Theoretische und experimentelle Untersuchungen zu einem hocheffizienten randomisierten Primzahltest

Diplomarbeit

zur Erlangung des akademischen Grades
Diplominformatiker

vorgelegt der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau
von

Christian Hoffmann
Matr.-Nr. 29241

betreut von
Prof. Dr.(USA) Martin Dietzfelbinger

eingereicht am 2. September 2005 in Ilmenau
Inventarisierungsnummer: 2005-09-02/081/IN99/2239

Danksagung

Auch wenn ich diese Diplomarbeit selbständig verfasst habe, wie es die Prüfungsordnung vorschreibt, habe ich dabei doch auch von der Hilfe anderer profitiert, denen ich an dieser Stelle dafür ganz herzlich danken möchte.

Ich danke Herrn Prof. Dr. Martin Dietzfelbinger für das interessante Thema und die Betreuung der Arbeit. Die lockeren und freundlichen Gespräche mit ihm waren jedes Mal ein Gewinn für mich. Seine Erklärungen haben mir ein klareres Bild der Zusammenhänge vermittelt, über Verständnisschwierigkeiten hinweggeholfen und ein Gefühl dafür gegeben, was bedeutend und was weniger bedeutsam ist. Durch seine Hinweise ist die Arbeit sehr verbessert worden.

Meinen Eltern danke ich für ihre Liebe und Unterstützung. Meinen Geschwistern und Freunden danke ich für die gemeinsam verbrachte Zeit.

Ich danke allen, die der Allgemeinheit kostenlos Informationen, Software oder andere Ergebnisse ihrer Arbeit zur Verfügung gestellt haben, so dass auch ich davon profitieren konnte. Stellvertretend für viele Personen möchte ich einige Dinge nennen, ohne die ich diese Arbeit nicht hätte erstellen können bzw. die mir sehr dabei geholfen haben: LaTeX, Debian, GCC, GMP, Gnuplot, Perl, ...

Mit diesem Dank an Menschen möchte ich einen respektvollen und ehrfürchtigen Dank an den verbinden, ohne den die Welt nicht existieren würde und von dem alles Gute kommt. Ich danke Gott, dass er durch Jesus Christus mit mir Frieden geschlossen hat und er mir viel Gutes noch zusätzlich schenkt — insbesondere alles, was es mir möglich gemacht hat, diese Diplomarbeit zu verfassen.

Zusammenfassung

Primzahltests sind Algorithmen, die Primzahlen von zusammengesetzten Zahlen unterscheiden. Die in dieser Arbeit betrachteten und praktisch relevanten Primzahltests sind Monte-Carlo-Algorithmen mit einseitigem beschränkten Fehler. Sie klassifizieren Primzahlen immer richtig, während sie zusammengesetzte Zahlen mit unabhängig von der Eingabe beschränkter Wahrscheinlichkeit auch fälschlicherweise als Primzahlen einstufen können. Durch mehrmalige Wiederholung eines solchen Primzahltests erhält man ein für die Praxis ausreichend verlässliches Verfahren zur Erkennung von Primzahlen.

Sehr häufig wird der Miller-Rabin-Primzahltest eingesetzt. Er untersucht die Verhältnisse in $\mathbb{Z}_n$: Wenn $n = p$ eine Primzahl ist, ist $\mathbb{Z}_p$ ein Körper mit p Elementen, so dass dort der Kleine Satz von Fermat sowie eine Aussage über Quadratwurzeln von 1 gelten, die bei zusammengesetzten Zahlen häufig nicht erfüllt sind. Der Miller-Rabin-Test hat eine Irrtumswahrscheinlichkeit von höchstens 1/4.

Ein neuer Primzahltest ist der randomisierte starke quadratische Frobenius-Test (RQFT), der im Mittelpunkt dieser Arbeit steht. Die mathematische Grundlage dieses Tests sind Eigenschaften von $\mathbb{Z}_n[X]/(X^2 - bX - c)$. Wenn $n = p$ eine Primzahl und $X^2 - bX - c$ ein irreduzibles Polynom ist, dann ist $\mathbb{Z}_p[X]/(X^2 - bX - c)$ ein Körper mit p^2 Elementen, in dem für den Frobenius-Homomorphismus $a \mapsto a^p$ eine Aussage gilt, die eine Verallgemeinerung des Kleinen Satzes von Fermat darstellt. Der RQFT prüft die Gültigkeit dieser Aussage und sucht nach Quadratwurzeln von 1. In [Gra98] wird gezeigt, dass die Irrtumswahrscheinlichkeit des RQFT höchstens 1/7710 beträgt. Dieser Beweis wird in der vorliegenden Arbeit ausgearbeitet, die dazu erforderlichen mathematischen Zusammenhänge werden vorher bereitgestellt.

Die Laufzeit des RQFT entspricht nach [Gra98] asymptotisch der dreifachen Laufzeit des Miller-Rabin-Tests. Der dazugehörige Beweis wird aufbereitet. Dabei wird ein in [Gra98] vernachlässigtes Detail (Invertierung in $\mathbb{Z}_n[X]/(X^2 - bX - c)$) in die Dar-

stellung mit einbezogen und gezeigt, dass die Laufzeitschranke von asymptotisch drei Miller-Rabin-Tests trotzdem gültig ist.

Weiterhin wird in dieser Arbeit die Laufzeit des RQFT mit der Laufzeit des Miller-Rabin-Tests experimentell verglichen. Ein wesentliches Ergebnis der Untersuchungen ist, dass für Zahlen bis zu 4096 Bit die Laufzeit des RQFT in der Implementierung nach [Gra98] etwa der von vier Miller-Rabin-Tests entspricht.

Der einfache quadratische Frobenius-Test wird in [CP01] behandelt. Inhaltlich unterscheidet er sich vom RQFT hauptsächlich darin, dass er nicht nach Quadratwurzeln von 1 sucht. Die experimentellen Untersuchungen in der vorliegenden Arbeit zeigen, dass die Laufzeit des einfachen quadratischen Frobenius-Tests in der Implementierung nach [CP01] praktisch der von drei Iterationen des Miller-Rabin-Tests entspricht.

Schließlich wird in dieser Arbeit gezeigt, dass sich die in [CP01] für den einfachen quadratischen Frobenius-Test angegebene Implementierungsstrategie auch für den RQFT nutzen lässt. Experimente mit dieser Implementierung des RQFT zeigen, dass der RQFT auf diese Weise für Zahlen bis zu 4096 Bit nicht mehr Zeit benötigt als drei Iterationen des Miller-Rabin-Tests.

Gliederung der Arbeit

In Kapitel 1 wird der praktische Nutzen von Primzahlen erläutert. Kapitel 2 berichtet dann, wie man große Primzahlen finden kann und welche Rolle Primzahltests dabei spielen. Dort wird auch der Begriff „Primzahltest“ definiert.

Die für eine Behandlung des RQFT erforderlichen mathematischen Begriffe und Zusammenhänge werden in Kapitel 3 zusammengestellt. Kapitel 4 beleuchtet die Algorithmen, auf denen die Implementierungen des RQFT aufbauen.

Kapitel 5 fasst das Wichtigste über den Fermat-Test und den Miller-Rabin-Test zusammen. Danach wird in Kapitel 6 der RQFT ausführlich behandelt. Insbesondere wird dort nachgewiesen, dass die Irrtumswahrscheinlichkeit höchstens $\frac{1}{7710}$ ist. Außerdem wird ausgehend von der Implementierung nach [Gra98] auch die [CP01] folgende schnellere Implementierung erläutert. Kapitel 7 stellt die Experimente vor, mit denen der Miller-Rabin-Test und die verschiedenen Varianten des quadratischen Frobenius-Tests miteinander verglichen wurden. Die Arbeit endet mit einer kurzen Bewertung des RQFT in Kapitel 8.

Inhaltsverzeichnis

1. Motivation — der praktische Nutzen großer Primzahlen

Kommunikation ist eine Wesensmerkmal der Menschheit. Ein beträchtlicher Teil davon geschieht in der heutigen Zeit digital. Was mitgeteilt werden soll, wird auf einem Computern erfasst und von diesem als „Datenklumpen“ an den Kommunikationspartner weitergeleitet. Dieser verfügt ebenfalls über einen Computer, der ihm die Daten in angemessener Form präsentiert und so die ursprüngliche Botschaft sichtbar werden lässt.

Der Wert einer Kommunikationsmöglichkeit hängt unter anderem davon ab, in wie weit sie bestimmte Sicherheitsmerkmale aufweist. Auf die *Authentizität* der Nachrichten können wir nicht verzichten. Denn wir wollen sicher sein, dass eine Botschaft, die wir erhalten haben, tatsächlich von der Person stammt, die als Absender angegeben ist. Eine andere Voraussetzung ist, dass die Botschaft den Empfänger so erreicht, wie wir sie verfasst haben und nicht in mehr oder weniger abgewandelter Form. Die *Integrität* der Nachricht soll gewahrt bleiben. Bei Verträgen ist *Verbindlichkeit* wichtig. Kein Vertragspartner soll abstreiten können, die im Vertrag geäußerte Willenserklärung tatsächlich abgegeben zu haben. Schließlich gibt es oft Dinge mitzuteilen, die nur der Kommunikationspartner erfahren soll und niemand sonst. Wir wünschen uns *Vertraulichkeit*.

Für die traditionellen Kommunikationswege (persönliches Gespräch, handschriftliche Kommunikation, Telefonieren, . . .) kennen wir Mittel, mit denen wir Sicherheit herstellen können, zumindest in einem für die Praxis ausreichenden Maß. Wer sich persönlich mit seinem Kommunikationspartner trifft, kann ihn gewöhnlich am Aussehen erkennen, „authentifizieren“. Probleme mit der Integrität der Kommunikation treten hier meist gar nicht auf. Zeugen dienen der Verbindlichkeit, Flüstern ermöglicht Vertraulichkeit.

Je mehr wir die digitalen Kommunikationswege nutzen, desto wichtiger wird es, auch hier für Sicherheit zu sorgen. Entsprechende Sicherheitstechniken — Algorithmen und Protokolle, die in der Software-Welt Ähnliches leisten wie Riegel, Vorhängeschlösser und Siegel in der mechanischen Welt — sind vorhanden. Einige sehr wichtige dieser Techniken benötigen große Primzahlen. Um diese bereitzustellen ist man auf Primzahltests angewiesen. Einer dieser Tests, der quadratische Frobenius-Test, ist Gegenstand dieser Arbeit. Bevor auf Primzahltests im Allgemeinen und den Frobenius-Test im Besonderen näher eingegangen wird, sollen zwei Sicherheitstechniken vorgestellt werden, die ohne große Primzahlen — und damit auch ohne Primzahltests — nicht auskommen.

1.1. Geheimnisse aufbauen mit öffentlicher Kommunikation

Ein Grundprinzip der Kryptographie ist, dass die Sicherheit der Verschlüsselung nicht durch Geheimhaltung des Verschlüsselungsverfahrens, sondern des Schlüssels hergestellt werden sollte. Damit stehen die Gesprächspartner, die vertraulich kommunizieren wollen, vor der Aufgabe, ein gemeinsames Geheimnis, nämlich einen Schlüssel, vereinbaren zu müssen. Dieser darf niemandem außer ihnen bekannt werden, er muss also selbst vertraulich sein. Das erweckt nun den Anschein, dass die Lösung eines Problems (vertrauliche Nachrichtenübermittlung) das Problem selbst wieder neu aufwirft (vertraulicher Schlüsselaustausch) und somit unbrauchbar ist.

Das ist sie natürlich nicht, denn oft gibt es durchaus vertrauliche Kommunikationsmöglichkeiten, die allerdings in gewisser Hinsicht teuer sind. Beispielsweise könnten sich Geschäftspartner, die weit voneinander entfernt arbeiten, persönlich treffen. Man kann eine solche Möglichkeit ein einziges Mal zum Schlüsselaustausch nutzen und anschließend viele Male auf einem billigeren, nicht vertraulichen Kommunikationsweg mit Hilfe des ausgetauschten Schlüssels verschlüsselt und somit vertraulich kommunizieren.

Allerdings ist es immer umständlich, eine vertrauliche Kommunikationsmöglichkeit für den Aufbau des gemeinsamen Geheimnisses nutzen zu müssen. Deshalb war es ein unglaublicher Fortschritt, als Diffie und Hellman 1976 ein Protokoll vorschlugen, mit dem zwei Partner auch über eine nicht vertrauliche Verbindung ein gemeinsames Geheimnis aufbauen können. Als erstes Beispiel für die praktische Bedeutung von Primzahlen wollen wir nun das Prinzip dieses Protokolls anhand des ursprünglichen

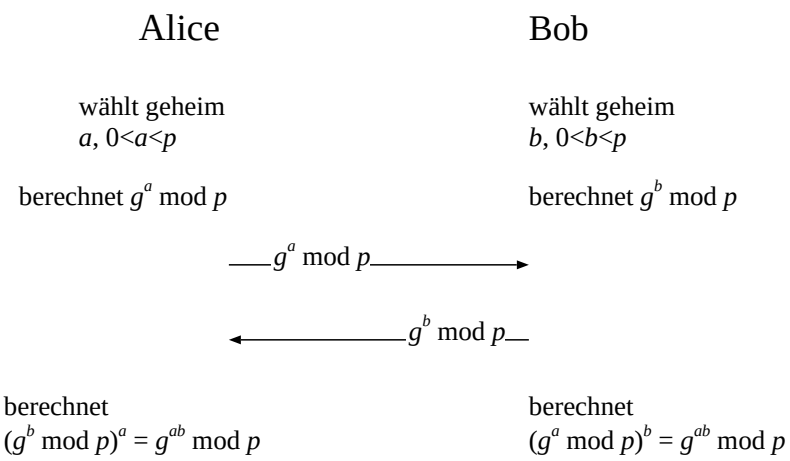

Abbildung 1.1.: Das Geheimnis $g^{ab} \bmod p$ aufbauen nach Diffie und Hellman.

Vorschlags betrachten. Für einen sinnvollen Einsatz in der Praxis müssen jedoch unbedingt zusätzlich einige Dinge beachtet werden, die beispielsweise in [FS03] diskutiert werden.

Grundlage für das Protokoll ist eine sehr große Primzahl p. Außerdem wird ein erzeugendes Element[1] g der multiplikativen Gruppe $\mathbb{Z}_p^*$ benötigt. Beide sind allgemein bekannt. Das gemeinsamen Geheimnis kommt zustande, indem jeder Partner einen geheimen Wert wählt und diese kombiniert werden. Im Einzelnen geschieht das wie folgt: Alice[2] wählt einen Wert a, $0 < a < p$. Dieser Wert ist geheim und wird niemandem mitgeteilt. Ebenso wählt Bob einen Wert b, für den das gleiche gilt. Das gemeinsame Geheimnis ist definiert als $g^{ab} \bmod p$ und wird den beiden Partnern wie folgt bekannt: Alice berechnet den Wert $g^a \bmod p$ und schickt ihn an Bob. Dieser berechnet daraus und mit Hilfe von b das gemeinsame Geheimnis $(g^a \bmod p)^b \bmod p = g^{ab} \bmod p$. Bob berechnet den Wert $g^b \bmod p$ und schickt ihn an Alice. Diese ermittelt damit und mit a ebenfalls das gemeinsame Geheimnis $(g^b \bmod p)^a \bmod p = g^{ab} \bmod p$. Die Lauscherin Eve erfährt zwar $g^a \bmod p$ und $g^b \bmod p$. Bis heute ist jedoch kein schnelles Verfahren bekannt, das daraus a, b oder $g^{ab} \bmod p$ ermittelt.

[1] Die mathematischen Begriffe werden in Kapitel 3 definiert und erläutert.

[2] Alice und Bob werden in der Kryptographie traditionell als Namen für die Kommunikationspartner verwendet. Ebenfalls traditionell ist die Bezeichnung Eve für die die Kommunikation angreifende Person.

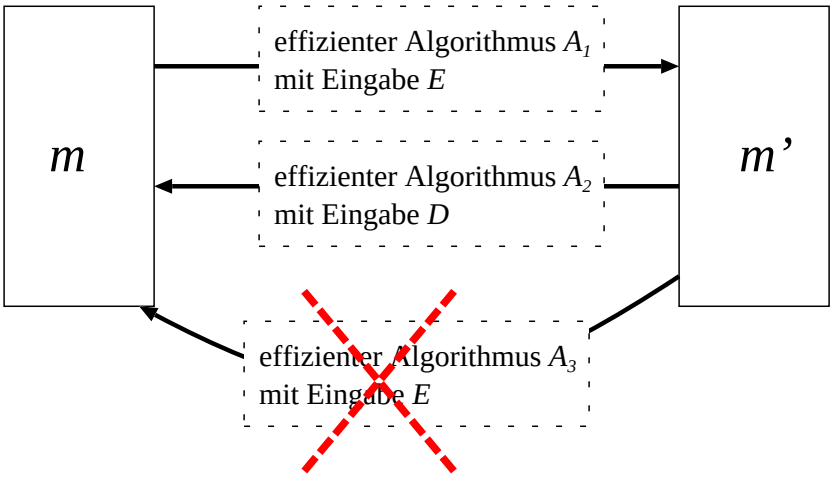

Abbildung 1.2.: Voraussetzung für ein Public-Key-Kryptosystem.

1.2. Verschlüsseln mit öffentlich bekannten Schlüsseln

Eine weitere berühmte digitale Sicherheitstechnik ist das RSA-Verfahren, das nach seinen Erfindern Ronald Rivest, Adi Shamir und Leonard Adleman benannt ist. RSA verwirklicht die Idee der Verschlüsselung mit öffentlichen Schlüsseln (Public-Key-Kryptographie).

Diese geht davon aus, dass jede an der Kommunikation beteiligte Person ein Schlüsselpaar (E, D) besitzt. E und D müssen die Voraussetzungen erfüllen, die in Abbildung 1.2 angedeutet sind: Es gibt ein effizientes Verfahren, das eine Nachricht m und den Schlüssel E als Eingabe entgegennimmt und daraus eine andere Nachricht m' herstellt. Mittels eines anderen effizienten Verfahrens, das m' und den Schlüssel D als Eingabe nutzt, kann man m wiedergewinnen. Wichtig ist, dass man ohne die Kenntnis von D aus m' die ursprüngliche Nachricht m *nicht* zurückgewinnen kann, insbesondere nicht unter Zuhilfenahme von E.

Wenn man diese Instrumente zur Verfügung hat, lässt sich Vertraulichkeit herstellen: Die Umwandlung von m nach m' mittels E entspricht dem Verschlüsseln[3]. Jedes E ist allgemein bekannt und wird deshalb öffentlicher Schlüssel genannt. Demzufolge entspricht der Schritt von m' zu m dem Entschlüsseln. D, was dazu nötig ist, muss geheim bleiben, darf also nur dem Besitzer des jeweiligen Schlüsselpaars bekannt sein. Wenn

[3]Die Bezeichnungen E und D rühren von dieser Anwendung her: E steht für encryption key, D für decryption key.

nun Alice mit Bob kommunizieren will, muss sich Alice Bobs öffentlichen Schlüssel beschaffen. Sie verschlüsselt damit dann ihre Nachrichten an Bob, die dieser mit seinem geheimen Schlüssel entschlüsselt. Um Alice zu antworten, beschafft er sich Alice' öffentlichen Schlüssel. Damit verschlüsselt er seine Antwort, die Alice wiederum mit ihrem geheimen Schlüssel dechiffriert.

Sehr vorteilhaft am Verschlüsseln mit öffentlichen Schlüsseln ist die Tatsache, dass vor der vertraulichen Kommunikation kein Geheimnis ausgetauscht werden muss. Es ist also kein initialer vertraulicher Kommunikationsweg nötig[4]. Daher kann ein Public-Key-Kryptosystem auch zum Aufbau eines gemeinsamen Geheimnisses verwendet werden, stellt also eine Alternative zum Verfahren nach Diffie-Hellman dar. Außerdem ist die Gesamtzahl aller Schlüssel in einem Kommunikationsszenario um ein Größenordnung kleiner als bei symmetrischen Verschlüsselungsverfahren: Dort benötigen n Teilnehmer $\binom{n}{2} = \Theta(n^2)$ Schlüssel, um vertraulich zu kommunizieren. Wenn öffentliche Schlüssel eingesetzt werden, sind nur $n = \Theta(n)$ Schlüssel nötig.

Public-Key-Kryptosysteme haben aber noch ein weiteres wichtiges Anwendungsszenario. Dazu müssen wir auf einige nicht formal erfasste Eigenschaften von Public-Key-Kryptosystemen zu sprechen kommen. Üblicherweise erhält man, wenn man mit einem „sinnvollen“ m (beispielsweise der Kodierung eines deutschen Briefes) beginnt und dieses mit E verschlüsselt, ein m', dem man keinen „Sinn“ entnehmen kann (im Beispiel: das nicht die Kodierung eines sinnvollen deutschen Textes ist). Die einzige sinnvolle Anwendung für ein solches m' besteht darin, es über einen unsicheren Kanal an jemanden zu versenden, der D kennt. Dieser kann m' entschlüsseln und die Botschaft verstehen, die in m enthalten ist. Nennt man dagegen die ursprüngliche Botschaft (etwa wieder den kodierten Brief) m' und wandelt ihn mittels A_2 und D in ein m um, so ergibt dieses m üblicherweise für sich genommen keinen Sinn. Wenn man aber jetzt das Paar (m', m) veröffentlicht, kann jeder nachprüfen, dass m' die Verschlüsselung von m ist. Weil niemand außer der Person, die D kennt, ein m erstellen kann, dass gerade die Verschlüsselung *dieses* sinnhaltigen m' ist, weiß man nun, dass der Besitzer von D hinter m' steht, dass er m' verfasst hat. Diese Tatsache wird genutzt, um „digitale Unterschriften“ zu erzeugen.

[4] Aber sehr wohl ein unverletzlicher: Alice muss sich sicher sein können, wirklich über Bobs öffentlichen Schlüssel zu verfügen und nicht über eine Fälschung von Eve, mit der diese Alice' Nachrichten abfangen, entschlüsseln und bei Bedarf mit Bobs echten öffentlichen Schlüsseln chiffriert an Bob weiterleiten kann. Eine solche Vorgehensweise von Eve heißt Man-in-the-Middle-Angriff.

Als zweites Beispiel für den praktischen Einsatz von Primzahlen wollen wir nun sehen, wie RSA funktioniert und ein Public-Key-Kryptosystem bereitstellt. Dabei behandeln wieder nur das Grundprinzip. Auch bei RSA muss für eine sinnvolle praktische Anwendung, etwa zum Verschlüsseln oder für digitale Unterschriften, auf einige Randbedingungen geachtet werden, die in [FS03] erläutert werden.

Um ein RSA-Schlüsselpaar (E, D) zu erzeugen, benötigt man zwei verschiedene große Primzahlen p, q. Es wird $n = pq$ gesetzt. Anschließend sucht man ein e, das teilerfremd ist zu $\varphi(n)$ und berechnet modulo $\varphi(n)$ das Inverse d zu e, also d mit $ed \equiv 1 \bmod \varphi(n)$. Damit sind alle Bestandteile des Schlüsselpaars definiert. n und e zusammen bilden nämlich den öffentlichen Schlüssel, alle übrigen Werte müssen geheim bleiben[5]. Eine Nachricht $m, 0 < m < n$ wird verschlüsselt, indem der Chiffretext $m' = m^e \bmod n$ berechnet wird. Zum Entschlüsseln wird ebenfalls exponenziert und modulo n reduziert: der Klartext ergibt sich zu $m'^d = (m^e)^d = m^{ed} = m^{k\varphi(n)+1} = (m^{\varphi(n)})^k \cdot m \equiv 1 \cdot m \bmod n$, falls $m \in \mathbb{Z}_n^*$. Falls m nicht teilerfremd zu n ist, ist es entweder durch p oder durch q teilbar. Nehmen wir an, m ist durch p teilbar. Modulo p sind dann sowohl m als auch m^e und m^{ed} kongruent zu 0. Modulo q ergibt sich $m^{ed} = m^{k\varphi(n)+1} = (m^{q-1})^{(p-1)k} \cdot m \equiv 1 \cdot m \bmod q$. Da m und m^{ed} sowohl modulo p als auch modulo q übereinstimmen, tun sie das auch modulo n.

Entspricht das RSA-Verfahren den Anforderungen an ein Public-Key-Kryptosystem? Wenn die Schlüssel bekannt sind, kann wie angegebenen m' aus m bzw. m aus m' berechnet werden. Mittels „schnellem Potenzieren“ (vgl. Abschnitt 4.3) ist dies effizient möglich. Die Frage, ob man ohne D entschlüsseln kann, ist schwieriger zu beantworten. Hier ist zunächst festzustellen, dass bis heute keine effizientes Verfahren bekannt ist, das dies ermöglicht, obwohl RSA seit über 25 Jahren bekannt ist. Weiterhin ist RSA auf eine nicht offensichtliche Art und Weise mit dem Faktorisierungsproblem verbunden: Falls es ein effizientes Verfahren gibt, mit dem man d aus e gewinnen kann, dann kann dieses zu einem neuen Verfahren erweitert werden, das n schnell in $n = pq$ zerlegt. Faktorisierung wird jedoch für ein algorithmisch recht schweres Problem gehalten. Auch hierfür gibt es keinen Beweis. Aber Faktorisierung ist ein sehr bekanntes Problem, für das schon lange Zeit und von sehr vielen verschiedenen Leuten effiziente Lösungsverfahren gesucht wurden, ohne dass bisher jemand Erfolg gehabt hätte.

[5] Insbesondere auch der Wert $\varphi(n) = (p-1)(q-1)$.

1.3. Gebräuchliche Software, die große Primzahlen benutzt

Der Diffie-Hellman-Schlüsselaustausch, aber besonders auch RSA werden in praktisch relevanter Software genutzt. Eine offensichtliche Anwendungsmöglichkeit ist Email. Mit Software wie GnuPG [GPG] oder PGP lassen sich Emails mit öffentlichen Schlüsseln verschlüsseln und damit vor Mitlesen schützen. Gleichzeitig kann man seine elektronische Post mit einer digitalen Unterschrift versehen. Wenn nun jemand unterwegs den Brief verändert, kann der Empfänger feststellen, dass die Unterschrift nicht zum Inhalt des Briefes passt. Außer dem Absender ist niemand in der Lage, eine Unterschrift im Namen des Absenders zu erzeugen. So kann die Integrität der Nachrichten gewahrt werden. All dies wird mittels RSA realisiert.

Ein weitere Anwendungsbereich ist Fernwartung von Computern [SSH]. Normalerweise authentifiziert man sich gegenüber dem entfernten Rechner durch Eingabe eines Passworts. Damit dies nicht mitgehört wird, muss eine verschlüsselte Verbindung aufgebaut werden, die wiederum einen geheimen Schlüssel benötigt. Wenn man dafür einen vertraulichen Kanal zur Verfügung hätte, könnte man auch gleich das Passwort über diesen Kanal übertragen. Der Diffie-Hellman-Schlüsselaustausch, aber auch RSA ermöglichen in dieser Situation ohne vertraulichen Kanal einen geheimen Schlüssel auszuhandeln. Weiterhin wird auch von der Möglichkeit Gebrauch gemacht, dass sich der Nutzer anstatt mit einem Passwort mit öffentlicher Verschlüsselung authentifiziert. Dazu muss er nur ein Mal dem betreffenden Computer seinen öffentlichen Schlüssel mitteilen. Alle nachfolgenden Authentifizierungen können dann darin bestehen, dass der Nutzer dem Rechner beweist, im Besitz des geheimen Schlüssels zu sein, indem er mit dem öffentlichen Schlüssel verschlüsselte Nachrichten dechiffriert. Im SSH-Protokoll sind diese Ideen verwirklicht.

Auch wer Passwörter in Web-Formulare einträgt, will nicht, dass eine der vielen Zwischenstationen, die es im Internet gibt, die Passwörter mitlesen kann. Deshalb gibt es das TLS-Verfahren (Transport Layer Security, [DA99]), das von vielen Web-Browsern und -Servern unterstützt wird. Auch hier steht vor der verschlüsselten Verbindung ein Schlüsselaustausch mit RSA oder nach Diffie-Hellman. Außerdem muss sich eine der beteiligten Parteien (meistens der Server) mit einem Zertifikat authentifizieren, damit Man-in-the-Middle-Angriffe verhindert werden.

