theory*. In: JPL DSN Progress Report 44 (02/1978). https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF.

[MV05] David A. McGrew und John Viega: *The Security and Performance of the Galois/Counter Mode (GCM) of Operation*. In: Progress in Cryptology - INDOCRYPT 2004. Springer Berlin Heidelberg, 2005, Seiten 343–355. https://eprint.iacr.org/2004/193.

[MR09] Daniele Micciancio und Oded Regev: *Lattice-based Cryptography*. In: Post-Quantum Cryptography. Herausgegeben von Daniel Bernstein, Johannes Buchmann und Erik Dahmen. Springer, 2009, Seiten 147–191. ISBN: 978-3-540-88701-0. https://www.springer.com/de/book/9783540887010.

[Mil86] Victor S. Miller: *Use of Elliptic Curves in Cryptography*. In: Advances in Cryptology — CRYPTO '85 Proceedings. Springer, 1986, Seiten 417–426. DOI: 10.1007/3-540-39799-X_31.

[Moo17] Dustin Moody: *The Ship Has Sailed. The NIST Post-Quantum Crypto "Competition"*. Vortrag bei „Asiacrypt". 2017. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/asiacrypt-2017-moody-pqc.pdf.

[RFC8017] K. Moriarty, B. Kaliski, J. Jonsson und A. Rusch: *PKCS #1: RSA Cryptography Specifications Version 2.2*. RFC 8017. IETF, 11/2016. https://tools.ietf.org/html/rfc8017.

[NW17] Ruben Niederhagen und Michael Waidner: *Practical Post-Quantum Cryptography. White Paper*. 2017. http://publica.fraunhofer.de/documents/N-481797.html.

[Nie86] Harald Niederreiter: *Knapsack-type cryptosystems and algebraic coding theory*. In: Prob. Control and Inf. Theory 15.2 (1986), Seiten 159–166. http://real-j.mtak.hu/7997/.

[NIS16] NIST: *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. 12/2016. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[NIS19] NIST: *Post-Quantum Cryptography, Round 2 Submissions*. 30.01.2019. https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions (besucht am 09.04.2019).

[OS09] Raphael Overbeck und Nicolas Sendrier: *Code-based cryptography*. In: Post-Quantum Cryptography. Herausgegeben von Daniel Bernstein, Johannes Buchmann und Erik Dahmen. Springer, 2009, Seiten 95–145. ISBN: 978-3-540-88701-0. https://www.springer.com/de/book/9783540887010.

[Pei14] Chris Peikert: *Lattice Cryptography for the Internet*. In: Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014. Springer, Heidelberg, 10/2014, Seiten 197–219. https://eprint.iacr.org/2014/070.

[PG14] Thomas Pöppelmann und Tim Güneysu: *Towards Practical Lattice-Based Public-Key Encryption on Reconfigurable Hardware*. In: SAC 2013. Band 8282. LNCS. Springer, Heidelberg, 08/2014, Seiten 68–85. https://www.ei.ruhr-uni-bochum.de/media/sh/veroeffentlichungen/2013/08/14/lwe_encrypt.pdf.

[RFC5990] J. Randall, B. Kaliski, J. Brainard und S. Turner: *Use of the RSA-KEM Key Transport Algorithm in the Cryptographic Message Syntax (CMS)*. RFC 5990. IETF, 09/2010. https://tools.ietf.org/html/rfc5990.

[Reg05] Oded Regev: *On Lattices, Learning with Errors, Random Linear Codes, and Cryptography*. In: J. ACM 56.6 (11/2009), 34:1–34:40. https://cims.nyu.edu/~regev/papers/qcrypto.pdf. (2009 erschienene Erweiterung des bekannteren Papers aus 2005, daher als Reg05 zitiert.)

[RSA78] Ronald L. Rivest, Adi Shamir und Leonard M. Adleman: *A Method for Obtaining Digital Signature and Public-Key Cryptosystems*. In: Communications of the Association for Computing Machinery 21.2 (1978), Seiten 120–126. https://people.csail.mit.edu/rivest/Rsapaper.pdf.

[RNSL17] Martin Roetteler, Michael Naehrig, Krysta M. Svore und Kristin Lauter: *Quantum Resource Estimates for Computing Elliptic Curve Discrete Logarithms*. In: ASIACRYPT 2017. 2017, Seiten 241–270. https://eprint.iacr.org/2017/598.

[Sho94] Peter W. Shor: *Algorithms for quantum computation: discrete logarithms and factoring*. In: Proceedings 35th Annual Symposium on Foundations of Computer Science. 11/1994, Seiten 124–134. DOI: 10.1109/SFCS.1994.365700. Erweiterte Fassung unter https://arxiv.org/abs/quant-ph/9508027.

[Sho01a] Victor Shoup: *A Proposal for an ISO Standard for Public Key Encryption*. Version 2.1. 2001. https://eprint.iacr.org/2001/112.

[Sho01b] Victor Shoup: *OAEP Reconsidered*. In: CRYPTO 2001. Band 2139. LNCS. Springer, Heidelberg, 08/2001, Seiten 239–259. https://www.shoup.net/papers/oaep.pdf.

[Sma+18] Nigel P. Smart, Michel Abdalla, Tor Erling Bjørstad, Carlos Cid, Benedikt Gierlichs, Andreas Hülsing, Atul Luykx, Kenneth G. Paterson, Bart Preneel, Ahmad-Reza Sadeghi, Terence Spies, Martijn Stam, Michael Ward, Bogdan Warinschi und Gaven Watson: *Algorithms, Key Size and Protocols Report (2018)*. 28.02.2018. http://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf.

[Wik19] Wiktionary: *Modul*. Wiktionary, Das freie Wörterbuch. 2019. https://de.wiktionary.org/wiki/Modul?oldid=6936894#Substantiv,_m (besucht am 22.04.2019).