1.4. Schutz vor gefälschten Primzahlen

Jede in der Praxis eingesetzte Sicherheitstechnik, mit der wertvolle Dinge vor Unberechtigten geschützt werden sollen, stellt für diese eine Einladung dar, den Schutz zu umgehen. Ein beliebter und erfolgreicher Ansatz dazu ist, bewusst die Regeln zu verletzen[6]. Deshalb muss man durchaus damit rechnen, dass jemand — etwa in kryptographischen Protokollen wie dem nach Diffie und Hellman, Abschnitt 1.1 — zusammengesetzte Zahlen an Stellen ins Spiel bringt, wo laut Vorschrift stets Primzahlen verwendet werden sollen. Um einen solchen Betrug aufzudecken, bevor dadurch größerer Schaden angerichtet werden kann, ist es ratsam, die entsprechenden Zahlen daraufhin zu untersuchen, ob sie prim sind oder nicht. Das erledigen Primzahltests, die im nächsten Kapitel vorgestellt werden.

[6] Beispielsweise um mit einem Pufferüberlauf das System zu korrumpieren.

2. Primzahlerzeugung und Primzahltests

Nachdem wir gesehen haben, dass große Primzahlen praktisch bedeutsam sind und bereits genutzt werden, stellt sich die Frage, wie man jeweils eine große Primzahl finden kann, wenn man sie benötigt. Eine Antwort darauf wird in dem nun folgenden Abschnitt 2.1 gegeben. Den Primzahltests, die dabei eine wichtige Rolle spielen, widmet sich Abschnitt 2.2 genauer. Dort werden zunächst der Begriff „Primzahltest“ und einige damit zusammenhängende Bezeichnungen definiert. Danach werden Beispiele für Primzahltests genannt, so dass auch der quadratische Frobenius-Test in den Blick kommt. Wir werden sehen, was ihn so interessant macht, dass er im Mittelpunkt dieser Arbeit steht.

2.1. Primzahlerzeugung

Verblüffend einfach ist ein Verfahren, mit dem die in der Kryptographie genutzten Primzahlen generiert werden: Es wird so lange zufällig eine große Zahl gewählt, bis die gewählte Zahl eine Primzahl ist. Dieser Ansatz ist erfolgreich, weil zwei Voraussetzungen gegeben sind: Erstens gibt es genug große Primzahlen, so dass man tatsächlich nach einer nicht zu großen Anzahl von Versuchen eine Primzahl wählt. Außerdem kann man relativ schnell testen, ob eine große Zahl eine Primzahl ist. Beide Aussagen sollen im Folgenden etwas präzisiert werden.

Bitlänge	128	1024	2048	4096	8192	65536
Versuchszahl	45	355	710	1420	2840	22713

Tabelle 2.1.: Nach wieviel Versuchen kann man erwarten eine Primzahl zu erhalten, wenn man jedesmal eine ungerade Zufallszahl der entsprechende Länge wählt?

2.1.1. Wie viele große Primzahlen gibt es?

Diese Frage lässt sich beantworten, wenn man das Wachstum der Primzahlzählfunktion

$$\pi(x) := \#\{p \text{ Primzahl} \mid p \leq x\}$$

gut kennt. Die Primzahlen sind zwar recht unregelmäßig verteilt, aber es gibt den berühmten Primzahlsatz, der besagt, dass

$$\lim_{x\to\infty} \frac{\pi(x)}{x/\ln x} = 1.$$

Daher kann man für große n $\pi(n)$ durch $n/\ln n$ annähern. Die Wahrscheinlichkeit, dass zufälliges Wählen einer Zahl aus $\{1, 2, 3, 4, \ldots, n\}$ gerade eine Primzahl trifft, ist also in etwa

$$\frac{n/\ln n}{n} = \frac{1}{\ln n}$$

ist. Das ist sehr schön, denn damit ist die erwartete Anzahl der Versuche, die man benötigt, um eine Primzahl zu wählen, proportional zur Bitlänge der gewünschten Primzahl. Praktischerweise beschränkt man sich dann noch auf ungerade Zahlen, so dass sich die erwartete Versuchszahl halbiert. Für eine 2048-Bit-Primzahl benötigt man also etwa $\ln(2^{2048})/2 \approx 710$ Versuche. Weitere Beispiele stehen in Tabelle 2.1.

2.1.2. Wie werden Primzahlen erkannt?

Kleine Teiler finden

Wenn eine große Zahl n vorgelegt ist, von der entschieden werden soll, ob sie eine Primzahl ist, prüft man zunächst eine einfache Bedingung, die in den meisten Fällen schon zu einem Ergebnis führt. Man testet nämlich, ob n kleine Teiler hat. Dazu bestimmt man alle Primzahlen $2, 3, 5, 7, 11, 13, \ldots, p_l$, die kleiner als eine Schranke B sind und teilt n durch diese. Geht eine dieser Divisionen ohne Rest auf, ist man fertig und weiß,

B	10	100	1000	10000	50000
$\varphi(m)/m$	0,2286	0,1203	0,0810	0,0609	0,0519
$\pi(B) = l$	4	25	168	1229	5133
$B/\ln B$	4,3	21,7	144,8	1085,7	4621,2
p_l	7	97	997	9973	49999

Tabelle 2.2.: Welcher Anteil an Kandidaten ist nach dem probeweisen Dividieren durch kleine Zahlen noch weiter zu untersuchen?

dass n keine Primzahl ist. Um die Primzahlen bis B zu finden, benutzt man Siebverfahren. Mit dem Sieb des Erathostenes beispielsweise kann man auf einem PC in wenigen Sekunden alle Primzahlen bis 10^7 finden. Für größere B kann man Siebverfahren nicht nutzen, weil sie dann zu viel Speicherplatz und auch zu viel Zeit verbrauchen[1].

Wir wollen kurz überlegen, was es nützt, zunächst nach kleinen Teilern zu suchen. Seien dazu die Primzahlen bis B mit $p_1 < p_2 < \ldots < p_l$ bezeichnet und $m := p_1 p_2 \ldots p_l$. Nach der versuchsweisen Division bleiben nur die Zahlen übrig, die durch keine der Primzahlen $p_1, p_2, \ldots, p_l$ teilbar, also teilerfremd zu m sind. Das ist ein Anteil von $\varphi(m)/m = \prod_{i=1}^{l} \frac{p_i - 1}{p_i}$. In Tabelle 2.2 ist dieser für einige B ausgerechnet. Wenn wir also beispielsweise mittels 25 Divisionen aller Teiler ≤ 100 erkennen, können wir in 88% der Fälle die gestellte Frage schon beantworten, nämlich mit „nein, n ist keine Primzahl“.

Wollte man allein mit versuchsweiser Division erkennen, ob n prim ist oder nicht, so müsste man bis $\lfloor \sqrt{n} \rfloor$ nach Teilern von n suchen. Das reicht, weil für jede Zerlegung $n = t \cdot t'$ notwendigerweise $t \leq \sqrt{n}$ oder $t' \leq \sqrt{n}$ gilt. Andererseits finden wir beispielsweise tatsächlich erst bei $\sqrt{n}$ den nichttrivialen Teiler von n, wenn n das Quadrat einer Primzahl ist. Wir müssen n also mindestens durch alle Primzahlen p teilen, für die $p \leq \sqrt{n}$. Abgesehen davon, dass nicht klar ist, wie man schnell erfahren kann, welche Zahlen konkret das sind, handelt es sich ungefähr um $\sqrt{n}/\ln(\sqrt{n}) = 2\sqrt{n}/\ln n$ Primzahlen und damit Divisionen. Bezogen auf die Länge $\log n$ der Eingabe sind das exponentiell viele[2] — zu viele für ein zeiteffizientes Verfahren, wenn n groß ist. Man kommt also nicht umhin, zu anderen Mitteln zu greifen.

[1] Das Sieb des Eratosthenes benötigt Platz für $O(B)$ Bits und $B \log\log B + O(B)$ arithmetische Operationen, um alle Primzahlen bis B zu finden, vgl. [CP01], Abschnitt 3.2.

[2] Exakt: $O(2^{\log n/2}/\log n)$, also weniger als $O(2^{\log n})$, aber mehr als jedes Polynom in $\log n$.

Anwendung eines Primzahltests

An dieser Stelle kommen Primzahltests, wie sie in dieser Arbeit behandelt werden, ins Spiel. Wir wollen jetzt am Beispiel des Miller-Rabin-Primzahltests (Algorithmus 6, Seite 81), der sehr bekannt ist und häufig verwendet wird, sehen, was Primzahltests leisten und wie sie helfen, große Primzahlen zu finden.

Eine Zahl n, die keine kleinen Teiler hat, wird dem Miller-Rabin-Test unterworfen. Falls der Test erklärt, dass n keine Primzahl ist, wird das Problem als entschieden betrachte und davon ausgegangen, dass n zusammengesetzt ist. Ansonsten tätigt der Test die Ausgabe „n ist prim“. Diese Ausgabe ist nicht hundertprozentig vertrauenswürdig. Deshalb wird in diesem Fall der Test erneut aufgerufen. Diesmal wird höchstwahrscheinlich in Schritt 1 von Algorithmus 6 ein anderes a gewählt. So kann es sein, dass der Test dieses Mal zu dem Ergebnis kommt, dass n keine Primzahl ist, was zur Folge hätte, dass die Berechnungen beendet und n als zusammengesetzt betrachtet werden kann. Wenn das nicht der Fall ist, wird ein drittes, viertes, fünftes, ..., k-tes Mal nach dem gleichen Prinzip getestet. Falls dabei jede Iteration des Tests die Ausgabe „n ist prim“ geliefert hat, wird n als Primzahl betrachtet.

Ist dieses Vorgehensweise korrekt? Wir werden in Kapitel 5 sehen, dass die Antwort von Algorithmus 6 richtig ist, wenn er n für zusammengesetzt erklärt. Außerdem kann man beweisen, dass die Wahrscheinlichkeit, dass eine zusammengesetzte Zahl n als Primzahl klassifiziert wird, höchstens $1/4$ ist. Die Wahrscheinlichkeit, dass dieses Ereignis in jeder von k stochastisch unabhängigen Iterationen eintritt, ist nicht größer als $1/4^k$. Mit zunehmendem k wird dieser Wert schnell deutlich kleiner als die Wahrscheinlichkeit, dass die Berechnung aufgrund von Hardwareversagen zu einem falschen Ergebnis kommt. Die so gewonnene Sicherheit reicht für die Praxis aus, um n als Primzahl zu betrachten.

In Kapitel 5 werden wir auch sehen, dass eine Iteration des Miller-Rabin-Test nicht sehr lange dauert, nämlich nur $O((\log n)^3)$ oder noch weniger Bitoperationen, je nachdem wie geschickt man ihn implementiert. Damit stellt der Miller-Rabin-Primzahltests ein geeignetes Mittel dar, um auch für eine große Zahl n zu entscheiden, ob sie eine Primzahl ist oder nicht.

2.2. Primzahltests

Neben dem Miller-Rabin-Test gibt es noch andere Primzahltests. Bevor einige genannt werden, sollen der Begriff „Primzahltest“ und mehrere andere Sprechweisen definiert und das Prinzip, nach dem viele Primzahltests vorgehen, abstrakt dargestellt werden.

2.2.1. Definitionen und ein Prinzip

Ein Primzahltest ist ein Verfahren, dass entscheidet, ob eine gegebene Zahl n eine Primzahl ist oder nicht, indem es „n ist prim“ oder „n ist nicht prim“ ausgibt. Wenn man als „Verfahren“ nur Algorithmen anerkennt, die stets die richtige Antwort geben, dann ist ein Primzahltest P genau dann korrekt, wenn gilt

$$n \text{ ist Primzahl} \quad \Leftrightarrow \quad P \text{ gibt „}n \text{ ist prim“ aus.} \tag{2.1}$$

Man sagt dann auch: P *beweist*, dass die Eingabe n eine Primzahl ist oder dass n eine zusammengesetzte Zahl ist.

Es gibt Algorithmen, die in diesem strengen Sinn Primzahltests sind. Berühmt ist das seit 2002 bekannte Verfahren nach Agrawal, Kayal und Saxena, das deterministisch in polynomieller Zeit Primzahlen von zusammengesetzten Zahlen unterscheiden kann [AKS02]. Allen bekannten Primzahltests im Sinne von (2.1) ist jedoch gemeinsam, dass sie für große Eingaben in der Praxis zu langsam sind. Es würde einfach zu lange dauern, sie beispielsweise im Rahmen der in Abschnitt 2.1.2 entwickelten Vorgehensweise anwenden und so große Primzahlen finden zu wollen.

Aus diesem Grund akzeptiert man als Primzahltests auch eine bestimmte Art von randomisierten Algorithmen, die sich „irren“ können, nämlich Monte-Carlo-Algorithmen mit einseitigem beschränkten Fehler. Ein Primzahltest in diesem Sinne ist jedes Verfahren P, für das es eine reelle Zahl $\varepsilon, 0 < \varepsilon < 1$, gibt, so dass P für jede Eingabe n folgendes Ausgabeverhalten zeigt:

$$n \text{ ist Primzahl} \quad \Rightarrow \quad P \text{ gibt „}n \text{ ist prim“ aus,} \tag{2.2}$$

$$n \text{ ist zusammengesetzt} \quad \Rightarrow \quad \text{Prob}(P \text{ gibt „}n \text{ ist prim“ aus}) \leq \varepsilon \tag{2.3}$$

Wenn P die Antwort „n ist nicht prim“ ausgibt, dann kann die Eingabe n wegen (2.2) keine Primzahl sein. In diesem Fall ist die Ausgabe also richtig und man kann davon reden, dass das Verfahren P bewiesen hat, dass n zusammengesetzt ist. Die Antwort

„n ist prim“ dagegen ist kein Beweis dafür, dass n eine Primzahl ist, sondern kann auch falsch sein. Deshalb sagt man, dass P ein Algorithmus mit „einseitigem Fehler“ ist. Es ist wichtig, dass die Wahrscheinlichkeit dafür, dass dieser Fehler eintritt — die *Irrtumswahrscheinlichkeit* —, unabhängig von der Eingabe n durch $\varepsilon < 1$ beschränkt ist. Daher redet man von einem Verfahren mit „einseitigem beschränkten Fehler“. Für den Rest dieser Arbeit sei festgelegt, dass unter einem Primzahltest ein Verfahren in dem gerade beschriebenen Sinne zu verstehen ist:

Definition 2.1. *Ein* Primzahltest *ist Monte-Carlo-Algorithmus mit einseitigem beschränkten Fehler, der entscheidet, ob die Eingabe prim ist oder nicht.*

Viele Primzahltests sind nach einem einfachen Prinzip aufgebaut. Sie benutzen eine mathematische Tatsache der Form

$$n \in \mathbb{N} \text{ ist Primzahl} \Rightarrow \text{für jedes } a \in A(n) \text{ gilt } E(a, n). \tag{2.4}$$

Dabei ist $A(n)$ eine von n abhängige Menge, deren konkrete Gestalt ganz unterschiedlich sein kann, je nachdem, um welche mathematische Tatsache es sich handelt. E ist eine algorithmisch leicht prüfbare Eigenschaft, die von a und n abhängt. Wenn man n gegeben hat und ein a findet, für das $E(a, n)$ nicht gilt, dann weiß man, dass n keine Primzahl ist. Ein solches a wird *E-Zeuge* für n genannt, denn a „bezeugt“ — wenn man (2.4) zugrundelegt — dass n zusammengesetzt ist. Ein einfacher Ansatz, einen Zeugen für n zu finden, ist zu raten. Das ist das Prinzip, nach dem viele Primzahltests vorgehen. Das allgemeine Schema dieser Vorgehensweise ist in Algorithmus 1 festgehalten.

Wegen (2.4) klassifiziert jedes Verfahren P, dass dem Schema von Algorithmus 1 folgt, Primzahlen korrekt als prim. Bedingung (2.2) wird also eingehalten. Damit auch (2.3) erfüllt ist, so dass P ein Primzahltest im Sinne unserer Definition ist, muss es ein $\varepsilon > 0$ geben, so dass für jedes zusammengesetzte n der Anteil der Zeugen in A mindestens ε ist.

Unabhängig davon, ob Algorithmus 1 in Verbindung mit einer mathematischen Tatsache der Form (2.4) zu einem Primzahltest führt oder nicht, wird die folgende Sprechweise eingeführt.

Definition 2.2. *Sei eine mathematische Tatsache der Form (2.4) gegeben und n eine natürliche Zahl. Dann heißt n E-*Primzahlkandidat[3] *bezüglich a, falls $E(a, n)$ erfüllt*

[3] Englisch: probable prime

Algorithmus 1 Schema für Primzahltests.

Require: Eine mathematische Tatsache der Form (2.4) gilt. Die Eingabe für den Algorithmus ist n.

1: Wähle $\mathtt{a} \in A(n)$ zufällig.
2: **if** $E(\mathtt{a}, n)$ gilt **then**
3: **return** „n ist prim.“
4: **else**
5: **return** „n ist nicht prim.“
6: **end if**

*ist. Falls n zusammengesetzt ist und $E(a, n)$ erfüllt ist, heißt n E-*Pseudoprimzahl[4] *bezüglich a.*

Ein E-Primzahlkandidat bezüglich eines $a \in A(n)$ ist also entweder tatsächlich eine Primzahl oder aber eine E-Pseudoprimzahl bezüglich a.

Im folgenden Abschnitt werden wichtige Beispiele für Primzahltests aufgezählt. Beispiele für die anderen gerade definierten Begriffe folgen in den Kapiteln 5 und 6.

2.2.2. Klassische Beispiele und der quadratische Frobenius-Test

Wie man den Erläuterungen in Abschnitt 2.1.2 entnehmen kann, ist der Miller-Rabin-Test ein Primzahltest im Sinne von Definition 2.1 mit Irrtumswahrscheinlichkeit $\leq 1/4$. Außer dem Miller-Rabin-Test gibt es auch noch andere Primzahltests. Der Solovay-Strassen-Test [SS71] (siehe auch beispielsweise [Die04, Kapitel 6]) etwa benötigt ähnlich viel Zeit wie der Miller-Rabin-Test und hat eine Irrtumswahrscheinlichkeit von maximal $1/2$. Außerdem sind alle seine Zeugen jeweils auch Zeugen für den Miller-Rabin-Test. Deshalb spielt der Solovay-Strassen-Test in der Praxis keine große Rolle. Die Irrtumswahrscheinlichkeit des starken Lucas-Tests ist durch $4/15$ beschränkt [Arn97]. Jedoch ist dieser Test nicht schneller als der Miller-Rabin-Test. Deshalb stellt er bei der Primzahlerzeugung keine Verbesserung gegenüber dem Miller-Rabin-Test dar.

Anders verhält es sich mit dem quadratischem Frobenius-Test. Es gibt verschiedene Varianten davon. Die erste wurde 1998 von Grantham vorgeschlagen [Gra98]. Müller [Mül01, Mül03] sowie Damgård und Frandsen [DF03] entwickelten Verbesserungen. Die Variante aus [Gra98] hat eine Irrtumswahrscheinlichkeit von höchstens $1/7710$,

[4] Englisch: pseudoprime

während sie asymptotisch die Zeit von drei Miller-Rabin-Tests benötigt. Damit sollte der Frobenius-Test dem Miller-Rabin-Test überlegen sein, denn um eine Zahl bis auf eine Irrtumswahrscheinlichkeit $0 < \alpha < 1$ als Primzahl zu erkennen, benötigt der Miller-Rabin Test $\left\lceil \frac{-\log \alpha}{2} \right\rceil$ Iterationen, während der Frobenius-Test eine Zeit benötigt, die $3\left\lceil \frac{-\log \alpha}{\log 7710} \right\rceil$ Iterationen des Miller-Rabin-Tests entsprechen sollte. Da $\frac{3}{\log 7710} \approx 0.2323$ ist, kann man damit rechnen, dass der Frobenius-Test etwa $0.5/0.2323 \approx 2.15$ mal so schnell ist wie der Miller-Rabin-Test.

Aus diesem Grund ist der quadratische Frobenius-Test besonders interessant und wird in der vorliegenden Arbeit eingehend untersucht. Die Darstellung beschränkt sich dabei auf den Test in der Variante aus [Gra98]. Die exakte Bezeichnung des Algorithmus, der ein Primzahltest im Sinne unserer Definition ist, lautet „randomisierter starker quadratischer Frobenius-Test“ (RQFT). Wir wollen sehen, auf welchen mathematischen Zusammenhängen er beruht, warum die Irrtumswahrscheinlichkeit kleiner als 1/7710 ist und wie man den Test zeiteffizient implementieren kann. Außerdem wollen wir anhand von Experimenten nachprüfen, ob der RQFT auch in der Praxis doppelt so schnell ist wie der Miller-Rabin-Test. Dazu werden wir die Laufzeit des RQFT mit der dreifachen Laufzeit des Miller-Rabin-Tests vergleichen.

3. Mathematische Hilfsmittel

Der quadratische Frobenius-Test ist nur zu verstehen, wenn man gewisse mathematische Begriffe und Zusammenhänge kennt. Deshalb kommen wir nicht darum herum, diese systematisch zusammenzustellen. Dazu soll dieses Kapitel dienen. Zuerst werden elementare Definitionen und Tatsachen aufgezählt, so dass in den Abschnitten 3.5 bis 3.8 die Aussagen herausgearbeitet werden können, mit denen die Irrtumswahrscheinlichkeit des RQFT abgeschätzt werden kann und die eine zeiteffiziente Implementierung des RQFT möglich machen. Ausführlicher als es hier dargestellt werden kann, finden sich die einzelnen Themen beispielsweise in [Wol96], [Bos04] und [IR90]. Abschnitt A.1 im Anhang enthält mathematische Definitionen und Vereinbarungen, die nicht in dieses Kapitel passen.

Fakt 3.1 (Division mit Rest). *Zu jedem „Dividenden“ $a \in \mathbb{Z}$ und jedem „Divisor“ $b \in \mathbb{Z}$, $b > 0$, gibt es genau einen „Quotienten“ $q \in \mathbb{Z}$ und genau einen „Rest“ $r \in \mathbb{Z}$, so dass $a = qb + r$ und $0 \leq r < b$ erfüllt ist.*

Für q schreibt man auch „$a \operatorname{div} b$“, für r auch „$a \bmod b$“. Man sagt „a geteilt durch b ist q, Rest r“. Manchmal kommt es auf den Quotienten q nicht an und man sagt einfach „a lässt bei Division durch b den Rest r“. Zwei Beispiele: 17 geteilt durch 5 ist 3, Rest 2, weil $17 = 5 \cdot 3 + 2$. 79 lässt bei Division durch 11 den Rest 2, denn $79 = 11 \cdot 7 + 2$.

3.1. Gruppen

Definition 3.2. *Eine* Gruppe $(G, \circ)$ *ist eine nichtleere Menge G zusammen mit einer Verknüpfung $\circ : G \times G \to G$, für die gilt:*

1. $\circ$ *ist* assoziativ*: für alle $a, b, c \in G$ gilt $(a \circ b) \circ c = a \circ (b \circ c)$.*

2. *Es gibt ein* neutrales Element e*, für das gilt: für alle* $a \in G$ *ist* $a \circ e = e \circ a = a$.

3. *Existenz* inverser *Elemente: Für jedes* $a \in G$ *gibt es ein* $a^{-1} \in G$*, so dass* $a \circ a^{-1} = a^{-1} \circ a = e$.

Falls die Verknüpfung zusätzlich noch kommutativ ist, also für $a, b \in G$ gilt $a \circ b = b \circ a$, nennt man die Gruppe *abelsch*. Für abelsche Gruppen wird meistens $+$ als Bezeichnung für die Gruppenoperation, 0 für das neutrale und $-a$ für das zu a inverse Element benutzt.

Sei $(G, \circ)$ eine Gruppe, $a \in G$ und $k \geq 1$ eine ganze Zahl. Unter a^k verstehen wir $\underbrace{a \circ a \circ \ldots \circ a}_{k \text{ Faktoren „}a\text{“}}$. Weiter setzen wir $a^{-k} := (a^{-1})^k$ und $a^0 := e$. Das Zeichen „$\circ$“ wird oft ausgelassen. Eine *Untergruppe* U von G ist eine Teilmenge $U \subset G$ die selbst eine Gruppe ist mit $\circ$ als Verknüpfung.

Ein Beispiel für eine abelsche Gruppe ist $\mathbb{Z}$, die Menge der ganzen Zahlen mit der üblichen Addition. Mit $n\mathbb{Z}$, $n \in \mathbb{Z}$ wollen wir die Vielfachen von n bezeichnen, also die Menge $\{nz \mid z \in \mathbb{Z}\}$. Man sieht, dass $n\mathbb{Z}$ eine Untergruppe von $\mathbb{Z}$ ist. Es gilt

Satz 3.3. *$\mathbb{Z}$ hat nur die Mengen $n\mathbb{Z}$, $n \in \mathbb{Z}$, als Untergruppen.*

Beweis. [Bos04], Seite 21, Lemma 1.3/4. Wir bemerken, dass für den Beweis Division mit Rest genutzt wird. □

Ist eine Untergruppe $U \subset G$ gegeben so kann man für jedes $a \in G$ die *Linksnebenklasse* $aU := \{au \mid u \in U\}$ definieren. Diese Nebenklassen sind gerade die Mengen, in die G mittels der Äquivalenzrelation

$$x \equiv y :\Leftrightarrow y^{-1}x \in U \tag{3.1}$$

partitioniert wird. Zwei Linksnebenklassen sind immer gleichmächtig und entweder identisch oder disjunkt. Die Menge aller Linksnebenklassen wird mit G/U bezeichnet. Analog kann man auch Rechtsnebenklassen betrachten: $Ua := \{ua \mid u \in U\}$. Die Anzahl aller Links- und Rechtsnebenklassen ist gleich und wird mit $G : U$ bezeichnet. Die Mächtigkeit der Gruppe G selbst wird mit *Ordnung* von G, $\operatorname{ord} G$, bezeichnet. Eine Gruppe G heißt *endlich*, wenn G nur endlich viele Elemente hat.

Satz 3.4 (Satz von Lagrange). *Sei G eine endliche Gruppe und H eine Untergruppe von G. Dann gilt*

$$\operatorname{ord} G = \operatorname{ord} H \cdot (G : H).$$

Beweis. [Bos04], Seite 17, Korollar 1.2/3. □

Normalteiler, Restklassen und Gruppenhomomorphismen

Eine Untergruppe $N \subset G$ heißt *Normalteiler* von G, falls für jedes $a \in G$ die Rechts- und Linksnebenklasse identisch sind, also $aN = Na$ gilt. In abelschen Gruppen ist jede Untergruppe ein Normalteiler. Die Nebenklassen bezüglich eines Normalteilers N nennt man auch Restklassen modulo N. Drei unterschiedlich Schreibweisen für dieselbe Restklasse sind $aN, a \circ N$ und $a \bmod N$. Definiert man das Produkt von zwei Teilmengen $X, Y \subset G$ durch

$$X \circ Y := \{x \circ y \mid x \in X, y \in Y\}, \tag{3.2}$$

so ist das Produkt von Restklassen wieder eine Restklasse und die Menge G/N aller Restklassen bildet eine Gruppe mit dem gerade definierten Produkt als Operation. Dabei ist $eN = N$ das neutrales Element und zu aN ist $a^{-1}N$ invers.

Eine Abbildung $\varphi : G \to H$ von einer Gruppe G in eine Gruppe H heißt *Gruppenhomomorphismus*, falls für alle $a, b \in G$ gilt: $\varphi(ab) = \varphi(a)\varphi(b)$. Ist φ außerdem noch bijektiv, so nennt man φ einen *Isomorphismus* und G und H isomorph, wofür man $G \simeq H$ schreibt. Der *Kern* $\ker \varphi$ eines Homomorphismus φ ist die Menge $\{g \in G \mid \varphi(g) = e\}$. Er ist stets ein Normalteiler von G und enthält immer das neutrale Element. Sehr wichtig ist der

Satz 3.5 (Homomorphiesatz für Gruppen). *Sei $\varphi : G \to G'$ ein surjektiver Homomorphismus mit Kern N. Dann ist $G/N \simeq G'$ mittels $gN \mapsto \varphi(g)$.*

Beweis. [Bos04], Seite 18, Satz 1.2/6. □

3.2. Ringe

Viele Strukturen verfügen nicht nur über eine Operation, sondern über zwei, die zueinander in Beziehung stehen. Das führt zum Begriff des Ringes.

Definition 3.6. *Ein* Ring $(R, +, \cdot)$ *ist eine abelsche Gruppe $(R, +)$, zusammen mit einer Verknüpfung $\cdot : R \times R \to R$, für die folgende Bedingungen gelten, zusätzlich zu den Anforderungen, die an die abelsche Gruppe $(R, +)$ gestellt werden:*

1. *$\cdot$ ist assoziativ: für alle $a, b, c \in R$ gilt $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.*
2. *Für $+$ und $\cdot$ gelten Distributivgesetze: für alle $a, b, c \in R$ ist $a \cdot (b + c) = a \cdot b + a \cdot c$ und $(a + b) \cdot c = a \cdot c + b \cdot c$.*

$+$ wird Addition und $\cdot$ Multiplikation genannt. Der Ring heißt Ring mit Eins*, falls es ein bezüglich der Multiplikation neutrales Element $1 \in R$ gibt, so dass für alle $r \in R$ gilt $1 \cdot r = r \cdot 1 = r$. Weiter heißt der Ring* kommutativ*, wenn $\cdot$ kommutativ ist.*

Wie beim Rechnen mit Zahlen üblich, wird „$\cdot$“ oft ausgelassen. Außerdem ist vereinbart, dass die Multiplikation stärker bindet als die Addition. Somit ist beispielsweise $a + bc$ zu verstehen als $a + (b \cdot c)$.

Sei R ein Ring mit Eins. $R^* := \{a \in R \mid \text{es gibt } b \text{ mit } ab = ba = 1\}$ wird die Menge der *Einheit*en von R genannt. $(R^*, \cdot)$ ist eine Gruppe. 0 kann niemals zu R^* gehören, weil $0 \cdot r = 0$ für alle $r \in R$. Wenn jedoch $R^* = R - \{0\}$ ist, nennt man R einen *Schiefkörper*. Falls zusätzlich noch $\cdot$ kommutativ ist, nennt man R einen *Körper*. $r \in R$ heißt *Nullteiler*, falls ein $r' \in R, r' \neq 0$, existiert mit $rr' = 0$. In Körpern ist 0 der einzige Nullteiler. Ein Ring $R \neq \{0\}$, der — abgesehen von 0 — keine Nullteiler hat, heißt *nullteilerfrei* oder *Integritätsring*. Zwei Elemente $r_1, r_2 \in R$ eines Integritätsringes heißen *assoziiert*, falls es eine Einheit $\varepsilon \in R^*$ gibt mit $r_1 = \varepsilon r_2$.

$U \subset R$ heißt *Unterring* von R, falls U selbst ein Ring ist mit den Operationen von R. Ist R_1 ein Unterring von R_2, so wird R_2 auch *Oberring* von R_1 genannt. Ganz entsprechend sind die Bezeichnungen Unter- bzw. Oberkörper zu verstehen.

Das klassische Beispiel für einen Ring ist $\mathbb{Z}$, die Menge der ganzen Zahlen mit der üblichen Addition und der üblichen Multiplikation. $\mathbb{Z}$ hat eine 1 und keine Nullteiler. Ein Ring ohne 1 ist beispielsweise $2\mathbb{Z}$, die Menge der geraden Zahlen. Beide Ringe sind kommutativ. Es gibt auch nicht kommutative Ringe, zum Beispiel im Bereich der Matrizen.

Definition 3.7. *Ab jetzt werden in dieser Arbeit Ringe stets als kommutative Ringe mit 1 vorausgesetzt.*

Satz 3.8. *Sei K ein Körper. Dann sind 1 und -1 die einzigen Quadratwurzeln von 1.*

Beweis. Sei $r \in K$ eine Quadratwurzel von 1, das heißt es gelte $r^2 = 1$. Das ist äquivalent zu $(r-1)(r+1) = 0$. Da K nullteilerfrei ist, folgt $r = 1$ oder $r = -1$. □

3.2.1. Ideale, Ringhomomorphismen und der Chinesische Restsatz

Die in diesem Abschnitt behandelten Themen finden sich ausführlicher in [Bos04], Abschnitt 2.2 und 2.3, auf den Seiten 34 bis 39 sowie 42.

Ideale spielen bei Ringen eine ähnliche Rolle wie Normalteiler bei Gruppen. Eine Untergruppe I der abelschen Gruppe $(R,+)$ heißt *Ideal* von R, falls für jedes $r \in R$ und jedes $i \in I$ gilt: $ri \in I$. Man beachte, dass ein Ideal I ein Normalteiler von $(R,+)$ ist, weil Letzteres eine abelsche Gruppe ist. Die Summe $I_1 + I_2 = \{x + y \mid x \in I_1, y \in I_2\}$ zweier Ideale $I_1, I_2 \subset R$ ist wieder ein Ideal, ebenso das Produkt $I_1 I_2 := \{\sum_{1 \le j \le k} x_j y_j \mid k \in \mathbb{N}, x_j \in I_1, y_j \in I_2 \text{ für } 1 \le j \le k\}$ und der Durchschnitt $I_1 \cap I_2$. Es gilt stets $I_1 I_2 \subset I_1 \cap I_2$.

Für $a_1, \ldots, a_k \in R$ ist

$$(a_1, a_2, \ldots, a_k) := \{r_1 a_1 + \ldots + r_k a_k \mid r_1, \ldots, r_k \in R\}$$

ein Ideal, das *von $a_1, \ldots, a_k$ erzeugte Ideal*. Es ist das kleinste Ideal, das $a_1, \ldots, a_k$ enthält. Besonders wichtig sind die sogenannten *Hauptideale* $(a) = \{ra \mid r \in R\}$, die jeweils von einem einzelnen Element $a \in R$ erzeugt werden.

Offensichtlich sind die Mengen $n\mathbb{Z} = (n)$, $n \in \mathbb{Z}$, Hauptideale in $\mathbb{Z}$. Wegen Satz 3.3 gibt es dort auch keine anderen Ideale, so dass gilt:

Satz 3.9. *Die Ideale in $\mathbb{Z}$ sind die Mengen der Form $n\mathbb{Z}$, wobei $n \in \mathbb{Z}$.*

Integritätsringe, in denen — wie in $\mathbb{Z}$ — jedes Ideal ein Hauptideal ist, werden *Hauptidealringe* genannt. Die erzeugenden Elemente von Hauptidealen sind in Integritätsringen bis auf Assoziiertheit eindeutig bestimmt.

Mit Normalteilern ließ sich die Gruppenstruktur von G auf G/N übertragen. Entsprechendes gilt auch für Ideale und Ringe. Sei dazu ein Ideal $I \subset R$ gegeben. Aus Abschnitt 3.1 ist bekannt, dass $(R/I,+)$ eine abelsche Gruppe ist mit $(r_1+I)+(r_2+I) = (r_1+r_2)+I$. Definiert man nun noch $(r_1+I)\cdot(r_2+I) := (r_1 \cdot r_2)+I$, so ist auch das Produkt von Restklassen wieder eine Restklasse, unabhängig von den Repräsentanten, so dass R/I mit diesen Verknüpfungen ein Ring ist. Nullelement von R/I ist $0+I$, Einselement ist $1+I$.

Sind R_1, R_2 zwei Ringe, so nennt man eine Abbildung $\varphi : R_1 \to R_2$ *Ringhomomorphismus*, falls φ ein Gruppenhomomorphismus ist und zusätzlich für alle $r, s \in R_1$ gilt: $\varphi(rs) = \varphi(r)\varphi(s)$ sowie $\varphi(1) = 1$. Stets ist $\operatorname{im}\varphi$, das Bild $\{\varphi(r) \mid r \in R_1\} \subset R_2$ von φ, ein Unterring von R_2. Der Kern eines Ringhomomorphismus ist sogar ein Ideal. Ausgehend von 3.5 beweist man

Satz 3.10 (Homomorphiesatz für Ringe). *Seien R, R' Ringe und $\varphi : R \to R'$ ein surjektiver Ringhomomorphismus mit Kern I. Dann ist R/I isomorph zu R' und zwar mittels $r + I \mapsto \varphi(r)$.*

Beweis. [Bos04], Seite 39, Satz 2.3/4, Korollar 2.3/5. □

Sind $R_1, \ldots, R_k$ Ringe, so ist $R = R_1 \times \ldots \times R_k$ ebenfalls ein Ring, das *direkte Produkt* der Ringe $R_1, \ldots, R_k$ (vgl. auch [Bos04], Seite 29, Beispiel (4)). Die Operationen sind dabei wie folgt definiert:

$$(r_1, \ldots, r_k) \circ (r'_1, \ldots, r'_k) := (r_1 \circ r'_1, \ldots, r_k \circ r'_k) \text{ für } \circ \in \{+, \cdot\}$$

Satz 3.11 (Chinesischer Restsatz). *Sei R ein Ring und $I_1, \ldots, I_n$ Ideale, für die $I_i + I_j = R$ für $i \neq j$ gilt. Es sei außerdem $I = I_1 \cap \ldots \cap I_n$. Dann besteht der Ringisomorphismus $R/I \simeq R/I_1 \times \ldots \times R/I_n$, $r + I \mapsto (r + I_1, \ldots, r + I_n)$. Dieser Isomorphismus zeigt auch, dass gilt: $(R/I)^* \simeq (R/I_1)^* \times \ldots \times (R/I_n)^*$.*

Beweis. [Bos04], Seite 42, Satz 2.3/12. Für die Aussage über die Gruppe der Einheiten siehe [IR90], Seite 35, Proposition 3.4.1. □

3.2.2. Polynomringe

Definition 3.12. *Sei R ein Ring. Die Menge*

$$R[X] := \{(a_0, a_1, a_2, \ldots) \mid (a_i)_{i \geq 0} \text{ ist Folge in } R \text{ und } a_i \neq 0 \text{ nur für endlich viele } i\}$$

der Polynome *in einer Variablen über R bildet einen Ring, wobei die Addition wie üblich komponentenweise und die Multiplikation durch Faltung definiert ist:*

$$\begin{aligned} (a_i)_{i \geq 0} + (b_i)_{i \geq 0} &:= (a_i + b_i)_{i \geq 0}, \\ (a_i)_{i \geq 0} \cdot (b_i)_{i \geq 0} &:= \left(\sum_{j=0}^{i} a_j b_{i-j} \right)_{i \geq 0}. \end{aligned}$$

Jedes Element $r \in R$ wird mit $(r, 0, 0, \ldots) \in R[X]$ identifiziert, so dass R ein Unterring von $R[X]$ ist. $0 = (0, 0, 0, \ldots)$ ist das Nullelement von $R[X]$ und wird auch als *Nullpolynom* bezeichnet. Das Element $(0, 1, 0, 0, \ldots) \in R[X]$ wird mit X abgekürzt. Sei $f = (a_0, a_1, a_2, \ldots) \in R[X]$. Dann heißt a_i *Koeffizient* vom Grad i von f. Der *Grad* des Polynoms f selbst ist definiert als

$$\deg(f) = \max\{i \mid a_i \neq 0\},$$

das Nullpolynom hat per Definition den Grad $-\infty$. Ist $f = (a_0, a_1, a_2, \ldots) \in R[X], f \neq 0$, so heißt $a_{\deg(f)}$ *führender Koeffizient* von f. f heißt *normiert*, wenn der führende Koeffizient 1 ist.

Definition 3.12 ist die mathematisch exakte Modellierung der Vorstellung, dass Polynome „Terme der Art“ $X^3 - X^2 + 5$ oder $X^7 - 19$ sein sollten, die man auf die naheliegende Weise addieren, subtrahieren und multiplizieren kann. Man kann sich nämlich überlegen, dass gilt:

Satz 3.13. *Sei R ein Ring. In $R[X]$ gilt dann $X^0 = (1, 0, 0, 0, \ldots)$, $X^1 = (0, 1, 0, 0, \ldots)$, $X^2 = (0, 0, 1, 0, 0, \ldots)$ und allgemein $X^i = (\underbrace{0, \ldots, 0}_{i\ 0en}, 1, 0, 0, \ldots)$, $i \geq 0$. Daher lässt sich jedes $f = (a_0, a_1, a_2, \ldots) \in R[X]$ auf genau eine Weise als Linearkombination von Potenzen von $X \in R[X]$ darstellen, nämlich als*

$$f = \sum_{i \geq 0} a_i X^i. \tag{3.3}$$

(Da $a_i \neq 0$ nur für endlich viele i, sind die Summanden auf der rechten Seite von (3.3) ab einem bestimmten Index i alle 0 und der Wert der Summe ist wohldefiniert.)

Satz 3.14. *Für $f, g \in R[X]$ gilt $\deg(f + g) \leq \max\{\deg(f), \deg(g)\}$ und $\deg(fg) \leq \deg(f) + \deg(g)$, wobei die letzte Ungleichungen zur Gleichung wird, wenn R ein Integritätsring ist.*

Beweis. [Bos04], Seite 32, Bemerkung 2.1/2. □

Satz 3.15. *Sei R ein beliebiger Ring. Dann gilt: X ist in $R[X]$ kein Nullteiler.*

Beweis. Es gelte $X \cdot f = 0$ für ein $f = \sum a_i X^i \in R[X], f \neq 0$. Sei d der Grad von f. In $X \cdot f$ ist dann der Koeffizient mit Grad $d + 1$ genau $a_d \neq 0$, so das $X \cdot f$ nicht das Nullpolynom sein kann. □

Satz 3.16. *Seien R und R' Ringe und $\varphi : R \to R'$ ein Ringisomorphismus. Dann sind auch $R[X]$ und $R'[X]$ isomorph, nämlich mittels $\sum a_i X^i \mapsto \sum \varphi(a_i) X^i$.*

Beweis. Leicht nachzurechnen. □

Einsetzen in Polynome Die Schreibweise $X^7 - 19$ suggeriert, dass X eine „Variable“ ist, in die man etwas „einsetzen“ kann. Das ist tatsächlich der Fall. Natürlich kann man Elemente aus R für X einsetzen. Aber oft ist es auch sinnvoll, ein Element r aus einem Oberring R' von R in ein Polynom $f = \sum_{i=0}^{k} a_i X^i$ über R einsetzen[1]. Formal geschieht

[1] Zum Beispiel $\sqrt{2} \in \mathbb{R} - \mathbb{Z}$ in $X^2 - 2 \in \mathbb{Z}[X]$.

das Einsetzen durch die Zuordnung $r \mapsto f(r) := \sum_{i=0}^{k} a_i r^i$. Ein Element $r \in R'$ mit $f(r) = 0$ heißt *Nullstelle* von f.

Weil $R[X]$ ein Oberring von R ist, kann man auch Polynome in Polynome einsetzen. Sind beispielsweise $f = X^2 - 1$ und $g = 2X + 17$ gegeben, so ist $f(g) = 4X^2 + 68X + 288$ und $g(f) = 2X^2 + 15$. Setzt man das Polynom X in f ein, so erhält man $f(X) = f$.

Lemma 3.17. *Sei R ein Ring, $g \in R[X]$ ein Polynom und σ die Abbildung $\sigma : R[X] \to R[X], f \mapsto f(g)$. Dann gilt: $\sigma(X^t) = \sigma(X)^t$.*

Beweis. Einerseits gilt: Weil $\sigma(X) = g$ ist, ist $\sigma(X)^t = g^t$. Andererseits ergibt g in X^t eingesetzt g^t, das heißt $\sigma(X^t) = g^t$. □

Polynomdivision Eine wichtige Eigenschaft von Polynomringen ist, dass man auch dort eine Division mit Rest durchführen kann:

Satz 3.18. *Sei R ein Ring und $g = \sum_{i=0}^{d} b_i X^i \in R[X]$ ein Polynom, dessen höchster Koeffizient b_d eine Einheit ist. Dann gib es zu jedem $f \in R[X]$ eindeutig bestimmte Polynome $q, r \in R[X]$ mit*

$$f = qg + r \text{ und } \deg(r) < d.$$

Beweis. [Bos04], Seite 32, Satz 2.1/4. □

In Körpern ist jedes Element $\neq 0$ eine Einheit. Deshalb ist in Polynomringen über Körpern Division mit Rest stets möglich, außer natürlich durch das Nullpolynom.

Der Beweis des letzten Satzes liefert ein konstruktives Verfahren, mit dem man q und r bestimmen kann, nämlich die Polynomdivision. Dies ähnelt sehr der schriftlichen Division, die man in der Schule lernt. Abbildung 3.1 zeigt ein Beispiel.

Euklidische Ringe Die Gemeinsamkeiten von Fakt 3.1 und Satz 3.18 werden verallgemeinert in folgender Definition:

Definition 3.19. *Ein Integritätsring R mit einer Abbildung $\delta : R - \{0\} \to \mathbb{N}$ heißt* euklidischer Ring*, wenn gilt: Zu jedem $f, g \in R, g \neq 0$ gibt es $q, r \in R$, für die gilt:*

$$f = qg + r, \quad \text{wobei } \delta(r) < \delta(g) \text{ oder } r = 0.$$

Satz 3.20. *Jeder euklidische Ring ist ein Hauptidealring.*

```
 (X^3 + 2X^2 - 3X - 1) : (X^2+X) = X + 1
-(X^3 +  X^2)
 ------------
         X^2
       -(X^2 +  X)
        ----------
              -4X - 1
```

Abbildung 3.1.: Ein Beispiel für Polynomdivision.

Beweis. [Bos04], Seite 45, Satz 2.4/2. Es wird wieder „Division mit Rest“ benutzt, wie im Beweis von Satz 3.3. □

Satz 3.21. *Ist K ein Körper, dann ist $K[X]$ ein Hauptidealring.*

Beweis. $K[X]$ ist ein euklidischer Ring, weil die Grad-Abbildung deg die Rolle von δ in Definition 3.19 übernehmen kann. Damit folgt die Behauptung aus Satz 3.20. □

3.3. Teilbarkeit

Wenn man verstehen will, was eine Primzahl ist, muss man wissen, was es bedeutet, dass eine Zahl durch eine andere teilbar ist. Wenn man den Frobenius-Test verstehen will, muss man wissen, was es bedeutet, dass ein Polynom durch andere Polynome teilbar — oder auch nicht teilbar, also irreduzibel — ist. Deshalb wird nun Teilbarkeit definiert und bei der Gelegenheit einige Aussagen zur Teilbarkeit zusammengetragen, die später bedeutsam sein werden.

3.3.1. Teilbarkeit in Integritäts- und Hauptidealringen

Es R ein Integritätsring und $r, t \in R$. Man schreibt $t|r$ und sagt „t ist ein *Teiler* von r“, „t *teilt* r“ oder „r ist *durch* t *teilbar*“, falls es $t' \in R$ gibt, so dass gilt: $r = t \cdot t'$. Für $|$ gelten folgende Rechenregeln, die man leicht nachprüfen kann (alle Variablen bezeichnen Elemente aus R):

1. $a|b$ und $b|c \Rightarrow a|c$.
2. $a|b$ und $a|c \Rightarrow a|xb + yc$.

3. $a|b$ und $b|a \Rightarrow a, b$ sind assoziiert.

Im Prinzip ist $|$ nur eine andere Schreibweise für Hauptideale: $t|r$ ist äquivalent zu $r \in (t)$. Oft wird mit den Restklassen modulo (t) gerechnet. Für $a + (t) = b + (t)$ schreibt man dann $a \equiv b \bmod t$. Wegen (3.1) ist dies äquivalent zu $a - b \in (t)$, also $t|a - b$.

Seien $a_1, a_2, \ldots, a_k \in R$ gegeben. $t \in R$ heißt *gemeinsamer Teiler* von $a_1, \ldots, a_k$, falls $t|a_i$ für alle $i = 1, \ldots, k$. t heißt *größter gemeinsamer Teiler* von $a_1, \ldots, a_k$, wenn t gemeinsamer Teiler von $a_1, \ldots, a_k$ ist und für jeden gemeinsamen Teiler t' von $a_1, \ldots, a_k$ gilt: $t'|t$. Zwei Ringelemente mit 1 als größtem gemeinsamen Teiler heißen *teilerfremd*.

Die größten gemeinsamen Teiler von $a_1, \ldots, a_k$ sind assoziiert. Deshalb wird, wenn t ein größter gemeinsamer Teiler dieser Elemente ist, dafür auch geschrieben

$$\mathrm{ggT}(a_1, \ldots, a_k) = t.$$

In $\mathbb{Z}$ kann diese nicht ganz exakte Schreibweise vermieden werden: Weil $\mathbb{Z}^* = \{1, -1\}$ die Einheiten von $\mathbb{Z}$ sind, unterscheiden sich zwei größte gemeinsame Teiler ganzer Zahlen höchstens in ihrem Vorzeichen. Daher wird festgelegt, dass mit *dem* größten gemeinsamen Teiler in $\mathbb{Z}$ stets der *positive* größte gemeinsame Teiler gemeint ist.

Satz 3.22. *Sei R ein Integritätsring und $a_1, \ldots, a_k \in R$. Wenn das von diesen Elementen erzeugte Ideal ein Hauptideal ist, also*

$$(t) = (a_1, \ldots, a_k)$$

für ein $t \in R$ gilt, dann haben $a_1, \ldots, a_n$ einen größten gemeinsamen Teiler, nämlich t. Insbesondere existiert in Hauptidealringen stets ein größter gemeinsamer Teiler.

Beweis. [Bos04], Seite 50, Satz 2.4/13(i). □

3.3.2. Kongruenzen mit ganzen Zahlen

Wir wollen uns nun mit Teilbarkeit im Ring der ganzen Zahlen beschäftigen. Zunächst sei Satz 3.22 in $\mathbb{Z}$ formuliert:

Satz 3.23. *Seien $a_1, \ldots, a_k \in \mathbb{Z}$ gegeben und $t = \mathrm{ggT}(a_1, \ldots, a_k)$. Dann gilt:*

$$\{x_1a_1 + x_2a_2 + \ldots + x_ka_k \mid x_1, \ldots x_k \in \mathbb{Z}\} = \{mt \mid m \in \mathbb{Z}\} \tag{3.4}$$

Seien $n > 1$ und a, b ganze Zahlen. In Abschnitt 3.3.1 wurde schon festgelegt, was unter $a \equiv b \bmod n$ zu verstehen ist, nämlich dass a und b zur gleichen Restklasse modulo (n) — oder modulo n, wie es ab jetzt einfacher heißen soll — gehören. Man sagt auch: a und b sind *kongruent modulo n*. Beispielsweise ist $3 \equiv 7 \bmod 4$ und $34 \equiv 0 \bmod 17$. Anstelle von $\mathbb{Z}/n\mathbb{Z} = \{0 + (n), 1 + (n), 2 + (n), \ldots, n - 1 + (n)\}$ verwendet man oft $\mathbb{Z}_n = \{0, 1, 2, \ldots, n - 1\}$.

Gleichungen in Restklassen ganzer Zahlen werden für die Analyse der Primzahltests grundlegend sein. Als Beispiel betrachten wir

$$3x \equiv 9 \mod 12, \tag{3.5}$$

$$4x \equiv 9 \mod 12, \tag{3.6}$$

$$5x \equiv 9 \mod 12. \tag{3.7}$$

Da $\mathbb{Z}/12\mathbb{Z}$ nur aus 12 Restklassen besteht, kann man durch Probieren die Lösungen finden, bzw. die Nichtlösbarkeit beweisen: (3.5) hat 3 Lösungen, nämlich 3, 7 und 10. (3.6) ist nicht lösbar, während (3.7) genau eine Lösung hat, nämlich 9. Woran das liegt klärt der folgende

Satz 3.24. *Seien $n, a, b \in \mathbb{Z}$, $n > 1$ und $t = \mathrm{ggT}(a, n)$. Dann ist*

$$ax \equiv b \mod n \tag{3.8}$$

lösbar genau dann, wenn $t|b$. Wenn (3.8) lösbar ist, hat die Gleichung genau t Lösungen modulo n, das heißt genau die Elemente von exakt t Restklassen aus $\mathbb{Z}/n\mathbb{Z}$ sind Lösungen. Ist x eine Lösung, so auch $x + n/t, x + 2n/t, \ldots, x + (t - 1)n/t$.

Beweis. [IR90], Seite 32, Proposition 3.3.1. □

Für natürliche Zahlen $n \geq 2$ ist die *Eulersche φ-Funktion* definiert als

$$\varphi(n) := |(\mathbb{Z}/n\mathbb{Z})^*|.$$

$\varphi(n)$ gibt also die Mächtigkeit der Gruppe der multiplikativen Einheiten in $\mathbb{Z}_n$ an — bzw. die Anzahl der zu n teilerfremden natürlichen Zahlen $< n$, wie der folgende Satz zeigt:

Satz 3.25.

$$\mathbb{Z}_n^* = \{a + n\mathbb{Z} \mid 1 \leq a \leq n - 1, \mathrm{ggT}(a, n) = 1\}.$$

Beweis. Nach Satz 3.23 gilt $\mathrm{ggT}(a, n) = 1$ genau dann, wenn $xa + yn = 1$ ganze Zahlen x, y als Lösungen hat. x ist gegebenenfalls Inverses zu a. — Alternativ siehe [IR90], Seite 33, Proposition 3.3.2. □

3.4. Primfaktorzerlegung

Nur auf triviale, uninteressante Weise teilbar sind in $\mathbb{Z}$ bzw. $\mathbb{N}$ die Primzahlen. Das gleiche gilt in $K[X]$ für die irreduziblen Polynome. Beides sind die multiplikativen Bausteine des jeweiligen Rings. Dies wird nun detailliert dargestellt und danach angewendet.

3.4.1. Primfaktorzerlegung allgemein — faktorielle Ringe

Primideale und maximale Ideale Zunächst benötigen wir zwei neue Definitionen: Sei R ein Ring. Ein Ideal $P \subset R$ heißt *Primideal*, falls für $a, b \in R$ stets gilt: aus $ab \in P$ folgt $a \in P$ oder $b \in P$. Ein Ideal $M \subset R$ heißt *maximales Ideal*, wenn $M \neq R$ ist und gilt: Ist I ein Ideal mit $M \subset I \subset R$, so folgt $I = M$ oder $I = R$.

Satz 3.26. *Sei R ein Ring. Dann gilt:*

1. *Ein Ideal $P \subset R$ ist genau dann ein Primideal, wenn R/P ein Integritätsring ist.*
2. *Ein Ideal $M \subset R$ ist genau dann ein maximales Ideal, wenn R/M ein Körper ist.*
3. *Jedes maximale Ideal ist ein Primideal.*

Beweis. [Bos04], Seite 41, Satz 2.3/8. □

Irreduzible Elemente und Primelemente Sei nun R ein Integritätsring und $p \neq 0$ ein Element, das keine Einheit ist. Dann heißt p *irreduzibel*, falls für jede Darstellung $p = xy$ mittels $x, y \in R$ gilt: $x \in R^*$ oder $y \in R^*$. Wenn p nicht irreduzibel ist, nennt man p *reduzibel*. Das Element p heißt *prim*, falls aus $p|xy$ mit $x, y \in R$ stets $p|x$ oder $p|y$ folgt. Letzteres ist genau dann der Fall, wenn (p) ein Primideal ist. In jedem Ring gilt ([Bos04], Seite 46, Bemerkung 2.4/5):

$$(p) \text{ maximales Ideal} \Rightarrow p \text{ prim} \Rightarrow p \text{ irreduzibel}.$$

Faktorielle Ringe

Satz 3.27. *Sei R ein Hauptidealring. Dann gilt:*

$$(p) \text{ maximales Ideal} \Leftrightarrow p \text{ prim} \Leftrightarrow p \text{ irreduzibel.}$$

Beweis. [Bos04], Seite 46, Satz 2.4/6. □

Als Folge ergibt sich:

Satz 3.28. *Jeder Hauptidealring R ist* faktoriell*, dass heißt es gilt:*

- *Ein Element $r \in R$ ist genau dann irreduzibel, wenn es ein Primelement ist.*
- *Jedes Element $r \in R$ lässt sich bis auf Assoziiertheit und Reihenfolge eindeutig als Produkt von irreduziblen Elementen darstellen. Das heißt: Bildet man $P \subset R$, indem man aus jeder Klasse zueinander assoziierter Primelemente eins auswählt, dann kann man jedes $a \in R$ darstellen als*

$$a = \varepsilon \prod_{p \in P} p^{m_p(a)}, \tag{3.9}$$

wobei ε ein Einheit ist und $m_p(a) \in \mathbb{N}$. ε und $m_p(a)$ sind durch a eindeutig bestimmt. In dem Produkt tauchen nur endlich viele Faktoren auf, das heißt nur für endlich viele $p \in P$ ist $m_p(a) \neq 0$. Die Darstellung (3.9) heißt Primfaktorzerlegung *von a.*

Beweis. [Bos04], Seite 47-48, Satz 2.4/7 und Satz 2.4.10. □

Satz 3.29. *$\mathbb{Z}$ ist ein faktorieller Ring. Für jeden Körper K ist der Polynomring $K[X]$ ein faktorieller Ring.*

Beweis. Satz 3.9, Satz 3.21 und Satz 3.28. □

Zwei Beispiele sollen die vorangegangen Tatsachen etwas illustrieren: Die ganze Zahl -12 lässt sich unter anderem darstellen als $-2 \cdot 2 \cdot 3 = 2 \cdot (-2) \cdot 3 = 2 \cdot 2 \cdot (-3) = -2 \cdot (-2) \cdot (-3)$. -2 und 2 sind assoziiert, weil sie sich nur um eine Einheit, nämlich -1 unterscheiden. 2 und -2 sind irreduzibel, also Primelemente. Ebenso sind 3 und -3 assoziierte Primelemente in $\mathbb{Z}$. — Das Polynom $X^2 - 2X + 1 \in \mathbb{Q}[X]$ hat unter anderen die Zerlegungen $(X-1)(X-1) = (-(X-1))(-(X-1)) = (X/2 - 1/2)(2X - 2)$. Alle Elemente aus $\mathbb{Q} - \{0\}$ sind Einheiten in $\mathbb{Q}[X]$.

Es gibt auch faktorielle Ringe, die keine Hauptidealringe sind, zum Beispiel $\mathbb{Z}[X]$.

Primfaktorzerlegung und der größte gemeinsame Teiler

Satz 3.30. *R sei ein faktorieller Ring und P ein Vertretersystem der Primelemente von R. Weiter seien $x_1, \ldots, x_n \in R$ gegeben mit der Primfaktorzerlegung*

$$x_i = \varepsilon_i \prod_{p \in P} p^{m_p(x_i)}, \quad i = 1, \ldots, n.$$

Dann existiert $\mathrm{ggT}(x_1, \ldots, x_n)$ *und es gilt:*

$$\mathrm{ggT}(x_1, \ldots, x_n) = \prod_{p \in P} p^{\min\{m_p(x_1), \ldots, m_p(x_n)\}}.$$

Beweis. [Bos04], Seite 50, Satz 2.4/12. □

Damit sind nun die folgenden Eigenschaften leicht einzusehen:

Satz 3.31. *Sei R ein faktorieller Ring, zum Beispiel $\mathbb{Z}$. Weiter seien $a, b, c, p, r \in R$. p sei ein Primelement in R und für r gelte $p \nmid r$. Dann gilt:*

1. $\mathrm{ggT}(ab, ac) = a\,\mathrm{ggT}(b, c)$.
2. *Aus* $\mathrm{ggT}(a, b) = 1$ *folgt* $\mathrm{ggT}(ab, ac) = a\,\mathrm{ggT}(b, ac)$.
3. $\mathrm{ggT}(r, pa) = \mathrm{ggT}(r, a)$.

Später werden wir die folgende Aussage benötigen:

Lemma 3.32. *Seien $p > 2$ und s ungerade ganze Zahlen und j eine natürliche Zahl. Dann gilt*

$$\mathrm{ggT}(2^j s, p^2 - 1) \mid \frac{p^2 - 1}{2} \;\Rightarrow\; 2^{j+1} \mid p^2 - 1$$

Beweis. Sei μ die größte ganze Zahl mit $2^\mu \mid p^2 - 1$ und gelte $\mathrm{ggT}(2^j s, p^2 - 1) \mid \frac{p^2-1}{2}$. Wir müssen ausschließen, dass $\mu \leq j$ ist. Wir schließen indirekt und nehmen an, dass $\mu \leq j$ gilt. Dann ist $\mathrm{ggT}(2^j s, p^2 - 1) = 2^\mu\,\mathrm{ggT}(2^{j-\mu} s, \frac{p^2-1}{2^\mu}) = 2^\mu \cdot l$ für eine ungerade Zahl l. Also gilt $2^\mu l \mid \frac{p^2-1}{2}$, woraus $2^\mu \mid \frac{p^2-1}{2}$ folgt, was der Definition von μ widerspricht. □

3.4.2. Primfaktorzerlegung in $K[X]$ — irreduzible Polynome und mehrfache Nullstellen

Satz 3.33. *Sei K ein Körper und $f \in K[X]$ ein Polynom mit Nullstelle $a \in K$. Dann ist f durch $X - a$ teilbar.*

Beweis. Polynomdivision ergibt $f = (X - a)q + r$, wobei $r, q \in K[X]$ und $\deg(r) < 1$ ist, also $r \in K$. Damit folgt $0 = f(a) = (a - a)q(a) + r = r$. Also ist f durch $X - a$ teilbar. □

Sei nun $f \in K[X]$ gegeben und $a \in K$ eine Nullstelle von f. Dann ist das irreduzible Element — und damit Prim-Element — $X - a$ ein Teiler von f und kommt in der Primfaktorzerlegung von f mit einer gewissen Vielfachheit $r > 0$ vor: $f = (X - a)^r g$ für ein geeignetes $g \in K[X]$. r wird *Vielfachheit* der Nullstelle a genannt. Weil in $K[X]$ der Grad eines Produktes die Summe der Grade der Faktoren ist (Satz 3.14), folgt:

Satz 3.34. *Sei K ein Körper und $f \in K[X]$ ein Polynom vom Grad $n \geq 0$. Dann hat f höchstens n Nullstellen, wobei jede Nullstelle mit ihrer Vielfachheit gezählt wird.*

Satz 3.35. *Sei K ein Körper und $f, g \in K[X]$ zwei normierte Polynome vom Grad höchstens $n \geq 0$. Haben f und g mindestens n gemeinsame Nullstellen, dann gilt $f = g$.*

Beweis. Wir wenden Satz 3.34 an. Es muss sowohl f als auch g den Grad genau n haben, denn sonst wäre f oder g ein Gegenbeispiel zu Satz 3.34. Damit hat $f - g$ höchstens den Grad $n - 1$, jedoch n Nullstellen, ist also ebenfalls ein Gegenbeispiel zu Satz 3.34, wenn nicht $f - g$ das Nullpolynom, also $f = g$ ist. □

Satz 3.36. *Sei K ein Körper. Lineare Polynome über K sind irreduzibel. Ist $f \in K[X]$ ein Polynom vom Grad 2 oder 3, so ist f genau dann irreduzibel, wenn f keine Nullstellen in K hat.*

Beweis. Sei $f \in K[X]$ zerlegt in $f = gh$, $g, h \in K[X]$. Wegen Satz 3.14 folgt $\deg(f) = \deg(g) + \deg(h)$.

Falls f linear ist, also $\deg(f) = 1$, ergibt sich daher $\deg(g) = 0$ oder $\deg(h) = 0$, also ist $g \in K^* = K[X]^*$ oder $h \in K^* = K[X]^*$.

Falls $\deg(f) = 2$ oder $\deg(f) = 3$ und f reduzibel ist, muss für geeignete g und h gelten $\deg(g) = 1$ oder $\deg(h) = 1$. Die Nullstelle des linearen Polynoms g bzw. h ist gleichzeitig eine Nullstelle von f. Damit ist gezeigt, dass f Nullstellen hat, wenn f reduzibel ist. Die Umkehrung liefert Satz 3.33. □

3.4.3. Primfaktorzerlegung in $\mathbb{Z}$ — Primzahlen

Wir wollen Satz 3.29 und (3.9) auf $\mathbb{Z}$ anwenden. Als Vertretersystem der Primelemente in $\mathbb{Z}$ wählen wir die positiven Primelemente. Diese werden auch *Primzahlen* genannt und mit $\mathbb{P}$ bezeichnet. Damit ergibt sich:

Satz 3.37. *Jede ganze Zahl außer 0 lässt sich bis auf die Reihenfolge eindeutig als Produkt von Primzahlen darstellen. Genauer: zu $z \in \mathbb{Z}$, $z \neq 0$ gib es $\omega \in \mathbb{N}$, $p_1 < p_2 < \ldots < p_\omega \in \mathbb{P}$, $m_1, \ldots, m_\omega \in \mathbb{N}, m_1, \ldots, m_\omega > 0$, und $\varepsilon \in \{1, -1\}$, für die gilt:*

$$z = \varepsilon p_1^{m_1} p_2^{m_2} \cdots p_\omega^{m_\omega}$$

$\omega, p_1, \ldots, p_\omega, m_1, \ldots, m_\omega$ und ε sind durch z eindeutig bestimmt.

Die ganzen Zahlen lassen sich also einteilen in vier disjunkte Gruppen: die 0, die Einheiten $\{1, -1\}$, die Primelemente $\{p, -p \mid p \in \mathbb{P}\}$ und die *zusammengesetzt*en Zahlen.

Ein Zahl n heißt *quadratfrei*, falls es keine Primzahl p gibt, für die $p^2|n$ gilt, oder — was dazu äquivalent ist — falls in der Primfaktorzerlegung alle Exponenten den Wert 1 haben. Damit kann eine für die Zahlentheorie wichtige Funktion definiert werden, die in dieser Arbeit allerdings nur an einer Stelle vorkommt:

Definition 3.38 (Möbiussche μ-Funktion). *Die* Möbiussche μ-Funktion *ist definiert als:*

$$\mu(n) = \begin{cases} 1, & \text{falls } n = 1, \\ 0, & \text{falls } n \text{ nicht quadratfrei ist,} \\ (-1)^l, & \text{falls } n \text{ quadratfrei ist} \\ & \quad \text{und die Primfaktorzerlegung } n = p_1 p_2 \cdots p_l \text{ hat.} \end{cases}$$

Lemma 3.39. *Sind n und m teilerfremde natürliche Zahlen, so gilt*

$$\mu(mn) = \mu(m)\mu(n).$$

Beweis. Nachrechnen mit Hilfe der Definition von μ und der Primfaktorzerlegung. □

Satz 3.40 (Chinesischer Restsatz für ganze Zahlen). *Sei $n = \prod_{i=1}^{\omega} p_i^{m_i}$ die Primfaktorzerlegung von n. Dann gilt:*

$$\begin{aligned} \mathbb{Z}_n &\simeq \mathbb{Z}_{p_1^{m_1}} \times \ldots \times \mathbb{Z}_{p_\omega^{m_\omega}} \\ \mathbb{Z}_n^* &\simeq \mathbb{Z}_{p_1^{m_1}}^* \times \ldots \times \mathbb{Z}_{p_\omega^{m_\omega}}^* \\ \mathbb{Z}_n[X] &\simeq \mathbb{Z}_{p_1^{m_1}}[X] \times \ldots \times \mathbb{Z}_{p_\omega^{m_\omega}}[X] \end{aligned}$$

Ist darüber hinaus f ein Polynom in $\mathbb{Z}[X]$, so gilt

$$\mathbb{Z}_n[X]/(f \bmod n) \simeq \mathbb{Z}_{p_1^{m_1}}[X]/(f \bmod p_1^{m_1}) \times \ldots \times \mathbb{Z}_{p_\omega^{m_\omega}}[X]/(f \bmod p_\omega^{m_\omega})$$

Beweis. $p_i^{m_i}$ und $p_j^{m_j}$ sind teilerfremd (Satz 3.30), also ist $(p_i^{m_i}) + (p_j^{m_j}) = \mathbb{Z}$ für $i \neq j$ (Satz 3.22). Außerdem ist $(n) = \bigcap_{i=1}^{\omega}(p_i^{m_i})$. Damit kann der Chinesische Restsatz, Satz 3.11, angewendet werden und es folgen die Aussage über die Struktur von $\mathbb{Z}_n$ und $\mathbb{Z}_n^*$. Die Aussage über $\mathbb{Z}_n[X]$ ergibt sich aus Satz 3.16. Damit kann auch die letzte Behauptung bewiesen werden, nämlich indem man folgende leicht nachzurechnende Tatsache benutzt:

Seien R und $R_1, \ldots, R_k$ Ringe, wobei $R \simeq R_1 \times \ldots R_k$ mittels $\varphi = (\varphi_1, \ldots, \varphi_k)$. Weiter sei I ein Ideal in R. Dann ist $I_j = \varphi_j(I)$ Ideal in R_j, $j = 1, \ldots, k$, und es gilt: $R/I \simeq R_1/I_1 \times \ldots \times R_k/I_k$. □

Damit kann auch der Wert der Eulerschen φ-Funktion genauer bestimmt werden:

Satz 3.41. *1. Für teilerfremde $n, m \in \mathbb{N}$ ist $\varphi(mn) = \varphi(m)\varphi(n)$.*

2. Sei p eine Primzahl und $k \in \mathbb{N}$. Dann ist $\varphi(p^k) = (p-1)p^{k-1}$

Im Sinne des Satzes definieren wir $\varphi(1) := 1$.

Beweis. 1. folgt aus dem Chinesischen Restsatz 3.11: Für teilerfremde n, m gilt $(n) + (m) = \mathbb{Z}$ und $(n) \cap (m) = (nm)$. Also ist $\mathbb{Z}_{nm}^* \simeq \mathbb{Z}_n^* \times \mathbb{Z}_m^*$.

2. ergibt sich aus Satz 3.25: Teilerfremd zu p^k sind unter den Zahlen $1, 2, \ldots, p^k - 1$ alle außer außer den Vielfachen von p, nämlich $p, 2p, 3p, 4p, \ldots, (p^{k-1}-1)p$. Also gehören $p^k - 1 - (p^{k-1} - 1) = p^{k-1}(p-1)$ Elemente zu $\mathbb{Z}_{p^k}^*$. □

3.5. Zyklische Gruppen

Wir kommen nun noch einmal zu den Gruppen zurück. Ist G eine Gruppe und $a \in G$, so können wir die Menge

$$\langle a \rangle = \{e, a, a^{-1}, a^2, a^{-2}, a^3, a^{-3}, a^4, a^{-4}, \ldots\}$$

betrachten. $\langle a \rangle$ ist eine Untergruppe von G. Gilt $G = \langle a \rangle$ für ein $a \in G$, so wird G *zyklische* Gruppe und a *erzeugendes Element* von G genannt. Die *Ordnung* eines Elements $a \in G$ ist definiert als die Ordnung der von a erzeugten zyklischen Untergruppe

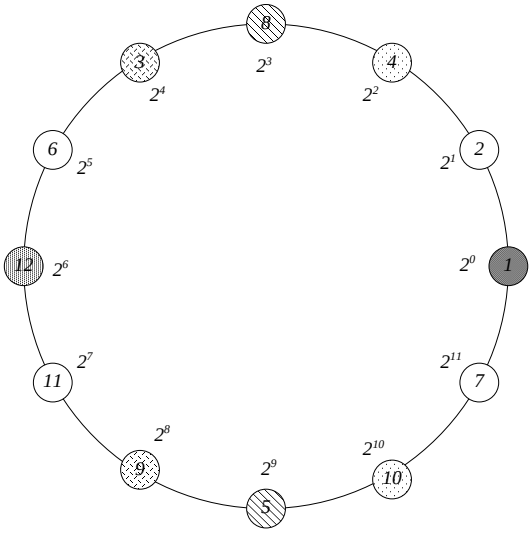

Abbildung 3.2.: Die zyklische Gruppe $(\mathbb{Z}_{13}^*, \cdot)$.

$\langle a \rangle$. Die Menge $\mathbb{Z}$ der ganzen Zahlen, aufgefasst als abelsche Gruppe mit der üblichen Addition, ist eine zyklische Gruppe, sie wird von 1, aber auch von -1 erzeugt. Für $n > 1$ ist $\mathbb{Z}/n\mathbb{Z} = \langle 1 + n\mathbb{Z} \rangle$ zyklisch. Damit sind nun schon bis auf Isomorphie alle zyklischen Gruppen erwähnt:

Satz 3.42. *Jede zyklische Gruppe $G = \langle g \rangle$ ist entweder mittels $z \mapsto g^z$ zu $\mathbb{Z}$ oder mittels $z + n\mathbb{Z} \mapsto g^z$ zu einer Gruppe $\mathbb{Z}/n\mathbb{Z}$, $n \in \mathbb{Z}$, $n > 1$, isomorph.*

Beweis. Eine Gruppe G ist genau dann zyklisch, wenn es einen surjektiven Homomorphismus ψ von $\mathbb{Z}$ nach $\langle g \rangle = G$ gibt, nämlich $\psi : z \mapsto g^z$. Damit folgt aus dem Homomorphiesatz 3.5, dass $\langle g \rangle$ isomorph zu $\mathbb{Z}/H$ ist, wobei $H = \ker \psi$ eine Untergruppe von $\mathbb{Z}$ ist. Nach Satz 3.3 hat $\mathbb{Z}$ aber nur $n\mathbb{Z}$, $n \in \mathbb{Z}$, als Untergruppen. — Ausführlicher in [Bos04], Seite 21, Satz 1.3/3. □

Wir wollen ein Beispiel betrachten, damit wir ein Gefühl für die zyklischen Gruppen bekommen. Wir wählen den Körper $\mathbb{Z}_{13}$. Durch Probieren erkennt man, dass 2 ein erzeugendes Element von $(\mathbb{Z}_{13}^*, \cdot)$ ist. In Abbildung 3.2 sind ist die Gruppe $(\mathbb{Z}_{13}^*, \cdot)$, wie sie von 2 erzeugt wird, abgebildet. Man sieht, dass sie — wie die Gruppe $(\mathbb{Z}_{12}, +)$, zu der sie isomorph ist, — mehrere nichttriviale Untergruppen hat. Beispielsweise bilden $\{8, 12, 5, 1\}$ eine Untergruppe der Ordnung 4. Sie besteht genau aus den Potenzen von 2^k, für die k durch 3 teilbar ist, Ebenso gibt es eine Untergruppe mit 3 Elementen,

die gerade aus den Potenzen von 2 mit durch 4 teilbaren Exponenten besteht, nämlich $\{4, 9, 1\}$. Auch für jeden anderen Teiler t von 12 bilden die Elemente der Form $2^{kt}, k \in \mathbb{Z}$, eine Untergruppe mit $12/t$ Elementen. Allgemein folgt aus dem Satz von Lagrange (Satz 3.4), dass alle Untergruppen von $(\mathbb{Z}_p^*, \cdot)$ eine Ordnung haben, die $p-1 = \varphi(p) = \operatorname{ord} \mathbb{Z}_p^*$ teilt.

Für den Frobenius-Test bedeutsam sind folgende zyklischen Gruppen:

Satz 3.43. *Für Primzahlen p ist $(\mathbb{Z}_p^*, \cdot)$ ist zyklisch und hat die Ordnung $\varphi(p) = p-1$. Dies ist ein Spezialfall der folgenden beiden Aussagen:*

1. *Sei $(K, +, \cdot)$ ein endlicher Körper mit q Elementen. Dann ist $(K^*, \cdot)$ zyklisch und hat die Ordnung $q-1$.*

2. *Für Primzahlen $p \geq 3$ und natürliche Zahlen $l > 1$ ist $(\mathbb{Z}_{p^l}^*, \cdot)$ zyklisch und hat die Ordnung $\varphi(p^l) = p^{l-1}(p-1)$.*

Beweis. Die Aussagen über die Ordnung ergeben sich aus Satz 3.41. Die Aussage über die Körper wird in [Bos04], Seite 121, Satz 3.6/14 bewiesen, die Aussage über $\mathbb{Z}_{p^l}^*$ in [IR90], Kapitel 4, Seite 43, Theorem 2. □

Satz 3.44. *Sei G eine endliche Gruppe. Dann gilt $a^{\operatorname{ord} G} = e$ für jedes $a \in G$.*

Beweis. Sei k die Ordnung der zyklischen Untergruppe $\langle a \rangle$ von G. Nach Satz 3.42 ist $\langle a \rangle$ isomorph zu $\mathbb{Z}/k\mathbb{Z}$ mittels $z \bmod k \mapsto a^z$. Da in $\mathbb{Z}/k\mathbb{Z}$ die Werte 0 und k zur gleichen Restklasse gehören, ist $e = a^0 = a^k$. Nach dem Satz von Lagrange (Satz 3.4) ist die Ordnung k der Untergruppe $\langle a \rangle$ von G ein Teiler von $\operatorname{ord} G$, so dass ein k' existiert mit $kk' = \operatorname{ord} G$. Also ist $a^{\operatorname{ord} G} = (a^k)^{k'} = e^{k'} = e$. □

Wendet man die Sätze 3.43 und 3.44 auf $\mathbb{Z}_p$ an, so ergibt sich der

Satz 3.45 (Kleiner Satz von Fermat). *Sei p eine Primzahl und $a \not| p$. Dann gilt $a^{p-1} \equiv 1 \bmod p$.*

Die folgenden Aussagen werden für Kapitel 6 bereitgestellt.

Lemma 3.46. *Sei K ein endlicher Körper mit q Elementen, q ungerade, und g ein erzeugendes Element der multiplikativen Gruppe $(K^*, \cdot)$. Dann gilt: $1 = g^0$ und $-1 = g^{\frac{q-1}{2}}$.*

Beweis. Die erste Aussage ist trivial. Für die zweite bemerken wir, dass $\left(g^{\frac{q-1}{2}}\right)^2 = g^{q-1} = g^0 = 1$ ist, weil $(K^*, \cdot)$ die Ordnung $q-1$ hat. $g^{\frac{q-1}{2}} \neq 1$ ist also eine Quadratwurzel von 1 und muss -1 sein nach Satz 3.8. □

Lemma 3.47. *Sei $p > 2$ eine Primzahl, $s, j \geq 0$ ganze Zahlen und $d = \mathrm{ggT}(2^j s, p^2-1)$. Weiter sei K ein endlicher Körper mit p^2 Elementen. Dann hat die Gleichung $y^{2^j s} = -1$ in K genau dann Lösungen, wenn $d \mid \frac{p^2-1}{2}$. Insbesondere hat die Gleichung nur dann Lösungen, wenn $2^{j+1} \mid p^2-1$. Falls die Gleichung lösbar ist, hat sie genau d Lösungen.*

Beweis. 0 ist keine Lösung. Wir wählen nach Satz 3.43 eine erzeugendes Element g von $(K^*, \cdot)$. Jede Lösung y lässt sich darstellen als $y = g^u$ für ein $u, 0 \leq u < p^2-1$. Damit ist wegen Lemma 3.46 die angegeben Gleichung äquivalent zu $(g^u)^{2^j s} = g^{\frac{p^2-1}{2}}$. Das heißt wegen dem in Satz 3.42 angegebenen Isomorphismus $u 2^j s \equiv \frac{p^2-1}{2} \bmod (p^2-1)$. Nun folgt die Behauptung aus Satz 3.24 und Lemma 3.32. □

Lemma 3.48. *Sei p eine Primzahl und K ein Körper mit p^2 Elementen. Dann hat die Gleichung $y^s = 1$ in K genau $\mathrm{ggT}(s, p^2-1)$ Lösungen y.*

Beweis. Analog Lemma 3.47. □

3.6. Quadratische Reste und das Jacobi-Symbol

Sei $n \geq 2$ eine natürliche Zahl und a eine ganze Zahl mit $\mathrm{ggT}(a, n) = 1$. a heißt *quadratischer Rest* modulo n genau dann, wenn die Gleichung

$$x^2 \equiv a \mod n \tag{3.10}$$

eine Lösung hat. Hat sie keine Lösung, so heißt a *quadratischer Nichtrest*.

Für Primzahlen $p > 2$ lassen sich die quadratischen Reste und Nichtreste modulo p leicht charakterisieren. Weil $(\mathbb{Z}_p^*, \cdot)$ zyklisch ist, können wir nämlich ein erzeugendes Element g wählen. Es gibt dann ein $b \in \mathbb{Z}$ mit $g^b = a$. Da $x = 0$ keine Lösung von (3.10) sein kann, lässt sich jede Lösung x ebenfalls als Potenz von g schreiben, etwa in der Form $x = g^y$ für geeignetes $y \in \mathbb{Z}$. Damit ist (3.10) äquivalent zu $g^{2y} = g^b$, was aufgrund der Isomorphie von $(\mathbb{Z}_p^*, \cdot)$ und $(\mathbb{Z}_{p-1}, +)$ nicht anderes ist als $2y \equiv b \bmod (p-1)$. Nun besagt Satz 3.24, dass dies genau dann lösbar ist, wenn $\mathrm{ggT}(2, p-1) = 2$ ein Teiler von b ist. Also sind genau die Hälfte der Restklassen $\neq 0$ modulo p quadratische Reste,

nämlich die, die eine Potenz von g mit geraden Exponenten darstellen. Wir erhalten also:

Satz 3.49. *Sei $p > 2$ eine Primzahl und g ein erzeugendes Element von $(\mathbb{Z}_p^*, \cdot)$. Dann sind $\{1, g^2, g^4, \ldots, g^{p-3}\}$ die quadratischen Reste modulo p und $\{g, g^3, g^5, \ldots, g^{p-2}\}$ sind die quadratischen Nichtreste modulo p. Ist r ein beliebiger quadratischer Rest, dann durchläuft rx^2, $x \in \mathbb{Z}_p^*$, alle quadratischen Reste, wobei jeder quadratische Rest genau zweimal auftaucht. Ist r ein beliebiger quadratischer Nichtrest, so durchläuft rx^2, $x \in \mathbb{Z}_p^*$, alle quadratischen Nichtreste, wobei ebenfalls jeder Nichtrest genau zweimal auftaucht.*

Als Abkürzung wird das Legendre-Symbol verwendet, das für eine ganze Zahl a und eine Primzahl $p > 2$ wie folgt definiert ist:

$$\left(\frac{a}{p}\right) = \begin{cases} 1, & \text{falls } a \text{ quadratischer Rest modulo } p \text{ ist,} \\ -1, & \text{falls } a \text{ quadratischer Nichtrest modulo } p \text{ ist,} \\ 0, & \text{falls } p|a. \end{cases}$$

Eine Verallgemeinerung des Legendre-Symbols ist das Jacobi-Symbol. Es ist für beliebige ungerade positive ganze Zahlen $n > 1$ und ganze Zahlen a definiert als:

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right),$$

wobei $n = p_1 p_2 \cdots p_k$ eine Primfaktorzerlegung von n ist. Wenn a ein quadratischer Rest modulo n ist, dann gilt auch für das Jacobi-Symbol $\left(\frac{a}{n}\right) = 1$. Die Umkehrung gilt im Allgemeinen nicht. Beispielsweise ist $\left(\frac{2}{15}\right) = \left(\frac{2}{3}\right)\left(\frac{2}{5}\right) = -1 \cdot (-1) = 1$, aber 2 ist kein quadratischer Rest modulo 15.

Jacobi- und Legendre-Symbol erfüllen einige grundlegende Eigenschaften, die man leicht beweisen kann.

Satz 3.50. *1.* $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$

2. $\left(\frac{a}{nm}\right) = \left(\frac{a}{n}\right)\left(\frac{a}{m}\right)$

3. p sei Primzahl und $\left(\frac{a}{p}\right) \neq 0$. Dann ist $\left(\frac{a}{np^2}\right) = \left(\frac{a}{n}\right)$.

Der Hauptzweck des Jacobi-Symbols ist, dass man damit das Legendre-Symbol schnell ausrechnen kann. Dies ist möglich aufgrund folgender Zusammenhänge:

Satz 3.51 (Quadratisches Reziprozitätsgesetz). *Seien $a, b \in \mathbb{Z}$ ungerade, $a, b > 0$. Dann gilt:*

1. $\left(\frac{-1}{b}\right) = \begin{cases} 1, & \text{falls } b \equiv 1 \bmod 4, \\ -1 & \text{falls } b \equiv 3 \bmod 4. \end{cases}$

2. $\left(\frac{2}{b}\right) = \begin{cases} 1, & \text{falls } b \equiv 1 \text{ oder } 7 \bmod 8, \\ -1, & \text{falls } b \equiv 3 \text{ oder } 5 \bmod 8. \end{cases}$

3. $\left(\frac{a}{b}\right) = \begin{cases} \left(\frac{b}{a}\right), & \text{falls } a \equiv 1 \bmod 4 \text{ oder } b \equiv 1 \bmod 4, \\ -\left(\frac{b}{a}\right), & \text{falls } a \equiv 3 \bmod 4 \text{ und } b \equiv 3 \bmod 4. \end{cases}$

Beweis. [IR90], Kapitel 5. □

3.7. Endliche Körper

Wir wollen uns nun mit nun endlichen Körpern beschäftigen, also mit Körpern, die nur endlich viele Elemente haben. Ein bestimmte Klasse von endlichen Körpern kennen wir bereits:

Satz 3.52. *Für jede Primzahl p ist $\mathbb{Z}/p\mathbb{Z} = \mathbb{Z}_p$ ein Körper. Dieser Körper wird auch mit $\mathbb{F}_p$ bezeichnet.*

Beweis. Satz 3.9, Satz 3.27 und Satz 3.26. □

Es gibt noch andere endliche Körper außer $\mathbb{Z}_p$. Bevor auf die für den Frobenius-Test benötigten, nämlich die durch quadratische Erweiterung konstruierten, näher eingegangen wird, sollen ganz allgemein einige wichtige Eigenschaften endlicher Körper dargestellt werden.

3.7.1. Allgemeine Aussagen

Sei im Folgenden K ein endlicher Körper mit q Elementen und $L \supset K$ ebenfalls ein Körper, ein Oberkörper von K. Als Erstes werden die Elemente von K gegenüber denen von L charakterisiert.

Satz 3.53. *Sei K ein endlicher Körper mit q Elementen und $L \supset K$ ein Oberkörper von K. Dann gilt für $a \in L$:*

$$a \in K \;\Leftrightarrow\; a^q = a \Leftrightarrow \; a \text{ ist Nullstelle von } X^q - X$$

Beweis. Wir betrachten das Polynom $f = X^q - X \in K[X] \subset L[X]$ und zeigen, dass die Nullstellen von f in L genau die Elemente von K sind. Sei zunächst $a \in K$. Der Wert $a = 0$ ist selbstverständlich Nullstelle von f. Ist $a \neq 0$ eine Element von K, so gilt wegen Satz 3.44 $a^{q-1} = 1$, woraus $f(a) = 0$ folgt.

Da K aus genau q Elementen besteht, kennen wir nun q verschiedene Nullstellen von f. f kann jedoch nicht mehr als q Nullstellen haben, weil der Grad von f nur q ist (Satz 3.34). Damit sind die Nullstellen von f in L genau die Elemente von K. □

Die durch den letzten Satz beschriebene Situation lässt sich auch so formulieren: Die nach Satz 3.29 existierende eindeutige Darstellung von $X^q - X$ als Produkt von Primelementen — also irreduziblen Polynomen — ist

$$\prod_{a \in K} (X - a) = X^q - X.$$

Nicht nur die Elemente von K in ihrer Gesamtheit, sondern auch jedes einzelne Element aus L, insbesondere die Elemente von $L \setminus K$, lassen sich mit einem Polynom über K charakterisieren:

Satz 3.54. *Sei L ein endlicher Oberkörper des Körpers K und $a \in L$. Dann gibt es genau ein normiertes Polynom kleinsten Grades $f \in K[X]$ mit $f(a) = 0$. f ist irreduzibel über K, erzeugt den Kern der Einsetzungshomomorphismus $K[X] \to L, g \mapsto g(a)$ und heißt* Minimalpolynom *von a über K.*

Beweis. Wir begründen, dass es ein Polynom $g \in K[X]$ gibt, von dem a eine Nullstelle ist. Dazu betrachten wir die Abbildung $\varphi : K[X] \to L,\ g \mapsto g(a)$, die a in Polynome einsetzt. Da L nur endlich viele Elemente beinhaltet, sich aber in $K[X]$ unendlich viele Polynome befinden, muss es zwei verschiedene Polynome $g_1, g_2 \in K[X]$ geben mit $\varphi(g_1) = \varphi(g_2)$. Daraus ergibt sich, dass a eine Nullstelle des Polynoms $g_1 - g_2$ ist.

Damit haben wir gezeigt, dass a *algebraisch* über K ist, das heißt, dass $c_1, \ldots, c_n \in K$ existieren, für die $a^n + c_1 a^{n-1} + \ldots + c_n = 0$. Dass über K algebraische $a \in L$ die im Satz behauptete Eigenschaft aufweisen, wird beispielsweise in [Bos04], Seite 91, Bemerkung 3.2/5 bewiesen. □

Nach diesem Satz ist jedes normierte über K irreduzible Polynom $f \in K[X]$ das Minimalpolynom aller seiner Nullstellen, die sich in $L \supset K$ befinden können. Sind zwei normierte über K irreduzible Polynome $f, g \in K[X]$ gegeben und ist $a \in L \supset K$ eine gemeinsame Nullstelle von f und g, so muss $f = g$ sein, weil sowohl f als auch g das eindeutig bestimmte Minimalpolynom von a ist. Also haben Minimalpolynome entweder keine gemeinsamen Nullstellen oder sind identisch.

Wenn $\cdot : L \times L \to L$, die Multiplikation im Körper L, einschränkt auf $\cdot : K \times L \to L$, kann man L als K-Vektorraum auffassen. Dies wird im Beweis des folgenden Satzes benutzt.

Satz 3.55. *Sei L ein endlicher Körper und $a \in L$. f sei das Minimalpolynom von a über K. Dann gilt: $|L| = |K|^{l \cdot \deg f}$ für eine natürliche Zahl l.*

Beweis. Der Satz ergibt sich aus den Aussagen von [Bos04], Abschnitt 3.2, Seite 89-92. Zunächst haben wir schon im Beweis von Satz 3.54 festgestellt, dass a und mit dem gleichen Argument sogar L algebraisch über K ist[2]. Nun besagt der Gradsatz [Bos04], 3.2/2, dass $\dim_K(L) = \dim_{K(a)}(L) \dim_K K(a)$ ist. Dabei ist $K(a)$ der kleinste K und a enthaltende Körper. Nach Satz [Bos04], 3.2/6 ist $\dim_K(K(a)) = \deg f$. Da L endlich ist, muss auch $\dim_K L$ endlich sein. Damit folgt die Behauptung aus der folgenden Eigenschaft von endlichdimensionalen Vektorräumen: $L \simeq K^{\dim_K(L)}$. □

Wir betrachten nun die Abbildung

$$\varphi : \mathbb{Z} \to K,\ z \mapsto z \cdot 1 := \underbrace{1 + 1 + \ldots + 1}_{z \text{ Faktoren } 1 \in K}.$$

Man kann sich leicht überlegen, dass φ ein Ringhomomorphismus ist. Das Bild $\operatorname{im} \varphi$ von φ ist ein Unterring von K und nach dem Homomorphiesatz für Ringe (Satz 3.10) isomorph zu $\mathbb{Z}/\ker \varphi$. Da K nullteilerfrei ist, ist $\operatorname{im} \varphi$ sogar ein Integritätsring, nach Satz 3.26 ist $\ker \varphi$ also ein Primideal. Die Ideale in $\mathbb{Z}$ sind Mengen $n\mathbb{Z}$ (Satz 3.9), von denen aber nur die mit $n = p$ für eine Primzahl p der Definition nach Primideale sein können. Also ist $\mathbb{Z}/p\mathbb{Z} \simeq \operatorname{im} \varphi \subset K$, so dass $\mathbb{Z}_p$ als Unterkörper in K enthalten ist. Wir haben somit den wesentlichen Teil des folgenden Satzes gezeigt:

Satz 3.56. *Sei K ein endlicher Körper. Dann gibt es eine eindeutig bestimmte Primzahl p, so dass gilt: $p \cdot a := \underbrace{a + \ldots + a}_{p \text{ Summanden „}a\text{“}} = 0$ für alle $a \in K$. p heißt* Charakteristik

[2] L heißt algebraisch über K, wenn jedes $a \in L$ algebraisch über K ist.

des Körpers K. Der Körper $\mathbb{F}_p$ ist ein Unterkörper von K und wird Primkörper *von K genannt. Weiterhin gilt dann $(a \pm b)^p = a^p \pm b^p$ für alle $a, b \in K$.*

Beweis. Siehe [Bos04], Abschnitt 3.1, Seiten 87-89. Die Aussage $(a+b)^p = a^p + b^p$ lässt sich wie folgt begründen: Für $1 \leq i \leq p-1$ ist der Binomialkoeffizient

$$\binom{p}{i} = \frac{p(p-1)\cdots(p-i+1)}{1 \cdot 2 \cdot \cdots \cdot i}$$

einerseits ganzzahlig, andererseits hat er im Zähler mindestens p stehen, aber im Nenner keinen durch p teilbaren Faktor. Also ist $\binom{p}{i}$ für $1 \leq i \leq p-1$ ein Vielfaches von p. Nun folgt die Behauptung aus dem binomischen Satz:

$$(a+b)^p = \sum_{i=0}^{p} \binom{p}{i} a^i b^{p-i} = a^p + b^p. \quad \square$$

Wir setzen nun voraus, dass K die Charakteristik p hat, dass also $\mathbb{Z}_p$ der Primkörper von K ist und in K die Gleichung $p \cdot 1 = 0$ gilt. Die Abbildung

$$\sigma : K \to K, \quad a \mapsto a^p \tag{3.11}$$

wird *Frobenius-Homomorphismus* genannt. Die Abbildung σ ist tatsächlich ein Ringhomomorphismus, denn $\sigma(ab) = (ab)^p = a^p b^p = \sigma(a)\sigma(b)$, weil $(K, \cdot)$ kommutativ ist, und $\sigma(a+b) = \sigma(a) + \sigma(b)$ gilt nach Satz 3.56. Der Frobenius-Homomorphismus hat einige wichtige Eigenschaften. Beispielsweise ist er konstant genau auf dem Unterkörper $\mathbb{Z}_p$ von K. Das besagt gerade Satz 3.53. Weiterhin ist der Frobenius-Homomorphismus bijektiv, also ein Isomorphismus. Dies kommt daher, dass Ringhomomorphismen zwischen Körpern immer injektiv sind ([Bos04], Seite 38, Bemerkung 2.3/3) und injektive Abbildungen einer endlichen Menge in sich selbst automatisch surjektiv sind. Isomorphismen, die eine Menge in sich selbst abbilden, werden Automorphismen genannt, daher nennt man σ auch den Frobenius-Automorphismus. Schließlich bildet der Frobenius-Homomorphismus Nullstellen eines Polynoms wieder auf Nullstellen dieses Polynoms ab. Ist nämlich $f = \sum_{i=0}^{k} a_i X^i \in K[X]$ ein Polynom und $b \in K$ eine Nullstelle von f, dann gilt

$$f(\sigma(b)) = \sum_{i=0}^{k} a_i \sigma(b)^i = \sigma(\sum_{i=0}^{k} a_i b^i) = \sigma(f(b)) = \sigma(0) = 0.$$

Satz 3.57. *Sei q eine ganze Zahl. Es gibt genau dann einen endlichen Körper mit exakt q Elementen, wenn q eine Primzahlpotenz, also $q = p^k$ für eine Primzahl p und eine natürliche Zahl $k > 0$ ist. Der Körper mit q Elementen ist, falls er existiert, bis auf Isomorphie eindeutig bestimmt und wird mit $\mathbb{F}_q$ bezeichnet.*

Beweis. [Bos04], Theorem 3.8/2 oder [IR90], Kapitel 7. □

3.7.2. Quadratische Körpererweiterungen

Für den Frobenius-Test ist eine spezielle Art von endlichen Körpern grundlegend: die durch quadratische Körpererweiterung gebildeten endlichen Körper. Das sind Körper mit p^2 Elementen, die mit Hilfe von quadratischen Polynomen ausgehend von $\mathbb{Z}_p$ konstruiert werden.

Eine Vereinbarung bezüglich der Notation

Im Folgenden werden wir oft mit Elementen aus dem Ring $\mathbb{Z}_n[X]/(f)$ oder $\mathbb{Z}_p[X]/(f)$ zu tun haben, wobei $n > 1$ eine natürliche Zahl ist und f ein Polynom aus $\mathbb{Z}_n[X]$, beispielsweise $f = X^2 - bX - c$. Wenn wir Elemente dieses Rings benennen, müssten wir — streng genommen — Ausdrücke wie $X + 3 + (f)$ oder $X + 3 \bmod (f)$ oder $X + 3 \bmod (X^2 - bX - c)$ gebrauchen. Das ist sehr umständlich. Deshalb sei hiermit vereinbart, dass wir in solchen Fällen das Ideal, modulo dem gerechnet wird, weglassen und zur Verdeutlichung, dass wir modulo diesem Ideal rechnen, die Variable klein schreiben. Die kurze Schreibweise für die eben aufgezählten Ausdrücke ist also $x + 3$. Ein weiteres Beispiel ist $x^n = b - x$, was für $X^n \equiv b - X \bmod (f)$ steht.

Konstruktion

Wir wollen nun sehen, wie diese Konstruktion der quadratischen Körpererweiterung vonstatten geht. Sei dazu p eine Primzahl und $0 \leq b, c < p$. Wir betrachten das Polynom $f = X^2 - bX - c$. Wenn f irreduzibel ist über $\mathbb{Z}_p[X]$, ist (f) ein maximales Ideal in $\mathbb{Z}_p[X]$ (Satz 3.27). Demzufolge ist $\mathbb{Z}_p[X]/(f)$ ein Körper (Satz 3.26), und es gilt $\mathbb{F}_{p^2} \simeq \mathbb{Z}_p[X]/(f)$. Da dieser Körper entsteht, indem Polynome modulo f betrachtet werden und f den Grad zwei hat, kann man sich den Körper als die Menge der Polynome in $\mathbb{Z}_p[X]$ vom Grad < 2 vorstellen:

$$\mathbb{Z}_p[X]/(f) \simeq \{aX + b \mid a, b \in \mathbb{Z}_p\}.$$

Die Addition ist in diesem Körper die übliche Addition von Polynomen in $\mathbb{Z}_p[X]$, also koeffizientenweises Addieren mit anschließender Reduktion modulo p. Bei der Multiplikation von linearen Polynomen entsteht im Allgemeinen ein Polynom mit einem quadratischen Anteil. Dieser wird reduziert mittels der Äquivalenz

$$X^2 \equiv bX + c \mod (X^2 - bX - c).$$

Beispiel für $f = X^2 - 2X - 1$ und $p = 11$: $(3X + 2)(2X + 3) = 6X^2 + 13X + 6 \equiv 6(2X + 1) + 13X + 6 = 25X + 12 \bmod (X^2 - 2X - 1) \equiv 4X + 1 \bmod 11$. — Dieser Vorgang heißt quadratische Körpererweiterung, weil dabei mit Hilfe des quadratischen Polynoms f der Körper $\mathbb{F}_p$ zu $\mathbb{F}_{p^2}$ erweitert wird.

Darstellung des Frobenius-Homomorphismus

Aus einem Grund, der bald klar wird, wollen wir die folgende Abbildung betrachten: $\sigma : \mathbb{Z}_p[X] \to \mathbb{Z}_p[X], g \mapsto g(b - X)$, also die Abbildung, die jedem Polynom g das Polynom zuordnet, das entsteht, wenn man $b - X$ in g einsetzt. Der folgende Satz besagt, dass σ verträglich mit den Restklassen modulo $X^2 - bX - c$ ist.

Satz 3.58. *Sei $n > 1$ ein ganze Zahl, $0 \leq b, c < n$ und σ die Abbildung $\sigma : \mathbb{Z}_n[X] \to \mathbb{Z}_n[X], g \mapsto g(b - X)$. Dann gilt: Wenn $g_1 \equiv g_2 \bmod (X^2 - bX - c)$, ist auch $\sigma(g_1) \equiv \sigma(g_2) \bmod (X^2 - bX - c)$. Die Abbildung*

$$\begin{aligned} \bar{\sigma} : \mathbb{Z}_n[X]/(X^2 - bX - c) &\to \mathbb{Z}_n[X](X^2 - bX - c), \\ g \bmod (X^2 - bX - c) &\mapsto g(b - X) \bmod (X^2 - bX - c) \end{aligned}$$

ist also wohldefiniert.

Bevor wir den Satz beweisen, soll ausdrücklich festgehalten werden, dass der Satz weder voraussetzt, dass n eine ungerade Primzahl ist, noch dass $X^2 - bX - c$ irreduzibel ist.

Beweis. Seien $g_1, g_2 \in \mathbb{Z}_n[X]$ und $g_1 \equiv g_2 \bmod (X^2 - bX - c)$. Dann gibt es ein Polynom $q \in \mathbb{Z}_n[X]$, so dass $(X^2 - bX - c)q = g_1 - g_2$. In diese Gleichung setzen wir $b - X$ ein und erhalten

$$\begin{aligned} \sigma(g_1) - \sigma(g_2) &= ((b - X)^2 - b(b - X) - c)q(b - X) \\ &= (X^2 - bX - c)q(b - X), \end{aligned}$$

also gilt $X^2 - bX - c \mid \sigma(g_1) - \sigma(g_2)$, das heißt $\sigma(g_1) \equiv \sigma(g_2) \bmod (X^2 - bX - c)$. □

Ab jetzt wird anstelle von $\bar{\sigma}$ auch σ geschrieben. Welche Abbildung gemeint ist, ergibt sich aus dem Zusammenhang. Auf $\mathbb{Z}_n[X]/(X^2 - bX - c)$ lässt sich σ auch genauer angeben:

$$\sigma(aX + b) = a(b - X) + b = -aX + ab + b. \tag{3.12}$$

Wir kehren von beliebigen n und f zurück zum Fall der quadratischen Körpererweiterung, also zu $n = p$ prim und $f = X^2 - bX - c$ irreduzibel in $\mathbb{Z}_p[X]$. Das Polynom f hat in $\mathbb{Z}_p$ keine Nullstellen, weil f irreduzibel ist (Satz 3.33). Im Oberkörper $\mathbb{Z}_p[X]/(f)$ dagegen hat es Nullstellen, nämlich $X \bmod f$ und $b - X \bmod f$: $f(X) = f \equiv 0 \bmod f$ sowie $f(b - X) = (b - X)^2 - b(b - X) - c = X^2 - bX - c = f \equiv 0 \bmod f$. Weitere Nullstellen kann f nicht haben, weil es nur Grad zwei besitzt. Also muss der Frobenius-Homomorphismus, den wir für einen Moment mit σ' bezeichnen wollen, X auf $b - X$ abbilden. Damit ergibt sich: $\sigma'(aX + b) = a\sigma'(X) + b$, weil σ' ein Körper-Homomorphismus und auf $\mathbb{Z}_p$ konstant ist, sowie weiter $a\sigma'(X) + b = a(b - X) + b = -aX + ab + b$. Wegen (3.12) folgt nun:

Satz 3.59. *Sei p eine Primzahl und $X^2 - bX - c$ ein in $\mathbb{Z}_p[X]$ irreduzibles Polynom. Dann ist in $\mathbb{Z}_p[X]/(X^2 - bX - c)$ der Frobenius-Homomorphismus $a \mapsto a^p$ nichts anderes als das Einsetzen von $b - X$, also die Abbildung $\sigma : g \bmod (X^2 - bX - c) \mapsto g(b - X) \bmod (X^2 - bX - c)$.*

Quadratische Polynome

Bisher haben wir einfach vorausgesetzt, dass $f = X^2 - bX - c$ irreduzibel ist. Nun soll festgestellt werden, unter welchen Bedingungen das der Fall ist. Nach Satz 3.36 ist f genau dann irreduzibel, wenn f keine Nullstellen hat. Im Fall $p = 2$ ist also $X^2 + X + 1$ ein irreduzibles Polynom über $\mathbb{Z}_p[X]$. Im wichtigeren Fall $p > 2$ untersuchen wir die Nullstellen von $X^2 - bX - c$ wie folgt:

$$\begin{aligned} X^2 - bX - c &= 0, \\ \Leftrightarrow 4X^2 - 4bX + b^2 &= b^2 + 4c, \\ \Leftrightarrow (2X - 1)^2 &= b^2 + 4c. \end{aligned}$$

Falls $b^2 + 4c$ ein quadratischer Rest ist, etwa $a^2 \equiv b^2 + 4c \bmod p$ für ein $0 \leq a < p$, hat $X^2 - bX - c$ die Nullstellen $2^{-1}(a - 1)$ und $2^{-1}(-a - 1)$ und ist somit nicht irreduzibel. Demzufolge ist $\mathbb{Z}_p[X]/(X^2 - bX - c)$ genau dann ein Körper, wenn $b^2 + 4c$ ein quadratischer Nichtrest, also $\left(\frac{b^2+4c}{p}\right) = -1$ ist.

Die Elemente aus $\mathbb{Z}_p$ haben die linearen Polynome aus $\mathbb{Z}_p[X]$ als Minimalpolynome. Wenn $\left(\frac{b^2+4c}{p}\right) = -1$ ist, dann ist $X^2 - bX - c$ das Minimalpolynom von x und $b - x$ aus $\mathbb{Z}_p[X]/(X^2 - bX - c) \simeq \mathbb{F}_{p^2}$, weil $X^2 - bX - c$ irreduzibel über $\mathbb{Z}_p$ ist und x und $b - x$ Nullstellen von $X^2 - bX - c$ sind. Könnte es in $\mathbb{F}_{p^2}$ auch Elemente geben, die als Minimalpolynom nicht ein quadratisches Polynom, sondern eines höheren Grades haben? Nein, das ist durch Satz 3.55 ausgeschlossen. Wir stellen also fest:

Satz 3.60. *Sei $a \in \mathbb{F}_{p^2}$. Dann ist das Minimalpolynom f von a über $\mathbb{Z}_p$ ein Polynom mit $\deg(f) \leq 2$. Es gilt $\deg(f) = 2$ genau dann, wenn $a \in \mathbb{F}_{p^2} - \mathbb{Z}_p$.*

Was passiert, wenn $f = X^2 - bX - c$ reduzibel ist, etwa $f = (X - a_1)(X - a_2)$ und wir trotzdem $\mathbb{Z}_p[X]/(f)$ betrachten? Den Fall $a_1 = a_2 = a$ wollen wir nicht behandeln, sondern nur bemerken, dass dann $f = X^2 - 2aX + a^2$ und somit $b = 2a$ und $c = -a^2$ ist, woraus $b^2 + 4c = 0$ folgt. Im Fall $a_1 \neq a_2$ sind $X - a_1$ und $X - a_2$ teilerfremd, so dass wegen Satz 3.22 gilt: $(X - a_1) + (X - a_2) = \mathbb{Z}_p[X]$. Außerdem folgt aus $a_1 \neq a_2$, dass $(X - a_1) \cap (X - a_2) = ((X - a_1)(X - a_2))$. Daher kann der Chinesische Restsatz (Satz 3.11) wie folgt angewendet werden:

$$\begin{aligned}\mathbb{Z}_p[X]/((X - a_1)(X - a_2)) &= \mathbb{Z}_p[X]/((X - a_1) \cap (X - a_2)) \\ &\simeq \mathbb{Z}_p[X]/(X - a_1) \times \mathbb{Z}_p[X]/(X - a_2) \\ &\simeq \mathbb{Z}_p \times \mathbb{Z}_p.\end{aligned}$$

Zur späteren Verwendung zeigen wir noch die folgende Aussage.

Lemma 3.61. *Sei $p > 2$ eine Primzahl und $g \in \mathbb{Z}_p[X]$ ein Polynom. Dann ist modulo p die Anzahl aller Paare (b, c) mit $\left(\frac{b^2+4c}{p}\right) = -1$ und $g \equiv 0 \bmod (p, X^2 - bX - c)$ nicht größer als $l/2$, wobei l die Anzahl der Lösungen $y \in \mathbb{F}_{p^2}$ von $g(y) = 0$ ist.*

Beweis. Wir definieren eine Abbildung ψ von

$$A := \left\{y \in \mathbb{F}_{p^2} - \mathbb{Z}_p \mid g(y) = 0\right\}$$

in die Menge

$$B := \left\{(b, c) \mid 0 \leq b, c \leq p - 1, \left(\frac{b^2 + 4c}{p}\right) = -1, g \equiv 0 \bmod (p, X^2 - bX - c)\right\}.$$

Sei dazu ein $y \in A$ gegeben. Nach Satz 3.60 hat das Minimalpolynom f von y die Form $f = X^2 - bX - c$, wobei $0 \leq b, c \leq p - 1$. Es muss gelten $\left(\frac{b^2+4c}{p}\right) = -1$, sonst wäre

f nicht irreduzibel und somit kein Minimalpolynom. Da $g(y) = 0$ ist, gehört g zum Kern des Einsetzungshomomorphismus $\mathbb{Z}_p[X] \to \mathbb{F}_{p^2}, h \mapsto h(y)$, der nach Satz 3.54 von f erzeugt wird. Also gibt es ein Polynom $q \in \mathbb{Z}_p[X]$ mit $g = fg$. Es gilt also $g \equiv 0 \bmod (p, X^2 - bX - c)$. Damit ist ψ wohldefiniert, wenn wir nun $\psi(y) := (b, c)$ setzen.

Wir machen uns jetzt klar, dass jedes $(b, c) \in B$ unter ψ Bild von genau zwei Elementen y_1 und y_2 aus A ist, dass also ψ ein 2-1-Zuordnung zwischen A und B vermittelt. Damit ist dann die Behauptung gezeigt.

Sei $(b, c) \in B$ vorgegeben und $f = X^2 - bX - c$. Es ist $\mathbb{Z}_p[X]/(f)$ ein Körper, der isomorph zu $\mathbb{F}_{p^2}$ ist. Daher können wir $y_1 := X + (f)$ und $y_2 := b - X + (f) \in \mathbb{Z}_p[X]/(f)$ als Elemente von $\mathbb{F}_{p^2}$ betrachten. Wir überlegen uns zunächst, dass $\psi(y_1) = (b, c)$. Dies gilt, weil y_1 offensichtlich zu $\mathbb{F}_{p^2} - \mathbb{Z}_p$ gehört, weil $g(X) = g \equiv 0 \bmod (p, g)$ nach Definition von B gilt, so dass $f(y_1) = 0$ ist, und weil f das Minimalpolynom von y_1 ist, da y_1 Nullstelle von f und f irreduzibel und normiert ist.

Nun können wir $\psi(y_2) = (b, c)$ begründen. Weil

$$f(b - X) = (b - X)^2 - b(b - X) - c = f \tag{3.13}$$

ist, gilt $f(b - X) \equiv 0 \bmod (p, f)$. Also ist auch y_2 Nullstelle von f und f ist Minimalpolynom von y_2. Weil wir schon wissen, dass $\psi(y_1) = (b, c)$ ist, ergibt sich wie bei der Definition von ψ, dass es $q \in \mathbb{Z}_p[X]$ mit $g = fq$ gibt. Nun ist

$$g(b - X) = f(b - X)q(b - X) \overset{(3.13)}{=} fq(b - X) \equiv 0 \bmod (p, f),$$

also ist auch y_2 Nullstelle von g und es folgt $\psi(y_2) = (b, c)$. Da f als Polynom vom Grad 2 höchstens 2 Nullstellen haben kann, gilt $|\psi^{-1}(b, c)| \leq 2$ und es folgt $\psi^{-1}(b, c) = \{y_1, y_2\}$. □

3.8. Lucas-Folgen

Zur schnellen Berechnung von Potenzen modulo einem quadratischen Polynom benötigen wir die Lucas-Folgen. Deshalb werden diese nun definiert und bedeutsame Eigenschaften hergeleitet. Ein Teil davon wird auch in [Gra98] genannt, weil sie für die dortige Implementierung des starken quadratischen Frobenius-Tests genutzt werden (siehe Abschnitt 6.6). Der andere Teil ist im Wesentlichen [CP01, Abschnitt 3.5] entnommen, wobei auch [Mon92] und [Ble96] zu Rate gezogen wurden. Nur Gleichung

(3.26) findet sich in den angegebenen Quellen nicht, ist aber eine triviale Folge der dort behandelten Tatsachen.

Sei $n > 1$ eine ungerade ganze Zahl sowie $0 \leq b, c < n$ ebenfalls ganze Zahlen mit $\left(\frac{b^2+4c}{n}\right) = -1$. Abkürzend schreiben wir $\Delta = b^2 + 4c$. Wir werden nun im Ring $\mathbb{Z}_n[X]/(X^2 - bX - c)$ rechnen. Als Erstes halten fest, dass dort für $\delta := x - (b - x)$ gilt: $\delta^2 = 4x^2 - 4bx + b^2 = 4(bx + c) - 4bx + b^2 = \Delta$. Bezüglich der Parameter (b, c) sind die *Lucas-Folgen* definiert als:

$$U_m := U_m(b,c) := \frac{X^m - (b-X)^m}{X - (b-X)} \bmod (n, X^2 - bX - c), \tag{3.14}$$

$$V_m := V_m(b,c) := X^m + (b-X)^m \bmod (n, X^2 - bX - c). \tag{3.15}$$

(U_m) und (V_m) sind als Folgen von Polynom-Restklassen definiert. Tatsächlich jedoch handelt es sich im Folgen in $\mathbb{Z}_n$, wie man dem folgenden Satz entnehmen kann.

Satz 3.62. *Für die Lucas-Folgen gilt:*

$$\begin{aligned}
U_0(b,c) &= 0, \\
U_1(b,c) &= 1, \\
V_0(b,c) &= 2, \\
V_1(b,c) &= b, \\
U_m(b,c) &= bU_{m-1}(b,c) + cU_{m-2}(b,c), \ m \geq 2 \qquad (3.16) \\
V_m(b,c) &= bV_{m-1}(b,c) + cV_{m-2}(b,c), \ m \geq 2. \qquad (3.17)
\end{aligned}$$

Beweis. U_0, U_1, V_0 und V_1 erhält man, indem man $m = 0, 1$ in die Definition einsetzt. (3.16) ergibt sich aus

$$\begin{aligned}
\delta(bU_{m-1} + cU_{m-2}) &= b(X^{m-1} - (b-X)^{m-1}) + c(X^{m-2} - (b-X)^{m-2}) \\
&= (bX + c)X^{m-2} - (b(b-X) + c)(b-X)^{m-2} \\
&\equiv X^2 \cdot X^{m-2} - (b^2 - 2bX + X^2)(b-X)^{m-2} \\
&\equiv X^m - (b-X)^m \bmod X^2 - bX - c.
\end{aligned}$$

Vollkommen analog zeigt man (3.17). □

Eine Potenz x^t von X mod $(n, X^2 - bX - c)$ lässt sich darstellen als $a_1 x + a_0$. Der folgende Satz zeigt, wie sich die Koeffizienten a_1, a_0 aus den Gliedern der Lucas-Folgen berechnen lassen.

Satz 3.63. *Für $t \in \mathbb{N}$ gilt:*

$$x^t = U_t x + 2^{-1}(V_t - bU_t). \tag{3.18}$$

Beweis.

$$\begin{aligned} U_t X + \frac{V_t - bU_t}{2} &= \frac{V_t}{2} + \frac{U_t(X - (b - X))}{2} \\ &= \frac{X^t + (b-X)^t}{2} + \frac{X^t - (b-X)^t}{2}. \end{aligned}$$ □

Folgende Eigenschaft der Lucas-Folgen werden wir außerdem benötigen:

Satz 3.64. *Für $j, k, m \in \mathbb{N}$ gilt:*

$$U_{j+k} = \frac{U_j V_k + U_k V_j}{2} \tag{3.19}$$

$$V_{j+k} = \frac{V_j V_k + \Delta U_j U_k}{2} \tag{3.20}$$

$$U_{2j} = U_j V_j \tag{3.21}$$

$$V_{2j} = V_j^2 - 2(-c)^j \tag{3.22}$$

$$V_{j+k} = V_j V_k - (-c)^j V_{k-j}, \quad \text{für } j \le k \tag{3.23}$$

$$U_m = \Delta^{-1}(2V_{m+1} - bV_m) \tag{3.24}$$

Beweis. (3.19) ergibt sich aus

$$\begin{aligned} \delta(U_j V_k + U_k V_j) &= \Big((x^j - (b-x)^j\Big)\Big(x^k + (b-x)^k\Big) + \Big(x^k - (b-x)^k\Big)\Big(x^j + (b-x)^j\Big) \\ &= 2x^j x^k - 2(b-x)^j(b-x)^k = 2\delta U_{j+k}, \end{aligned}$$

(3.20) aus

$$\begin{aligned} V_j V_k + \Delta U_j U_k &= \Big(x^j + (b-x)^j\Big)\Big(x^k + (b-x)^k\Big) + \frac{\Delta}{\delta^2}\Big(x^j - (b-x)^j\Big)\Big(x^k - (b-x)^k\Big) \\ &= 2x^j x^k + 2(b-x)^j(b-x)^k = 2V_{j+k}. \end{aligned}$$

(3.21) folgt sofort aus (3.19), (3.22) wegen $V_0 = 2$ aus (3.23). Für (3.23) betrachten wir

$$\begin{aligned} V_j V_k &= \Big(x^j + (b-x)^j\Big)\Big(x^k + (b-x)^k\Big) \\ &= x^j x^k + x^j(b-x)^k + (b-x)^j x^k + (b-x)^k(b-x)^k \\ &= V_{j+k} + x^j(b-x)^k + (b-x)^j x^k, \end{aligned}$$

und

$$\begin{aligned} x^j(b-x)^k + (b-x)^j x^k &= x^j(b-x)^j\Big((b-x)^{k-j} + x^{k-j}\Big) \\ &= x^j(b-x)^j V_{k-j} \end{aligned}$$

sowie $x^j(b-x)^j = (bx-x^2)^j = (bx-x^2+x^2-bx-c)^j = (-c)^j$.

(3.24) folgt aus (3.20), wenn man $j = m$ und $k = 1$ setzt, sowie $U_1 = 1$ und $V_1 = b$ beachtet. □

Später wird es erforderlich sein, beliebige $V_j(b,c)$ zu bestimmen. Es wird sich herausstellen, dass dies im Fall $c = -1$ besonders schnell möglich ist. Deshalb werden wir $V_j(b,c)$ ermitteln, indem zunächst $V_m(B,-1)$ für geeignete B, m bestimmt wird, woraus anschließend $V_j(b,c)$ abgeleitet wird. Im Rest dieses Abschnittes werden die für diese Vorgehensweise benötigten Zusammenhänge hergeleitet. Abgesehen von der Verwendung von (3.26) ist diese Strategie [CP01, Abschnitt 3.5.3] entnommen.

Lemma 3.65. $V_m(cd, -d^2) = d^m V_m(c, -1)$.

Beweis. Wir rechnen nicht mit Polynom-Restklassen, sondern mit echten Polynomen, also Elementen von $\mathbb{Z}_n[X]$, die als Reste bei Polynomdivision entstanden. So gibt es nach Definition von $V_m(cd, -d^2)$ ein Polynom $q \in \mathbb{Z}_n[X]$, für das gilt

$$X^m + (cd - X)^m = q \cdot (X^2 - cdX + d^2) + V_m(cd, -d^2).$$

Daraus folgt durch Einsetzen von dX für X

$$(dX)^m + (cd - dX)^m = q(dX) \cdot ((dX)^2 - cd(dX) + d^2) + V_m(cd, -d^2),$$

denn das Einsetzen verändert $V_m(cd, -d^2)$ nicht, weil dies ein Polynom vom Grad ≤ 0 ist. Das heißt

$$d^m\Big(X^m + (c - X)^m\Big) = q(dX) \cdot d^2(X^2 - cX + 1) + V_m(cd, -d^2).$$

Also gilt modulo $X^2 - cX + 1$:

$$\begin{aligned} d^m V_m(c, -1) &\equiv d^m\Big(X^m + (c - X)^m\Big) \\ &\equiv V_m(cd, -d^2). \quad \square \end{aligned}$$

Lemma 3.66. $V_m(a, -d^2) = d^m V_m(ad^{-1}, -1)$.

Beweis. $V_m(a,-d^2) = V_m((ad^{-1})d,-d^2) = d^m V_m(ad^{-1},-1)$ nach Lemma 3.65. □

Lemma 3.67. $V_{2m}(b,-c) = V_m(b^2-2c,-c^2)$.

Beweis. Induktion nach m. Der Induktionsanfang $V_{2\cdot 0}(b,-c) = 2 = V_0(b^2-2c,-c^2)$ und $V_{2\cdot 1}(b,-c) = bV_{2\cdot 1-1}(b,-c) - cV_{2\cdot 1-2}(b,-c) = b\cdot b - c\cdot 2 = V_1(b^2-2c,-c^2)$ ergibt sich aus Satz 3.62. Für den Induktionsschritt sei $k>1$ und die Behauptung für $m<k$ bewiesen. Wir zeigen die Behauptung für $m=k$:

$$\begin{aligned} V_{2k}(b,-c) &= V_{2+(k-2)}(b,-c) \\ &\overset{(3.23)}{=} V_2(b,-c)V_{2k-2}(b,-c) - c^2 V_{2k-4}(b,-c) \\ &= (b^2-2c)V_{2k-2}(b,-c) - c^2 V_{2k-4}(b,-c) \\ &\overset{\text{IV}}{=} (b^2-2c)V_{k-1}(b^2-2c,-c^2) - c^2 V_{k-2}(b^2-2c,-c^2) \\ &\overset{(3.17)}{=} V_k(b^2-2c,-c^2). \quad \square \end{aligned}$$

Satz 3.68. *Sei c modulo n invertierbar und $B = (b^2+2c)c^{-1}$. Dann gilt:*

$$V_{2m}(b,c) = c^m V_m(B,-1). \tag{3.25}$$

Beweis. $V_{2m}(b,c) = V_m(b^2+2c,-c^2) = c^m V_m((b^2+2c)c^{-1},-1)$ nach Lemma 3.67 und Lemma 3.66. □

Mit (3.25) ist das Problem, $V_j(b,c)$ zu bestimmen, für gerade j auf die Bestimmung von $V_m(B,-1)$ und c^m zurückgeführt. Für ungerade j hilft folgender Satz:

Satz 3.69. *Sei b modulo n invertierbar. Dann gilt:*

$$V_{2m-1}(b,c) = b^{-1}(V_{2m}(b,c) - cV_{2m-2}(b,c)). \tag{3.26}$$

Beweis. (3.17) besagt $V_{2m}(b,c) = bV_{2m-1}(b,c) + cV_{2m-2}(b,c)$. Nach $V_{2m-1}(b,c)$ umgestellt ergibt dies die Behauptung. □

4. Algorithmische Hilfsmittel

Dieses Kapitel behandelt die algorithmischen Rahmenbedingungen für die Untersuchung von Primzahltests. Zuerst wird — zugeschnitten auf Primzahltests — kurz der Algorithmusbegriff und die Bedeutung von Zeiteffizienz diskutiert. Dabei wird erklärt, wie Addition ganzer Zahlen und auch die anderen Grundrechenarten realisiert werden können. Ausgehend von $\mathbb{Z}$ wird dann auch auf das Rechnen in $\mathbb{Z}_n$ und $\mathbb{Z}_n[X]/(X^2 - bX - c)$ eingegangen, was für den quadratischen Frobenius-Test eine wichtige Rolle spielt. Besonders entscheidend für die Laufzeit des Frobenius-Test ist das Potenzierungsproblem, das in Abschnitt 4.3 detailliert behandelt wird. Am Ende des Kapitels wird noch erwähnt, wie das Jacobi-Symbol berechnet und Quadratzahlen erkannt werden können.

4.1. Grundlagen

Ein Algorithmus ist ein Verfahren, mit dessen Hilfe ein Computer Probleme lösen kann. Unter einem Problem versteht man in diesem Zusammenhang die Aufgabe, ausgehend von einer vorgegebenen Datenmenge, der Eingabe, ein davon abhängiges Ergebnis, die Ausgabe, zu ermitteln. Ein solches Problem ist zum Beispiel die ganzzahlige Division mit Rest. Bei diesem Problem besteht die Eingabe aus einem eine Paar (a, b) ganzer Zahlen, wobei b nicht den Wert 0 annehmen darf. Die Aufgabe besteht darin, eine ganze Zahl q und eine natürliche Zahl r zu ermitteln, so dass $a = bq + r$ und $0 \leq r < b$ gilt. Das Paar (q, r) ist also die Ausgabe unseres Beispielproblems. Wichtig ist, dass das Problem aus einer unendlich großen Menge von Probleminstanzen besteht und ein und dasselbe Verfahren jede dieser Instanzen lösen kann. Die Aufgabe „teile 17 ganzzahlig durch 4“ ist also kein Problem in unserem Sinne, sondern nur eine Instanz des Problems „ganzzahlige Division mit Rest“. Ein Verfahren, dass einfach $(4, 1)$ ausgibt, liefert zwar

genau das Ergebnis der ganzzahligen Division von 17 durch 4, ist aber kein Algorithmus, weil es nur eine einzige Instanz des Problems löst.

Praktisch gewinnbringend ist ein Algorithmus nur dann, wenn er das Problem nicht nur löst, sondern es *effizient* löst. Damit ist gemeint, dass er die gestellte Aufgabe mit vertretbarem Aufwand an Zeit und Speicherplatz lösen kann. Beide Größen hängen von der Beschaffenheit der Eingabe ab. Zur Vereinfachung untersucht man die Abhängigkeit von der *Größe* der Eingabe. Die in dieser Arbeit betrachteten Algorithmen erhalten als Eingabe ganze Zahlen, die viel größer sein können als die Register eines Computers. Deshalb wird hier als Größe der Eingabe die Bitlänge der eingegebenen Zahlen zugrundegelegt.

Weniger der Speicherplatzbedarf, sondern viel mehr die Laufzeit ist bei den in dieser Arbeit behandelten Algorithmen das über die Qualität entscheidende Kriterium. Wir werden drei Maße für den Zeitbedarf von Algorithmen verwenden: Abhängig von der Größe der Eingabe wird

- die Zahl der Bitoperationen,
- die Zahl der Multiplikationen bzw. der Multiplikationen modulo eines von der Eingabe abhängigen Modulus oder
- die durchschnittliche Laufzeit bei Probeläufen auf real existierender Hardware

gemessen.

Laufzeit bei Probeläufen Die Bedeutung der zuletzt genannten Messgröße ist unmittelbar klar. Sie ist das realistischste Maß, hat aber den Nachteil, dass man den Algorithmus erst programmieren muss. Daher kann für ein und denselben Algorithmus das Maß „Laufzeit bei Probeläufen“ variieren, je nach konkreter Implementierung, verwendetem Compiler, Betriebssystem, Hardware und anderen Bedingungen. Außerdem kann man so nur einen endlichen Ausschnitt von Eingaben untersuchen. Um das prinzipielle Laufzeitverhalten von Algorithmen zu beschreiben, ist dieses Maß daher nur bedingt geeignet.

Bitoperationen Das Maß „Zahl der Bitoperationen“ abstrahiert von der Laufzeitumgebung des Algorithmus. Damit dennoch der Zeitbedarf des Algorithmus realistisch abgebildet wird, wird der Umgang von Computern mit großen ganzen Zahlen

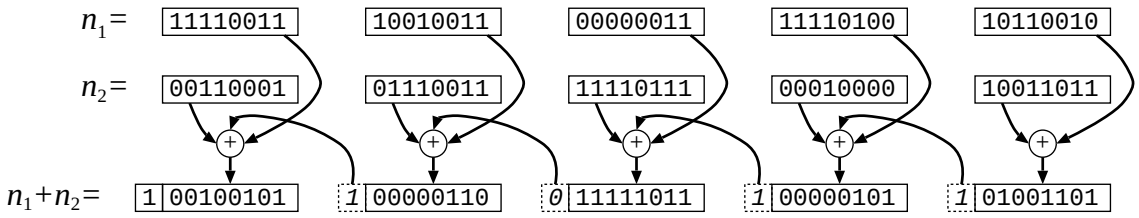

Abbildung 4.1.: Addition großer ganzer Zahlen n_1, n_2.

berücksichtigt. Direkt behandeln, das heißt in einem Schritt bzw. durch einen Maschinenbefehl, können Computer nämlich normalerweise nur Zahlen, die in einem Prozessorregister Platz finden. Dies sind typischerweise 16-, 32- oder 64-Bit-Zahlen. Möchte man mit größeren Zahlen rechnen, so muss man einen Algorithmus verwenden, der in mehreren Schritten die gewünschte Operation durchführt. Viele dieser Algorithmen ähneln den Verfahren, die man in der Schule zum schriftlichen Rechnen lernt. Als Beispiel wollen wir die Addition von Zahlen betrachten. In der Schule lernt man, dass man bei der schriftlichen Addition von rechts nach links jeweils eine Ziffer des ersten Summanden n_1 und des zweiten Summanden n_2 addieren muss, wobei unter Umständen ein Übertrag entsteht, der beim nächsten Ziffernpaar zu berücksichtigen ist. Ganz ähnlich verläuft die Addition großer ganzer Zahlen auf Computern. In Abbildung 4.1 ist zu sehen, dass die Binärdarstellung der Summanden in Teile zerlegt werden, die jeweils in ein Register passen. Diese Teile spielen nun die gleiche Rolle wie die Dezimalstellen bei der schriftlichen Addition von Hand: sie werden summiert, jeweils ein Teil von n_1 mit dem entsprechenden Teil von n_2, wobei gegebenenfalls ein Übertrag weitergeleitet und berücksichtigt wird. Man kann sich überlegen, dass auf diese Weise zur Addition zweier Zahlen n_1 und n_2, die beide nicht länger als l Bit sind, im Wesentlichen $\lceil l/k \rceil$ Additionen auf Maschinenbefehlsebene benötigt werden, wobei k die Größer der Prozessorregister in Bits ist. Hinzu kommen einige Operationen, die Daten zwischen Registern und dem Hauptspeicher transferieren. Die Anzahl dieser Operationen ist proportional zu $\lceil l/k \rceil$ mit einem kleinen Proportionalitätsfaktor. Setzt man $k = 1$, so erhält man die Anzahl der Operationen die nötig wären, wenn jeder Maschinenbefehl nur einzelne Bits manipulieren könnte. Dies ist die Zahl der Bitoperationen, die der Algorithmus ausführt.

Anhand der Anzahl der Bitoperationen ist nicht mehr zu erkennen, dass die Algorith-

men in der Realität beschleunigt werden, wenn man die Registergröße bei unveränderter Zeit pro Maschinenoperation steigert. Aber das ist — ähnlich wie der Effekt von schnelleren Prozessoren — eine Frage der Laufzeitumgebung des Algorithmus, von der das Maß für die Laufzeit gerade abstrahieren soll. Was die Anzahl der Bitoperationen zu einem guten Laufzeitmaß macht ist die Tatsache, dass die tatsächliche Laufzeit proportional zur Zahl der Bitoperationen ist.

Auch aus den in der Schule behandelten Verfahren zu schriftlichen Subtraktion, Multiplikation und Division kann man Algorithmen zum Rechnen mit großen ganzen Zahlen konstruieren. Auf einfache Weise ergibt sich der folgende

Satz 4.1 (Grundrechenarten in $\mathbb{Z}$). *Addieren, Subtrahieren und Vergleichen zweier ganzer Zahlen n_1, n_2 benötigt höchstens $O(\max\{\|n_1\|, \|n_2\|\})$ Bitoperationen. Zuweisung einer Zahl n an eine Variable benötigt $O(\|n\|)$ Bitoperationen. Multiplikation von n_1 und n_2 ist in $O(\|n_1\|\|n_2\|)$ Bitoperationen möglich, Division mit Rest von n_1 durch n_2 in $O(\|n_2\|(\|n_1\| - \|n_2\| + 1))$.*

Anzahl der Multiplikationen Dieses Maß wird am Ende des nächsten Abschnitts erläutert.

4.1.1. Grundrechenarten modulo n

Der Hauptteil der während eines Primzahltests anfallenden Berechnungen geschieht in $\mathbb{Z}_n$, also modulo einer ganzen Zahl $n > 1$. Deshalb ist zum Beispiel die Laufzeit einer modularen Addition wichtig. Als Eingabe der modularen Addition zählt der Modulus n sowie zwei Elemente aus $\mathbb{Z}_n$, repräsentiert durch ganze Zahlen a, b, wobei $0 \leq a, b < n$. Als Ausgabe soll $c, 0 \leq c < n$, mit $c \equiv a + b \bmod n$ berechnet werden. Dies kann recht laufzeiteffizient realisiert werden werden: Zunächst berechnet man $a + b$. Aufgrund der Voraussetzungen an a und b muss $0 \leq a + b < 2n$ gelten. Falls $a + b < n$ ist, stellt $a + b$ schon das Ergebnis der modularen Addition dar. Andernfalls ist $a + b - n$ dieses Ergebnis. Somit benötigt eine modulare Addition die Zeit von einer Addition, einem Vergleich und einer Subtraktion in $\mathbb{Z}$. Auf ähnliche Weise kann bei der modularen Subtraktion verfahren werden.

Die modulare Multiplikation ist aufwendiger. Für $a, b \in \mathbb{Z}_n$, repräsentiert durch a, b mit $0 \leq a, b < n$ soll $ab \in \mathbb{Z}_n$ berechnet werden. Dazu kann man zunächst $c := a \cdot b$

berechnen und anschließen $c \bmod n$. So wird eine Multiplikation und eine Division mit Rest benötigt.

Satz 4.2 (Addition und Multiplikation modulo n). *Addition und Subtraktion von $a, b \in \mathbb{Z}_n$ benötigt $O(\max\{\|a\|, \|b\|\}) = O(\|n\|)$ Bitoperationen. Das Ergebnis $ab \in \mathbb{Z}_n$ der modularen Multiplikation kann mit $O\big(\|a\|\|b\| + \|n\|(\|ab\| - \|n\| + 1)\big) = O(\|n\|^2)$ Bitoperationen berechnet werden.*

Schließlich ist es auch gelegentlich nötig, zu $a, 0 < a < n$, das Inverse modulo n auszurechnen, also b mit $0 < b < n$ und $ab \equiv 1 \bmod n$. Satz 3.24 zeigt, dass das Inverse genau dann existiert, wenn der $\mathrm{ggT}(a, n) = 1$ ist. Diese Bedingung kann mit dem berühmten Euklidischen Algorithmus überprüft werden, denn dieser bestimmt den größten gemeinsamen Teiler zweier Zahlen. Wenn feststeht, dass das $\mathrm{ggT}(a, n) = 1$, dann liefert Satz 3.23 den Ansatzpunkt, wie das Inverse b gefunden werden kann: Der Satz garantiert die Existenz von ganzen Zahlen x, y, so dass $xa + yn = 1$. Setzen wir $b := x \bmod n$, so gilt $ab = a(x + ln) = 1 - yn + aln \equiv 1 \bmod n$. Das einzige Problem ist nun, x und y zu bestimmen. Diese Aufgabe kann in den Euklidischen Algorithmus integriert werden, so dass der sogenannte erweitere Euklidische Algorithmus entsteht (Algorithmus 2). Der Algorithmus ist korrekt, weil — wie man sich überlegen kann

Algorithmus 2 Der erweiterte Euklidische Algorithmus.

Require: n, a, b mit $n > 1$, $b > 0$

1: $(\mathtt{a}, \mathtt{b}, \mathtt{xa}, \mathtt{ya}, \mathtt{xb}, \mathtt{yb}) \leftarrow (a, b, 1, 0, 0, 1)$
2: **while** $\mathtt{a} \bmod \mathtt{b} \neq 0$ **do**
3: $\quad \mathtt{q} \leftarrow \mathtt{a} \operatorname{div} \mathtt{b}$
4: $\quad (\mathtt{a}, \mathtt{b}, \mathtt{xa}, \mathtt{ya}, \mathtt{xb}, \mathtt{yb}) \leftarrow (\mathtt{b}, \mathtt{a} \bmod \mathtt{b}, \mathtt{xb}, \mathtt{yb}, \mathtt{xa} - \mathtt{q} \cdot \mathtt{xb}, \mathtt{ya} - \mathtt{q} \cdot \mathtt{yb})$
5: **end while**
6: **return** $(\mathtt{b}, \mathtt{xb}, \mathtt{yb})$

Ensure: $\mathtt{b} = \mathrm{ggT}(a, b)$ und $\mathtt{b} = \mathtt{xb} \cdot b + \mathtt{xa} \cdot a$.

— folgende Schleifeninvarianten gelten: $\mathrm{ggT}(\mathtt{a}, \mathtt{b}) = \mathrm{ggT}(a, b)$, $\mathtt{a} = \mathtt{xa} \cdot a + \mathtt{ya} \cdot b$ und $\mathtt{b} = \mathtt{xb} \cdot a + \mathtt{yb} \cdot b$. Man kann zeigen, dass die Laufzeit des erweiterten Euklidischen Algorithmus $O(\|a\|\|b\|) = O(\|n\|^2)$ ist (zum Beispiel [vzGG03, Theorem 3.13]). Damit folgt

Satz 4.3 (Bildung des Inversen modulo n). *Sei $n > 1$ eine ganze Zahl und $0 <$*

$a < n$. In $O(\|n\|^2)$ Bitoperationen kann das Inverse $b, 0 < b < n$ mit $ab \equiv 1 \bmod n$ berechnet werden oder festgestellt werden, dass a modulo n nicht invertierbar ist.

Schnellere Verfahren Mit erheblichem Aufwand kann die modulare Multiplikation beschleunigt werden [vzGG03]. Das liegt zum einen daran, dass die Schulmethode nicht das schnellste bekannte Verfahren zur Multiplikation ganzer Zahlen ist. Vielmehr ist es möglich, die Komplexität zur Multiplikation zweier Zahlen $n \geq m$ auf $O(\|n\| \cdot \log \|n\| \cdot \log\log \|n\|)$ zu senken. Weiterhin kann man daraus auch für die ganzzahlige Division mit Rest ein Verfahren ableiten, das nur um einen konstanten Faktor langsamer ist, also in der O-Notation die gleiche Laufzeit hat. Somit gilt:

Satz 4.4 (Schnelle Multiplikation modulo n). *Sei $n > 1$ eine ganze Zahl und $0 < a < n$, $0 < b < n$. Dann kann in $O(\|n\| \cdot \log \|n\| \cdot \log\log \|n\|)$ Bitoperationen die ganze Zahl c mit $0 < c < n$, $c \equiv ab \bmod n$ berechnet werden.*

Der erweiterte Euklidische Algorithmus kann ebenfalls beschleunigt werden, siehe beispielsweise [vzGG03]. Allerdings wird dabei nicht die gleiche Bitkomplexität erreicht wie bei der Multiplikation, so dass Invertierung modulo n asymptotisch etwas länger dauert als Multiplikation:

Satz 4.5 (Schnelle Bildung des Inversen modulo n). *Sei $n > 1$ eine ganze Zahl und $0 < a < n$. Dann kann mit $O(M(\|n\|) \log \|n\|)$ Bitoperationen b mit $0 < b < n, ab \equiv 1 \bmod n$ berechnet werden oder festgestellt werden, dass a modulo n nicht invertierbar ist. Dabei ist $M(l)$ die Anzahl der Bitoperationen, die für die Multiplikation zweier l-Bit Zahlen erforderlich ist. Es ergeben sich also $O(\|n\| \cdot (\log \|n\|)^2 \cdot \log\log \|n\|)$ Bitoperationen für eine Invertierung modulo n.*

Zahl der Multiplikationen als Komplexitätsmaß Häufig ist es unbequem, das Bitkostenmaß zu verwenden, weil die Bitkosten eines Algorithmus durch komplexe Ausdrücke beschrieben werden. Für zahlentheoretische Algorithmen werden stattdessen häufig nur die Anzahl der Multiplikationen gezählt. Additionen, Subtraktionen, Vergleiche und Zuweisungen werden ignoriert. Weil — wie wir gesehen haben — alle bekannten Algorithmen zur Multiplikation modulo n asymptotisch langsamer sind als die für Addition und Subtraktion, scheint das gerechtfertigt zu sein.

Problematisch ist hier allerdings die Frage, wie die Operationen, die nicht die gleiche Bitkomplexität haben wie modulare Multiplikation (ggT- und Inversen-Bildung, Be-

rechnung des Jacobi-Symbols), gewertet werden sollen. Das Problem ist, dass je nach Implementierung diese Frage anders zu beantworten ist. Ist beispielsweise die modulare Multiplikation nach der Schulmethode implementiert, also mit $O(\|n\|^2)$ Bitoperationen, dann benötigt eine Invertierung die Zeit von $O(1)$ Multiplikationen[1]. Werden dagegen die komplizierten $O(\|n\| \log \|n\| \log\log \|n\|)$-Verfahren für die Multiplikation genutzt, dann benötigt eine Invertierung die gleiche Zahl von Bitoperationen wie $\Theta(\log \|n\|)$ Multiplikationen. Dieses Problem ist durchaus praktisch relevant, weil für kleine Zahlen sinnvollerweise die Schulmethode verwendet wird, aber ab einer gewissen Bitlänge asymptotisch schnellere Verfahren (siehe Abschnitt 7.1.1).

Ein experimenteller Vergleich der Laufzeiten von Multiplikation und Invertierung findet sich in Abbildung 4.2. Die Experimente wurden dabei in der gleichen Umgebung durchgeführt wie auch die Experimente mit den Primzahltests (vgl. Kapitel 7). Gemessen wurden Multiplikation in $\mathbb{Z}$ (in Abbildung 4.2 bezeichnet mit „mul“), Multiplikation in $\mathbb{Z}_n$ (bezeichnet mit „mulmod“) und Invertierung in $\mathbb{Z}_n$ (bezeichnet mit „invmod“). Abhängig von der Bitlänge der untersuchten Zahlen, die auf der Abszisse abgetragen ist, wird die pro Operation benötigte Zeit dargestellt. Dazu gehört die Achseneinteilung am linken Rand. Die Beschriftung am rechten Rand gibt den Maßstab für die mit „invmod/mul“ und „invmod/mulmod“ bezeichneten Graphen an. Diese stellen die Quotienten aus für Invertierung (stets in $\mathbb{Z}_n$) und für Multiplikation (in $\mathbb{Z}$ bzw. in $\mathbb{Z}_n$) benötigter Zeit dar.

4.2. Rechnen in Ringen von Polynom-Restklassen

Endliche Körper sind Ringe der Form $\mathbb{Z}_n[X]/(f)$ für ein Polynom $f \in \mathbb{Z}_n[X]$. Der Frobenius-Test rechnet in solchen Ringen, wobei f ein quadratisches normiertes Polynom ist. Deshalb wird in diesem Abschnitt die Komplexität der Addition, Subtraktion, Multiplikation und der Bestimmung von Inversen in solchen Ringen untersucht. Sei dazu $n \geq 2$ eine ganze Zahl und $0 \leq b, c < n$. Wir untersuchen die Komplexität von Operationen in $R := R(n, b, c) := \mathbb{Z}_n[X]/(X^2 - bX - c)$.

Elemente von R sind Restklassen modulo $X^2 - bX - c$, die unendlich viele Elemente enthalten. Deshalb müssen sie für die Manipulation in Algorithmen geeignet repräsentiert werden. Dies geschieht vollkommen analog zu den Restklassen $k + n\mathbb{Z}$

[1] Dies wird in [Gra98, §3] bei der Definition des „selfridge“ als Kostenmaß für Primzahltests angenommen.

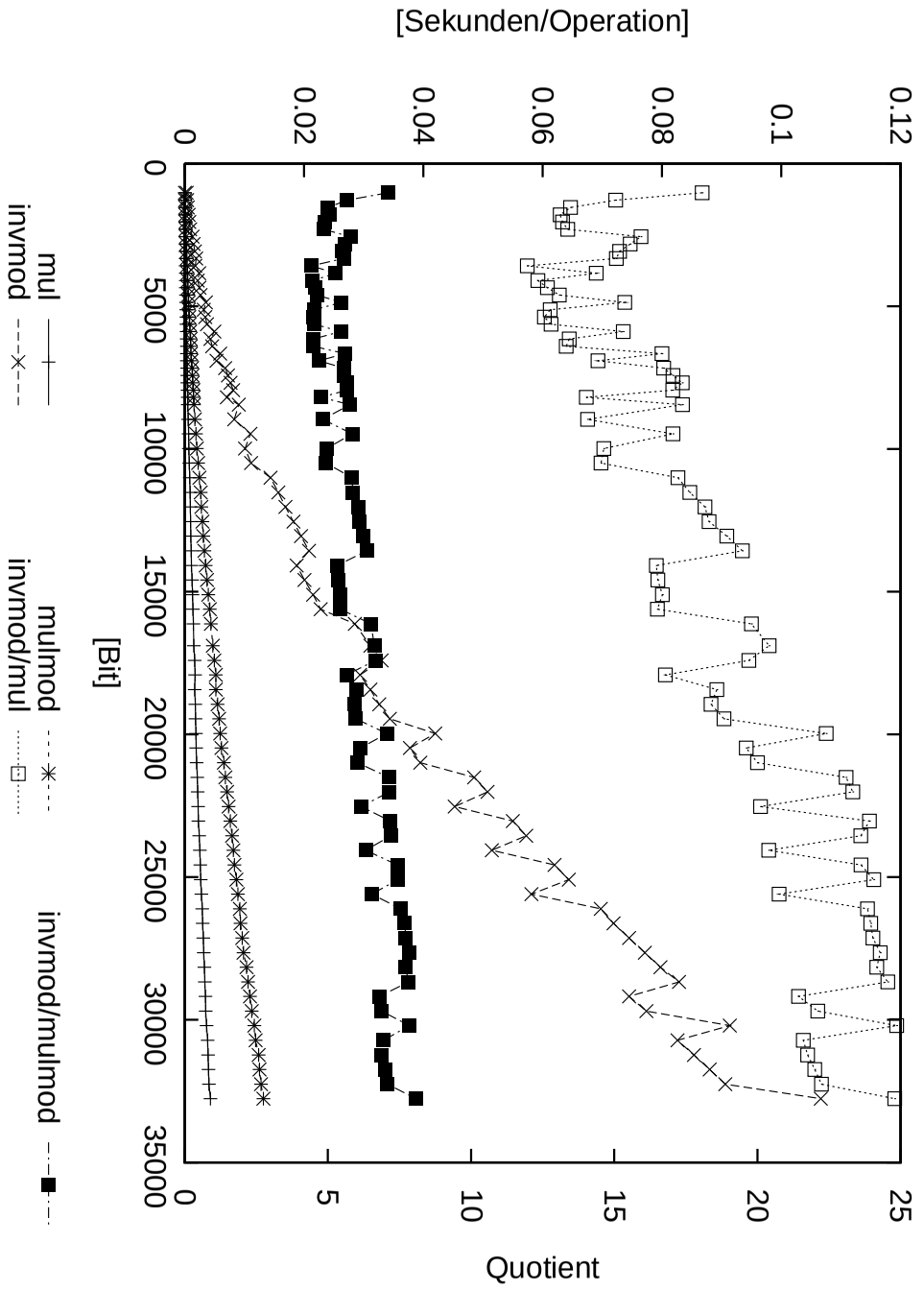

Abbildung 4.2.: Laufzeit von Multiplikation und Invertierung im experimentellen Vergleich.

von $\mathbb{Z}/n\mathbb{Z}$ — diese sind jeweils durch ihren Repräsentanten k mit $0 \leq k < n$ dargestellt. Analog stellen wir die Restklasse $g + (X^2 - bX - c)$ durch g mit $\deg g < \deg(X^2 - bX - c) = 2$ dar.

Seien nun $u = u_1X + u_0, v = v_1X + v_0 \in R$ gegeben. Summe und Differenz sind dann $u \pm v = u_1X + u_0 \pm (v_1X + v_0) = (u_1 \pm v_1)X + (u_0 \pm v_0)$. Dies ist wieder ein Repräsentant. Also ergibt sich sofort:

Satz 4.6 (Addition und Subtraktion modulo X^2-bX-c). *Addition bzw. Subtraktion in $R(n, b, c) = \mathbb{Z}_n[X]/(X^2 - bX - c)$ benötigt zwei Additionen bzw. Subtraktionen modulo n, also $O(\|n\|)$ Bitoperationen.*

Für das Produkt uv gilt:

$$\begin{aligned} uv &= (u_1X + u_0)(v_1X + v_0) \\ &= u_1v_1X^2 + (u_1v_0 + u_0v_1)X + u_0v_0 \\ &\equiv u_1v_1(bX + c) + (u_1v_0 + u_0v_1)X + u_0v_0 \\ &= (u_1v_1b + u_1v_0 + u_0v_1)X + (u_1v_1c + u_0v_0) \end{aligned} \tag{4.1}$$

Wendet man (4.1) direkt als Berechnungsvorschrift für das Produkt uv an, so ergibt sich ein Algorithmus, der 6 Multiplikationen und 3 Additionen modulo n benötigt. Weil Multiplikationen viel teurer sind als Additionen, ist man bestrebt, mit weniger Multiplikationen auszukommen. In [Gra98] wird deshalb gezeigt:

Satz 4.7 (Multiplikation modulo $X^2 - bX - c$). *Multiplikation in $R(n, b, c) = \mathbb{Z}_n[X]/(X^2 - bX - c)$ ist möglich mittels 5 Multiplikationen modulo n und 6 Additionen bzw. Subtraktionen modulo n.*

Beweis. Die fünf Multiplikationen sind u_1v_1, u_0v_0, $(u_1v_1)b$, $(u_1v_1)c$ und $(u_1+u_0)(v_1+v_0)$. Dann kann der Koeffizient vor X^0 in (4.1) durch eine Addition berechnet werden und der Koeffizient vor X durch $u_1v_1b + (u_1 + u_0)(v_1 + v_0) - u_1v_1 - v_1v_0$. □

Als Vorletztes wollen wir die Kosten für das Bilden von multiplikativen Inversen in R bestimmen. Sei $u = u_1X + u_0$ gegeben. Wir suchen $v = v_1X + v_0 \in R$ mit $uv = 1$. Das heißt nach (4.1), dass wir Lösungen $v_1, v_0 \in \mathbb{Z}_n$ suchen für das Gleichungssystem

$$(u_1b + u_0)v_1 + u_1v_0 = 0 \tag{4.2}$$

$$u_1cv_1 + u_0v_0 = 1. \tag{4.3}$$

Falls $u_1 = 0$, folgt $v_1 = 0$ und $v_0 \equiv u_0^{-1} \bmod n$, falls u_0 invertierbar ist bzw. es gibt kein Inverses zu u, falls u_0 modulo n nicht invertierbar ist. Betrachten wir nun den Fall $u_1 \neq 0$. (4.2) ist äquivalent zu

$$v_0 = -u_1^{-1}(u_1 b + u_0)v_1. \tag{4.4}$$

In (4.3) eingesetzt ergibt dies $(u_1 c - u_0 b - u_1^{-1}u_0^2)v_1 = 1$ also

$$u_1^{-1} \underbrace{(u_1^2 c - u_1 u_0 b - u_0^2)}_{=:T} v_1 = 1.$$

Folglich hat u kein Inverses in R, wenn T modulo n nicht invertierbar ist. Falls T dagegen invertierbar ist, gilt $v_1 = u_1 T^{-1}$. Dies in (4.4) eingesetzt ergibt $v_0 = -(u_1 b + u_0)T^{-1}$. Damit ist v bestimmt und der folgende Satz bewiesen:

Satz 4.8 (Invertierung modulo $X^2 - bX - c$). *Multiplikative Inverse können in $R(n, b, c) = \mathbb{Z}_n[X]/(X^2 - bX - c)$ mittels einer Invertierung modulo n, 7 Multiplikationen modulo n und 4 Additionen bzw. Subtraktionen modulo n bestimmt werden.*

Die Operation, die in Satz 3.58 definiert wurde, das Einsetzen von $b - X$, ist nicht so bekannt ist wie die Grundrechenarten. Für die Laufzeit des Frobenius-Test spielt sie jedoch eine wichtig Rolle. Deshalb stellen wir nun noch die Kosten einer σ-Operation modulo $(n, X^2 - bX - c)$ fest.

Satz 4.9. *Ist $f = a_1 X + a_0 \bmod (X^2 + bX - c) \in \mathbb{Z}_n[X]/(X^2 - bX - c)$ gegeben, so kann die Abbildung $\sigma : f \bmod (X^2 - bX - c) \mapsto f(b-X) \bmod (X^2 - bX - c)$ mit einer Multiplikation, einer Addition und einer Subtraktion modulo n berechnet werden.*

Beweis.

$$f(b - X) = a_1(b - X) + a_0 = -a_1 X + a_1 b + a_0. \quad \square \tag{4.5}$$

4.3. Potenzieren

Die Laufzeit sowohl des Fermat- und des Miller-Rabin- als auch des Frobenius-Tests werden wesentlich davon geprägt, eine oder mehrere Potenzen auszurechnen. Deshalb soll in diesem Abschnitt das Potenzierungsproblem genau betrachtet werden. Darunter wollen wir Folgendes verstehen: Es ist eine Menge M und eine „Multiplikation“ $\circ : M \times M \to M$ gegeben, die assoziativ ist. Für eine elegante Darstellung ist es nützlich,

auch noch ein neutrales Element $e \in M$ mit $e \circ a = a \circ e = a$ für alle $a \in M$ vorauszusetzen. Der Potenzierungsalgorithmus erhält als Eingabe ein Element $a \in M$ und eine natürliche Zahl $k \geq 1$ und soll als Ausgabe das Element $a^k := \underbrace{a \circ \cdots \circ a}_{k \text{ Faktoren „}a\text{“}} \in M$ liefern. Beispielsweise könnte $M = \mathbb{Z}$ und $\circ$ die Multiplikation ganzer Zahlen sein. Dann entspricht das gerade definierte Problem dem Potenzierungsproblem für ganze Zahlen. Eine andere Möglichkeit ist, dass Potenzen von Elementen eines endlichen Körper $\mathbb{F}_{p^2}$, ausgerechnet werden sollen. Dann ist $M = \mathbb{F}_{p^2}$ und $\circ$ entspricht der Multiplikation von linearen Polynomen mit anschließender Reduktion modulo einem quadratischen Polynom. Da wir nicht wissen, um welche Struktur es sich bei $(M, \circ)$ handelt, werden wir die Laufzeit der verschiedenen Potenzierungsalgorithmen in $\circ$-Operationen, die wir auch „Multiplikationen“ nennen, messen.

Als Lösung des Potenzierungsproblems ungeeignet ist der Ansatz, die Rekursion

$$a^k = \begin{cases} a^{k-1} \circ a, & \text{falls } k \geq 1, \\ e, & \text{falls } k = 0 \end{cases} \tag{4.6}$$

in einen iterativen Algorithmus umzusetzen, also a in einer Schleife $k-1$ mal mit sich selbst zu multiplizieren. Denn hierbei werden $k - 1 = O(2^{\|k\|})$ Multiplikationen benötigt, also exponentiell viele.

Ganz anders verhält es sich mit folgender Rekursion:

$$a^k = \begin{cases} (a^{k/2})^2, & \text{falls } 2|k \text{ und } k \geq 1, \\ a^{k-1} \circ a, & \text{falls } 2 \not| k \text{ und } k \geq 1, \\ e, & \text{falls } k = 0. \end{cases}$$

Den Wert von a^{10} kann man demnach so ausrechnen: $a^{10} = (a^5)^2 = (a^4 \circ a)^2 = ((a^2)^2 \circ a)^2 =$

$$(((e \circ a)^2)^2 \circ a)^2. \tag{4.7}$$

Das ist eine Verbesserung im Vergleich zu dem Ansatz nach (4.6): Es werden statt 9 nur noch 4 Multiplikationen benötigt, nämlich 3 Quadrierungen und eine[2] sonstige Multiplikation. Bei größeren k wird der Unterschied noch deutlicher: $a^{1000} = (a^{500})^2 = ((a^{250})^2)^2 = \ldots =$

$$(((((((((e \circ a)^2 \circ a)^2 \circ a)^2 \circ a)^2 \circ a)^2)^2 \circ a)^2)^2)^2 \tag{4.8}$$

[2]Ohne zu rechnen steht a als Ergebnis von $e \circ a$ fest.

benötigt nur 14 Multiplikationen anstelle von 999. Vergleicht man die Binärdarstellung der Exponenten, $10 = (1010)_2$ und $1000 = (1111101000)_2$, mit den Termen (4.7) bzw. (4.8), so stellt man fest, dass man diese Terme erhält, indem man mit dem Term e beginnt und diesen Schritt für Schritt erweitert, während man die Binärdarstellung der Exponenten von links nach rechts durchläuft. Für jede 1 in der Binärdarstellung quadriert man den bisherigen Term und multipliziert das Ergebnis mit a. Für jede 0 wird der bisherige Term einfach nur quadriert. Diese Vorgehensweise führt zu Algorithmus 3.

Algorithmus 3 Schnelles Potenzieren.

Require: Eingabe: $a \in M$, $k \in \mathbb{N}, k \geq 1$,
$k = (1k_{l-1} \dots k_1 k_0)_2$ sei die Binärdarstellung von k.

```
1:  b ← a
2:  m ← l − 1
3:  while m ≥ 0 do
4:     b ← b ∘ b
5:     if k_m = 1 then
6:        b ← b ∘ a
7:     end if
8:     m ← m − 1
9:  end while
10: return b
```

Ensure: $\mathtt{b} = a^k$

Offensichtlich gilt also:

Satz 4.10 (Potenzieren). *Das allgemeine Potenzierungsproblem, a^k für ein $k \in \mathbb{N}, k \geq 1$, auszurechnen, wird durch schnelles Potenzieren (Algorithmus 3) mit $\|k\| - 1$ Quadrierungen und $\|k\|_1 - 1$ sonstigen Multiplikationen gelöst. Die Gesamtzahl aller Multiplikationen beträgt somit höchstens $2\|k\| - 2$.*

[Knu98] und [Coh96] folgend kann dieses Ergebnis noch verbessert werden. Das ist vor allem für sehr große Exponenten interessant, besonders dann, wenn Quadrieren schneller möglich ist als eine normale Multiplikation. Die Idee ist, zunächst den obige Ansatz zu verallgemeinern, indem man sich nicht mehr an der Binärdarstellung,

sondern an der 2^m-ären Darstellung, $m \geq 1$, orientiert. Es ergeben sich dann obere Schranken für die Zahl der Quadrierungen und sonstigen Multiplikationen, die bei geeigneter Wahl von m asymptotisch besser sind als die aus Satz 4.10.

Sei $k = (k_l k_{l-1} \dots k_1 k_0)_{2^m}$ die 2^m-äre Darstellung von k, also $k = \sum_{0 \leq i \leq l} k_i (2^m)^i$ wobei $0 \leq k_i < 2^m$ für $i = 0, 1, 2, \dots, l$ und $k_l \neq 0$. Die Potenz a^k lässt sich nun wie folgt zerlegen:

$$\begin{aligned} a^k &= a^{(\dots((k_l \cdot 2^m)+k_{l-1})\cdot 2^m + \dots k_1)\cdot 2^m + k_0} \\ &= (\dots((a^{k_l})^{2^m} \circ a^{k_{l-1}})^{2^m} \circ \dots a^{k_1})^{2^m} \circ a^{k_0}. \end{aligned} \tag{4.9}$$

Zur Berechnung der a^{k_i} bestimmt man zu Beginn für jedes $b \in \{3, 5, 7, \dots, 2^m - 1\}$ den Wert a^b. Dies erfordert eine Quadrierung und $2^{m-1} - 1$ sonstige Multiplikationen. Damit lässt sich für jeden Wert $b, 0 \leq b < 2^m$ die Potenz a^b berechnen. Man zerlegt nämlich $b = 2^r s$ in eine Zweierpotenz 2^r und eine ungerade Zahl s und bestimmt $a^b = (a^s)^{2^r}$ indem man a^s r mal quadriert. a^s ist einer der schon berechneten Werte $A := \{a^b \mid b = 3, 5, 7, \dots, 2^m - 1\}$. Bei der Berechnung von a^k müssen nun die Quadrierungen im Vergleich zu (4.9) noch etwas umverteilt werden:

$$a^k = ((\dots((a^{s_l})^{2^{r_l+m-r_{l-1}}} \circ a^{s_{l-1}})^{2^{r_{l-1}+m-r_{l-2}}} \circ \dots a^{s_1})^{2^{r_1+m-r_0}} \circ a^{s_0})^{2^{r_0}}, \tag{4.10}$$

wobei $2^{r_i} s_i := k_i$ für $i = 0, \dots, l$ die Zerlegung von k_i in eine Zweierpotenz und einer ungerade Zahl ist. Nach (4.10) lässt sich a^k folglich in insgesamt

$$\underbrace{ml + r_l}_{\text{für (4.10)}} + \quad \underbrace{1}_{\text{für } A} \quad \text{Quadrierungen und} \tag{4.11}$$

$$\underbrace{l}_{\text{für (4.10)}} + \underbrace{2^{m-1} - 1}_{\text{für } A} \quad \text{sonstigen Multiplikationen} \tag{4.12}$$

berechnen. Nun ist $l = \lfloor \log_{2^m} k \rfloor = \lfloor \log k / m \rfloor$. Es ergeben sich also höchstens[3]

$$\begin{aligned} \log k + \quad & m \quad && \text{Quadrierungen und} \\ \lfloor \log k/m \rfloor + \quad & 2^{m-1} - 1 \quad && \text{sonstige Multiplikationen.} \end{aligned}$$

Eine gute Wahl für m ist $m = \lfloor \frac{1}{2} \log\log k \rfloor$, oder allgemein $m = \lfloor \alpha \log\log k \rfloor$ mit $0 < \alpha < 1$. Dann ist $m > (\alpha - \varepsilon) \log\log k$ für $k \geq 2^{2^{1/\varepsilon}}$. Für solche k werden also höchstens

$$\begin{aligned} \log k + \quad & O(\log\log k) && \text{Quadrierungen und} \\ \frac{\log k}{(\alpha - \varepsilon)\log\log k} + \quad & \tfrac{1}{2}(\log k)^\alpha && \text{sonstige Multiplikationen} \end{aligned}$$

[3] $m\lfloor \log k/m \rfloor + r_l + 1 \leq \log k + (m-1) + 1$

benötigt, um a^k zu bestimmen. Wenn bekannt ist, dass die Bitlänge des Exponenten mindestens 1025 beträgt, dann ist $k \geq 2^{2^{1/\varepsilon}}$ mit $\varepsilon = 1/10$ erfüllt. Wir wählen $\alpha = 3/5$ und erhalten:

Satz 4.11. *Das allgemeine Potenzierungsproblem, a^k für $k \in \mathbb{N}, k \geq 1$, zu berechnen, lässt sich lösen mit $\log k + O(\log\log k)$ Quadrierungen und $\frac{2\log k}{\log\log k} + \frac{1}{2}(\log k)^{3/5}$ sonstigen Multiplikationen.*

Abschließend sei noch bemerkt, dass man diese Methode weiter verbessern kann, indem man die Binärdarstellung von links nach rechts durchgehend immer ein höchstens m Bit langes Fenster betrachtet, das mit einer 1 beginnt und endet (1-X-1-Fenster). Die Hoffnung dabei ist, dass sich zwischen den betrachteten Fenstern Lücken aus 0en ergeben und die Anzahl l' der betrachteten 1-X-1-Fenster kleiner ist als l. Die Zahl der Quadrierungen ändert sich nicht, während sich die Anzahl der sonstigen Multiplikationen auf $l' + 2^{m-1} - 1$ verringert.

4.4. Weitere Algorithmen

Die Primzahltests müssen außer den Grundrechenarten und dem Potenzieren noch die Berechnung des Jacobi-Symbols beherrschen. Weiterhin muss geklärt werden, wie Quadratzahlen schnell erkannt werden können, denn diese werden vom Frobenius-Test nicht als Eingabe akzeptiert.

4.4.1. Berechnung des Jacobi-Symbols

Hierbei hilft ganz entscheidend das quadratische Reziprozitätsgesetz (Satz 3.51). Wir können Algorithmus 4 verwenden. Es handelt sich dabei um Algorithmus 2.3.5 aus [CP01]. Dort wird die Bitkomplexität des Algorithmus mit $O((\log b)^2)$ Operationen angegeben, weil er auf ähnliche Weise analysiert werden kann wie der euklidische Algorithmus.

4.4.2. Quadratzahlen erkennen

Die Frage, ob n eine Quadratzahl ist, wird beantwortet, indem die Quadratwurzel $\lfloor\sqrt{n}\rfloor$ berechnet wird. Nach [vzGG03, Aufgabe 9.43] ist dies mit Newton-Iteration möglich durch $O(M(\log n))$ Bitoperationen, wobei $M(l)$ die Anzahl der Bitoperationen für die

Algorithmus 4 Bestimmung des Jacobi-Symbols.

Require: Ganze Zahlen a, b mit $b > 0$ ungerade.

Ensure: Die Ausgabe ist $\left(\frac{a}{b}\right)$.

1: $(\mathtt{a}, \mathtt{b}, \mathtt{s}) \leftarrow (a \bmod b, b, 1)$
2: **while** $\mathtt{a} \neq 0$ **do**
3: **while** $2|\mathtt{a}$ **do**
4: $\mathtt{a} \leftarrow \mathtt{a} \operatorname{div} 2$
5: **if** $\mathtt{b} \equiv 3 \bmod 8$ **or** $\mathtt{b} \equiv 5 \bmod 8$ **then**
6: $\mathtt{s} \leftarrow -\mathtt{s}$
7: **end if**
8: **end while**
9: **if** $\mathtt{a} \equiv \mathtt{b} \equiv 3 \bmod 4$ **then**
10: $\mathtt{s} \leftarrow -\mathtt{s}$
11: **end if**
12: $(\mathtt{a}, \mathtt{b}) \leftarrow (\mathtt{b} \bmod \mathtt{a}, \mathtt{a})$
13: **end while**
14: **if** $\mathtt{b} = 1$ **then**
15: **return** $\mathtt{s}$
16: **else**
17: **return** 0
18: **end if**

Multiplikation zweier l-Bit Zahlen ist. Für die Details sei auf [vzGG03, Kapitel 9] verwiesen.

5. Klassische Primzahltests

Die Bedeutung des quadratischen Frobenius-Tests kann besser beurteilt werden und die mathematischen Hintergründe dieses Tests sind besser zu verstehen, wenn man die Verfahren kennt, mit denen bisher in der Praxis Primzahlen von zusammengesetzten Zahlen unterschieden werden. Deshalb soll in diesem Kapitel kurz das Wichtigste über den Fermat-Test und den Miller-Rabin-Test zusammengefasst werden. Wir knüpfen dabei an das an, was in Kapitel 2, besonders in Abschnitt 2.2.1, gesagt wurde.

5.1. Der Fermat-Test

Viele Begriffe wurden definiert, die „eine mathematische Tatsache der Form (2.4)“ voraussetzen. Deshalb ist es höchste Zeit, ein Beispiel für eine solche mathematische Tatsache zu geben, nämlich

$$n > 3 \text{ ist Primzahl} \Rightarrow \text{für jedes } a \in \mathbb{N}, 2 \leq a \leq n-2, \text{ gilt } a^{n-1} \equiv 1 \bmod n, \qquad (5.1)$$

was aus dem Kleinen Satz von Fermat (Satz 3.45) folgt. Die Menge $A(n)$ von Seite 24 ist in diesem Fall offensichtlich

$$A(n) = \{2, 3, \ldots, n-2\},$$

die Eigenschaft $E(a, n)$ lautet

$$a^{n-1} \equiv 1 \bmod n.$$

Auf eine formal notwendige, aber unfruchtbare Definition von $A(n)$ und $E(a, n)$ für $n \leq 3$ verzichten wir. (Für solche n benötigen wir keine Primzahltests.) Stattdessen füllen wir zwei Begriffe aus Abschnitt 2.2.1 mit Leben: Weil (5.1) auf dem Kleinen Satz von Fermat beruht, wollen wir die durch (5.1) definierten E-Primzahlkandidaten, E-Pseudoprimzahlen und E-Zeugen entsprechend benennen: Eine natürlich Zahl $n > 3$ ist

also ein *Fermat-Primzahlkandidat* bezüglich a, $2 \leq a \leq n-2$, genau dann, wenn $a^{n-1} \equiv 1 \bmod n$ gilt. Jede zusammengesetzte Zahl, die Fermat-Primzahlkandidat bezüglich a ist, nennen wir *Fermat-Pseudoprimzahl* bezüglich a. Gilt für $n > 3$ und a, $2 \leq a \leq n-2$ die Aussage $a^{n-1} \not\equiv 1 \bmod n$, dann heißt a *Fermat-Zeuge* für n (vgl. die Definition von „E-Zeuge“ auf Seite 24).

Weiter erinnern wir uns an einige Tatsachen aus Kapitel 4, um uns zu vergewissern, dass $E(a, n)$ tatsächlich „algorithmisch leicht prüfbar“ ist, wie es in Abschnitt 2.2.1 gefordert wird. Laut Satz 4.10 können wir $a^{n-1} \bmod n$ mit $O(\log n)$ Multiplikationen modulo n bestimmen, von denen nach Satz 4.2 jede mit $O((\log n)^2)$ Bitoperationen erledigt werden kann. Insgesamt genügen also $O((\log n)^3)$ Bitoperationen, um zu entscheiden, ob eine Zahl n ein Primzahlkandidat bezüglich a ist. Der *Fermat-Test* (Algorithmus 5) erledigt genau diese Aufgabe.

Algorithmus 5 Der Fermat-Test.

Require: n, a. n ungerade, $n > 3$ und $2 \leq a \leq n-2$.

1: $\mathtt{b} \leftarrow a^{n-1} \bmod n$.
2: **if** $\mathtt{b} = 1$ **then**
3: **return** „n ist Fermat-Primzahlkandidat bezüglich a.“
4: **else**
5: **return** „a ist ein Fermat-Zeuge dafür, dass n zusammengesetzt ist.“
6: **end if**

Fermat-Primzahlkandidaten erkennen zu können ist praktisch, weil „die meisten Fermat-Primzahlkandidaten echte Primzahlen sind“ bzw. „Fermat-Pseudoprimzahlen selten sind im Vergleich zu den richtigen Primzahlen“. Das ist eine mathematisch nicht exakte Umschreibung der folgenden Tatsache [CP01, Theorem 3.3.2]:

Satz 5.1. *Sei $a \geq 2$ eine ganze Zahl und $f(x)$ die Anzahl der Fermat-Pseudoprimzahlen bezüglich a, die kleiner oder gleich x sind. Dann gilt:*

$$\lim_{x \to \infty} \frac{f(x)}{\pi(x)} = 0.$$

Dieser Fakt wird durch Beispiele bestätigt, sogar schon bei kleinen Zahlen: In Tabelle 5.1 kann man ablesen, zu welchen Ergebnissen der Fermat-Test mit $a = 2$ für die ungeraden Zahlen von 11 bis 37 kommt. Man sieht, dass in diesem Bereich jeder Fermat-Primzahlkandidat bezüglich 2 tatsächlich eine Primzahl ist. Es lässt sich nachrechnen, dass 341 die kleinste Fermat-Pseudoprimzahl bezüglich 2 ist.

n	11	13	15	17	19	21	23	25	27	29	31	33	35	37
$2^{n-1} \bmod n$	1	1	4	1	1	4	1	16	13	1	1	4	9	1

Tabelle 5.1.: Ergebnisse des Fermat-Test mit $a = 2$ bei kleinen Zahlen.

Deshalb ist es sinnvoll, die in Abschnitt 2.1 vorgegebene Strategie zum Finden von großen Primzahlen etwas zu modifizieren: Wenn man eine Zahl n gewählt hat, die keine kleinen Teiler besitzt, dann startet man den Fermat-Test mit Eingabe n und $a = 2$. Nur auf die Zahlen n, die danach noch als Primzahlen in Frage kommen, also auf Fermat-Primzahlkandidaten bezüglich 2, wird der eigentliche Primzahltest angewendet.

Man könnte versuchen, aus dem Fermat-Test einen Primzahltest im Sinne von Definition 2.1 zu erstellen, indem — wie Algorithmus 1 angibt — a nicht Teil der Eingabe ist, sondern vom Algorithmus zufällig gewählt wird. Dieser Ansatz ist aber nicht erfolgreich, weil es die sogenannten Carmichael-Zahlen gibt.

Definition 5.2. *Eine zusammengesetzte Zahl n, die $a^{n-1} \equiv 1 \bmod n$ für alle zu n teilerfremden $a \in \mathbb{Z}$ erfüllt, heißt* Carmichael-Zahl.

Es ist bekannt, dass es unendlich viele Carmichael-Zahlen gibt [AGP94]. Der Fermat-Test erkennt Carmichael-Zahlen nur dann als zusammengesetzt, wenn a mit $1 < \mathrm{ggT}(a, n) < n$ gewählt wird. Computerberechnungen ergeben, dass es Carmichael-Zahlen n gibt, für die der Anteil dieser a unter allen Zahlen von 2 bis $n - 2$ sehr klein ist. Dies ist in Tabelle 5.2 angedeutet. Deshalb ist zu erwarten, dass es nicht möglich ist, den Fermat-Test durch Randomisierung zu einem Monte-Carlo-Algorithmus mit einseitigem *beschränkten* Fehler zu machen.

5.2. Der Miller-Rabin-Test

Da sich (5.1) als nützlich erwiesen hat, bietet sich diese Aussage als Ausgangspunkt an, wenn wir nach einem Beispiel für eine „mathematische Tatsache der Form (2.4)“ suchen, mit der Algorithmus 1 tatsächlich einen Primzahltest ergibt. Deshalb beschäftigen wir

n	1024651 $19 \cdot 199 \cdot 271$	1152271 $43 \cdot 127 \cdot 211$	1193221 $31 \cdot 61 \cdot 631$
$\varphi(n)/n$	0.939	0.964	0.950
n	40987480801 $281 \cdot 4201 \cdot 34721$	40999665001 $1021 \cdot 3001 \cdot 13381$	4486865842801 $1999 \cdot 15319 \cdot 146521$
$\varphi(n)/n$	0.996	0.9986	0.9994
n	4494395030893 $3863 \cdot 23173 \cdot 50207$	7583173444791361 $9323 \cdot 419491 \cdot 1938977$	7592162901158401 $167809 \cdot 176641 \cdot 256129$
$\varphi(n)/n$	0.999678	0.9998898	0.99998

Tabelle 5.2.: Unter den in [Pin98] aufgelisteten Carmichael-Zahlen findet man schnell solche, die in einer randomisierten Variante des Fermat-Tests mit hoher Wahrscheinlichkeit nicht als zusammengesetzt erkannt werden würden.

uns mit der folgenden Aussage, die stärker als (5.1) ist.

$$\begin{aligned} & n > 3 \text{ ist Primzahl und } n-1 = 2^r s,\ s \text{ ungerade} \\ \Rightarrow \quad & \text{für jedes } a \in \mathbb{N}, 2 \le a \le n-2, \text{ gilt} \\ & \quad \text{entweder } a^s \equiv 1 \bmod n \\ & \quad \text{oder } a^{2^j s} \equiv -1 \bmod n \text{ für ein } j, 0 \le j \le r-1. \end{aligned} \tag{5.2}$$

Beweis von (5.2). Sei $p = n > 3$ eine Primzahl, $p-1$ zerlegt in $p-1 = 2^r s$, s ungerade, und $2 \le a \le p-2$. Wir wissen, dass $\mathbb{Z}_p$ ein Körper ist (Satz 3.52) und führen die Überlegungen in diesem durch.

Nach dem Kleinen Satz von Fermat (3.45) gilt in $\mathbb{Z}_p$ die Gleichung $a^{p-1} = 1$. Dies lässt sich wegen $p - 1 = 2^r s$ weiter zerlegen in

$$a^{p-1} = (a^s)^{2^r} = \underbrace{(((a^s)^2)^2 \ldots)^2}_{r \text{ Exponenten „2“}} = 1. \tag{5.3}$$

Falls nun $a^s = 1$ gilt, ist (5.2) schon gezeigt. Andernfalls können wir aufgrund von (5.3) den kleinsten Exponenten k, $1 \le k \le r$, mit $a^{2^k s} = 1$ wählen. Mit $j = k - 1$ ist also $a^{2^j s}$ eine Quadratwurzel von 1. Wegen Satz 3.8 folgt $a^{2^j s} = -1$ oder $a^{2^j s} = 1$. Letzteres würde der Minimalität von k widersprechen, also gilt Behauptung. □

Die durch (5.2) definierten E-Primzahlkandidaten bzw. -Pseudoprimzahlen bezüglich a nennen wir *Miller-Rabin-Primzahlkandidaten* bzw. *Miller-Rabin-Pseudoprimzahlen*

bezüglich a, denn der *Miller-Rabin-Test* (Algorithmus 6, [Mil76, Rab80, Mon80]) besteht gerade darin, (5.2) dem Schema von Algorithmus 1 entsprechend anzuwenden. Dass der Miller-Rabin-Test (2.2) erfüllt brauchen wir also nicht mehr zu begründen. Zum Nachweis, dass es sich — wie schon mehrfach erwähnt — um einen Primzahltest mit Irrtumswahrscheinlichkeit von höchstens 1/4 handelt, ist also nur noch zu zeigen, dass für jedes n der Anteil der *Miller-Rabin-Zeugen*[1] für n unter den ganzen Zahlen $\{2, 3, \ldots, n-2\}$ mindestens 3/4 ist. Das soll hier nicht bewiesen werden, es seit stattdessen auf [Rab80, Mon80] verwiesen. Leichter einzusehen ist im Übrigen, dass die Irrtumswahrscheinlichkeit des Miller-Rabin-Tests durch 1/2 beschränkt ist, was beispielsweise in [Die04, Kapitel 5] ausführlich hergeleitet wird.

Algorithmus 6 Der Miller-Rabin-Test.

Require: n. n ungerade, $n > 3$.

1: Wähle zufällig ein $\mathtt{a}$, $2 \leq \mathtt{a} \leq n-2$.
2: Zerlege $n-1$ in $n-1 = 2^{\mathtt{r}}\mathtt{s}$, wobei $\mathtt{s}$ ungerade ist.
3: $\mathtt{b} \leftarrow \mathtt{a}^{\mathtt{s}} \bmod n$.
4: **if** $\mathtt{b} = 1$ oder $\mathtt{b} = n-1$ **then**
5: **return** „n ist prim.“
6: **end if**
7: **for** $j = 1, 2, 3, \ldots, \mathtt{r}-1$ **do**
8: $\mathtt{b} \leftarrow \mathtt{b}^2 \bmod n$.
9: **if** $\mathtt{b} = 1$ **then**
10: **return** „n ist nicht prim.“
11: **else if** $\mathtt{b} = n-1$ **then**
12: **return** „n ist prim.“
13: **end if**
14: **end for**
15: **return** „n ist nicht prim.“

Bevor wir uns noch mit der Laufzeit der Miller-Rabin-Tests beschäftigen, sei erwähnt, dass der Miller-Rabin-Test in der englischsprachigen Literatur auch „strong probable

[1] Der Definition von „E-Zeuge“ auf Seite 24 entsprechend muss ein Miller-Rabin-Zeuge für $n > 3$, $n-1 = 2^r s$, s ungerade, eine ganze Zahl a, $2 \leq a \leq n-2$ sein, für die $a^s \equiv 1 \bmod n$ gilt oder für die es ein für ein j, $0 \leq j \leq r-1$, gibt, so dass die Aussage $a^{2^j s} \equiv -1 \bmod n$ gilt.

prime test“ genannt wird[2], der Fermat-Test auch „probable prime test“.

Laufzeit Die Zerlegung $n-1 = 2^r s, s$ ungerade, besteht im Wesentlichen darin, in der Binärdarstellung von $n-1$ von rechts nach der ersten 1 zu suchen, so dass dafür weniger Zeit benötigt wird als für eine Addition in $\mathbb{Z}$. Für die Laufzeit des Miller-Rabin-Tests entscheidend ist dagegen die Berechnung von $a^s \bmod n$ und die anschließende for-Schleife. Beide Berechnungen zusammen entsprechen der Berechnung von $a^{(n-1)/2}$ mit dem Unterschied, dass in der for-Schleife einige zusätzliche Vergleiche durchgeführt werden und dass die for-Schleife möglicherweise schon verlassen wird, bevor $a^{s2^{r-1}} = a^{(n-1)/2}$ erreicht ist. Die Vergleiche und sonstigen Operationen in der for-Schleife, die keine Quadrierungen sind, können vernachlässigt werden. Damit ergibt sich aus den Sätzen 4.2, 4.4, 4.10 und 4.11:

Satz 5.3. *Eine Iteration des Miller-Rabin-Tests mit Eingabe n benötigt im Wesentlichen die Zeit zur Bestimmung der Potenz $a^{(n-1)/2} \bmod n$. Das sind höchstens $2\log n$ Multiplikationen modulo n, wenn Algorithmus 3 angewendet wird (Satz 4.10) — bzw. $\log n + \frac{2\log n}{\log\log n} + \frac{1}{2}(\log n)^{3/5} + O(\log\log n)$, also $\log n + o(\log n)$ Multiplikationen modulo n, wenn man die Verbesserungen benutzt, die zu Satz 4.11 führen.*

[2]Genauer gesagt nicht Algorithmus 6, sondern ein deterministischer Algorithmus, der — wie Algorithmus 5 — n und a als Eingabe nimmt und entscheidet, ob n ein Miller-Rabin-Primzahlkandidat bezüglich a ist.

6. Der quadratische Frobenius-Test

Nun sind wir umfassend ausgerüstet, um uns mit dem quadratischen Frobenius-Test zu beschäftigen. Das Ziel dieses Kapitels ist, den randomisierten starken quadratischen Frobenius-Test (RQFT) vorzustellen, der ein Primzahltest mit Irrtumswahrscheinlichkeit kleiner als $\frac{1}{7710}$ ist und asymptotisch die Zeit von drei Miller-Rabin-Tests benötigt.

Zuerst werden wir in Abschnitt 6.1 die grundlegende Bedingung kennenlernen, die in jeder Variante des quadratischen Frobenius-Tests geprüft wird, um zusammengesetzte Zahlen zu erkennen. Dies führt zur Definition des „einfachen quadratischen Frobenius-Tests“ und zur Definition einiger Begriffe wie den „Frobenius-Pseudoprimzahlen“. Danach, in Abschnitt 6.2, wird eine Verschärfung des einfachen quadratischen Frobenius-Tests vorgestellt, der starke quadratische Frobenius-Test (QFT), der den Kern des RQFT bildet. Abschnitt 6.3 dient dazu, einen Satz über die Anzahl der für den QFT zulässigen Parameter (b, c) zu beweisen. Indem diese Parameter zufällig gewählt werden und der QFT aufgerufen wird, entsteht der RQFT, den wir dann in Abschnitt 6.4 definieren. Die aufwendigen Untersuchungen der Irrtumswahrscheinlichkeit und der Laufzeit des RQFT bilden dann den Gegenstand von Abschnitt 6.5 und 6.6. In letzterem werden wir nicht nur die ursprüngliche Implementierung des RQFT nach [Gra98] besprechen, sondern auch die Implementierung des einfachen quadratischen Frobenius-Tests, die in [CP01, Abschnitt 3.5.3] beschrieben wird, auf den RQFT übertragen, was für eine Verbesserung der Laufzeit des RQFT sorgen wird.

Bevor wir den quadratischen Frobenius-Test detailliert behandeln, soll noch erwähnt werden, dass der quadratische Frobenius-Test verallgemeinert werden kann. In [Gra01] sind der (allgemeine) Frobenius-Test und der (allgemeine) starke Frobenius-Test definiert. Beide Tests beruhen auf einer mathematischen Tatsache der Form (2.4), wobei die Menge $A(n)$ aus bestimmten normierten Polynomen beliebigen Grades besteht. Für Einzelheiten sei auf [Gra01] und [CP01, Abschnitt 3.5.5] verwiesen.

Eine weitere interessante Tatsache ist, dass der quadratische Frobenius-Test eine Verschärfung des sogenannten Lucas-Tests[1] darstellt. Der Lucas-Test wiederum ist deshalb interessant, weil er in Verbindung mit dem Fermat-Test anscheinend sehr effektiv ist: Bisher konnte niemand eine zusammengesetzte Zahl $n \equiv \pm 2 \bmod 5$ finden, die gleichzeitig eine Fermat-Pseudoprimzahl bezüglich 2 und eine Lucas-Pseudoprimzahl bezüglich $(1, 1)$ ist — und das, obwohl Experten der algorithmischen Zahlentheorie einen Preis von 620$ für das erste Beispiel bzw. einen Beweis, dass keines existiert, ausgelobt haben [CP01, Aufgabe 3.41].

6.1. Der einfache quadratische Frobenius-Test

Die Tests in Kapitel 5 nutzen die Verhältnisse in endlichen Körpern mit p Elementen, p prim (siehe Beweis von (5.2)). Der quadratische Frobenius-Test hingegen basiert auf endlichen Körpern mit p^2 Elementen, p prim. Aus Abschnitt 3.7.2 wissen wir, dass solche Körper entstehen, indem man eine Primzahl $p > 2$ wählt, sowie b, c mit $0 \le b, c < p$ und $\left(\frac{b^2+4c}{p}\right) = -1$. Dann ist nämlich $X^2 - bX - c$ in $\mathbb{Z}_p[X]$ ein irreduzibles Polynom und $\mathbb{Z}_p[X]/(X^2 - bX - c)$ ein Körper mit p^2 Elementen. Wir haben gesehen (Satz 3.59), dass in diesem Körper der Frobenius-Homomorphismus $a \mapsto a^p$ nichts anderes als das Einsetzen von $b - X$ ist. Deshalb gilt in $\mathbb{Z}_p[X]/(X^2 - bX - c)$ folgende Aussage[2]:

$$\boxed{x^p = b - x.} \tag{6.1}$$

Der quadratische Frobenius-Test besteht darin, diese Bedingung zu prüfen. Dazu formulieren wir nun wieder eine mathematische Tatsache der Form (2.4):

$$\begin{aligned} & n > 2 \text{ ist Primzahl} \\ \Rightarrow \quad & \text{für jedes Paar } (b, c),\ 0 \le b, c \le n-1, \text{ mit } \left(\frac{b^2+4c}{n}\right) = -1 \text{ gilt:} \\ & \quad X^n \equiv b - X \bmod (n, X^2 - bX - c). \end{aligned} \tag{6.2}$$

Wie in Kapitel 5 wenden wir die Definition 2.2 an, dieses Mal aber auf (6.2):

[1] Gemeint ist hier der Test, wie er beispielsweise in [CP01, Abschnitt 3.5] vorgestellt wird. Er sollte nicht verwechselt werden mit dem Lucas-Lehmer-Test, der Mersennesche Primzahlen findet, indem er *beweist*, dass gewisse $2^p - 1$ prim sind.

[2] Zu Notation beachte man die Vereinbarung auf Seite 52.

Definition 6.1. *Sei $n > 2$ eine ungerade natürliche Zahl, die keine Quadratzahl ist. Für das Paar (b, c) gelte $0 \leq b, c \leq n-1$ und $\left(\frac{b^2+4c}{n}\right) = -1$. Dann heißt n* Frobenius-Primzahlkandidat *bezüglich (b, c), wenn in $\mathbb{Z}_n[X]/(X^2 - bX - c)$ gilt: $x^n = b - x$. Gilt dagegen $x^n \neq b - x$, so heißt (b, c)* Frobenius-Zeuge *für n (vgl. die Definition von „E-Zeuge“ auf Seite 24). Ist n eine zusammengesetzte Zahl und ein Frobenius-Primzahlkandidat bezüglich (b, c), so heißt n auch* Frobenius-Pseudoprimzahl *bezüglich (b, c).*

Es hat einen Grund, warum in Definition 6.1 gerade Zahlen und Quadratzahlen ausgeschlossen wurden. Für gerade Zahlen n ist das Jacobi-Symbol $\left(\frac{-}{n}\right)$ nicht definiert. Aber gerade Zahlen stellen selbstverständlich kein Problem bei der Suche nach Primzahlen dar, weil sie in der Binärdarstellung an ihrem letzten Bit erkannt werden. Ist $n = k^2$ eine Quadratzahl, so kann das Jacobi-Symbol $\left(\frac{a}{n}\right) = \left(\frac{a}{k^2}\right) = \left(\frac{a}{k}\right) \cdot \left(\frac{a}{k}\right)$ niemals den Wert -1 annehmen. Es gibt dann also kein einziges Paar (b, c), dass als Frobenius-Zeuge für n in Frage kommt. Deshalb kann der quadratische Frobenius-Test nicht auf Quadratzahlen angewendet werden. Aber auch diese stellen bei der Suche nach Primzahlen keine Schwierigkeit dar (Abschnitt 4.4.2).

Wie aus (5.1) der Fermat-Test (Algorithmus 5) entstand, so ergibt sich aus (6.2) der *einfache quadratische Frobenius-Test* (Algorithmus 7). Dieser Test wird in [CP01, Abschnitt 3.5] als „quadratischer Frobenius-Test“ vorgestellt[3]. Dort wird auch gezeigt, wie man ihn so implementieren kann, dass er nicht mehr als die dreifache Laufzeit eines Fermat-Tests benötigt. Auf die dazu eingesetzte Technik wird in dieser Arbeit später noch eingegangen, weil wir sie für den starken quadratischen Frobenius-Test nutzen wollen (Abschnitt 6.6.1).

Algorithmus 7 Der einfache quadratische Frobenius-Test.

Require: n, b, c. $n > 2$ ungerade und keine Quadratzahl, $0 \leq b, c < n$, $\left(\frac{b^2+4c}{n}\right) = -1$.

1: **if** $X^n \equiv b - X \bmod (n, X^2 - bX - c)$ **then**
2: **return** „n ist Frobenius-Primzahlkandidat bezüglich (b, c).“
3: **else**
4: **return** „(b, c) ist Frobenius-Zeuge dafür, dass n keine Primzahl ist.“
5: **end if**

[3] Zunächst wird in [CP01, Definition 3.5.5] der Begriff „Frobenius pseudoprime“ definiert, wobei auch $\left(\frac{b^2+4c}{n}\right) = 1$ zugelassen wird. In [CP01, Algorithmus 3.5.9] wird dann der „Frobenius probable prime test“ angegeben.

Wir wollen jetzt noch eine Aussage über Frobenius-Primzahlkandidaten beweisen. Sie entspricht zum Teil [CP01, Übungsaufgabe 3.27]. Insbesondere folgt daraus, dass jede Frobenius-Pseudoprimzahl bezüglich (b, c) eine Fermat-Pseudoprimzahl bezüglich c ist — ein Spezialfall eines Resultats aus [Gra01].

Lemma 6.2. *Es gelte für n, b, c, n ungerade, $0 \leq b, c < n$, die Gleichung $X^n \equiv b - X \bmod (n, X^2 - bX - c)$. Dann folgt*

$$X^{n+1} \equiv -c \bmod (n, X^2 - bX - c) \tag{6.3}$$

$$(b - X)^n \equiv X \bmod (n, X^2 - bX - c) \tag{6.4}$$

$$c^{n-1} \equiv 1 \bmod n \tag{6.5}$$

$$X^{n^2-1} \equiv 1 \bmod (n, X^2 - bX - c) \tag{6.6}$$

Beweis. (6.3) ergibt sich so: $X^{n+1} = XX^n \equiv X(b - X) \equiv -c \bmod (b, X^2 - bX - c)$.

Nach Voraussetzung ist in $\mathbb{Z}_n[X]$ das Polynom $X^2 - bX - c$ ein Teiler von $X^n - (b - X)$. Es gibt also ein Polynom $q \in \mathbb{Z}_n[X]$ mit $q \cdot (X^2 - bX - c) = X^n - (b - X)$. Wir setzen auf beiden Seiten für X das Polynom $b - X$ ein. Damit folgt

$$q(b - X) \cdot ((b - X)^2 - b(b - X) - c) = (b - X)^n - (b - (b - X)).$$

Das vereinfacht sich zu

$$q(b - X) \cdot (X^2 - bX - c) = (b - X)^n - X.$$

Damit ist $(b - X)^n - X$ durch $X^2 - bX - c$ teilbar, es gilt also (6.4). Weiter gilt

$$\begin{aligned}(b - X)^n \equiv (X^n)^n = X^{n^2} = XX^{(n+1)(n-1)} &\equiv X(-c)^{n-1} \\ &= Xc^{n-1} \bmod (n, X^2 - bX - c).\end{aligned} \tag{6.7}$$

Mit (6.4) erhalten wir also $X \equiv Xc^{n-1} \bmod (n, X^2 - bX - c)$, woraus $1 \equiv c^{n-1} \bmod n$ folgt. Somit gilt (6.5). Damit haben wir aber auch schon (6.6) gezeigt, wie sich aus der Rechnung (6.7) ergibt. □

6.2. Der starke quadratische Frobenius-Test (QFT)

Der einfache quadratische Frobenius-Test wird für ein gutes Verfahren zum Erkennen von Primzahlen gehalten [CP01, Abschnitt 3.5]. Tatsächlich basiert der Algorithmus

„RQFT“ aus [Gra98], der ein Primzahltest mit Irrtumswahrscheinlichkeit $< 1/7710$ ist und asymptotisch die Zeit von drei Miller-Rabin-Tests benötigt, zu einem wesentlichen Teil auf dem einfachen quadratischen Frobenius-Test. Um die angegebene Irrtumswahrscheinlichkeit nachweisen zu können, müssen an Algorithmus 7 jedoch einige Verstärkungen vorgenommen werden. Das Ergebnis ist Algorithmus 8. Er stammt aus [Gra98], wird dort als „quadratischer Frobenius-Test“ bezeichnet und mit QFT abgekürzt. In dieser Arbeit soll er der *starke quadratische Frobenius-Test* genannt werden. Das passt besser zu der Begriffsbildung, die der Autor von [Gra98] in [Gra01] selbst vornimmt. Es unterstreicht die Tatsache, dass der starke quadratische Frobenius-Test über den einfachen quadratischen Frobenius-Test im Wesentlichen dadurch hinausgeht, dass er nach Quadratwurzeln von 1 sucht. Das ist auch der Unterschied, der den Miller-Rabin-Test (englisch oft „strong probable prime test“) gegenüber dem Fermat-Test (englisch oft „probable prime test“) auszeichnet. Die Abkürzung „*QFT*“ steht in dieser Arbeit durchweg für den *starken* quadratischen Frobenius-Test, also Algorithmus 8.

Algorithmus 8 Der starke quadratische Frobenius-Test (QFT) aus [Gra98].

Require: n, b, c. $n > 2$ ungerade und keine Quadratzahl, $0 \leq b, c \leq n-1$, $\left(\frac{b^2+4c}{n}\right) = -1$, $\left(\frac{-c}{n}\right) = 1$.

1: *Alle Berechnungen geschehen modulo* $(n, X^2 - bX - c)$.
2: **if** $x^{\frac{n+1}{2}} \notin \mathbb{Z}_n$ **then**
3: **return** „(b, c) ist ein QFT-Zeuge dafür, dass n zusammengesetzt ist.“
4: **else if** $x^{n+1} \neq -c$ **then**
5: **return** „(b, c) ist ein QFT-Zeuge dafür, dass n zusammengesetzt ist.“
6: **else**
7: Zerlege $n^2 - 1$ in $n^2 - 1 = 2^r s$, s ungerade.
8: **if** $x^s \neq 1$ und $x^s, x^{2s}, x^{2^2 s}, \ldots, x^{2^{r-2} s} \neq -1$ **then**
9: **return** „(b, c) ist ein QFT-Zeuge dafür, dass n zusammengesetzt ist.“
10: **else**
11: **return** „n ist QFT-Primzahlkandidat bezüglich (b, c).“
12: **end if**
13: **end if**

Wir wollen nun nachvollziehen, dass der QFT stets die Ausgabe „n ist Primzahlkandidat bezüglich (b, c)“ liefert, wenn er eine Primzahl $n > 2$ und den Anforderungen entsprechende Parameter b, c als Eingabe erhält. Seien dazu n, b, c den Vorbedingungen

von Algorithmus 8 entsprechende Eingabeparameter und $n = p$ eine Primzahl.

Wir beginnen mit Zeile 4. Dort wird einfach nur die Bedingung (6.1) geprüft. Es ist nämlich — unabhängig davon, ob n prim ist oder nicht — $x^n = b - x$ äquivalent zu $x^{n+1} = bx - x^2 = x^2 - bx - c + bx - x^2 = -c$. Dabei ergibt sich die eine Richtung durch Multiplikation mit x, die umgekehrte Folgerung ist möglich, weil X kein Nullteiler in $\mathbb{Z}_n[X]$ ist (Satz 3.15).

Der Test $x^{\frac{n+1}{2}} \stackrel{?}{\in} \mathbb{Z}_n$ in Zeile 2 und die Vorbedingung $\left(\frac{-c}{n}\right) = 1$ gehören zusammen. Da $n = p$ eine Primzahl ist, muss $x^{n+1} = -c$ und $-c$ ein quadratischer Rest im Körper $\mathbb{Z}_n$ sein. Demzufolge liegt auch die Quadratwurzel $x^{\frac{n+1}{2}}$ von x^{n+1} in $\mathbb{Z}_n$. Genau das wird in Zeile 2 überprüft. Der Test in Zeile 2 zusammen mit der Vorbedingung an $\left(\frac{-c}{n}\right)$ wird später bei der Abschätzung der Irrtumswahrscheinlichkeit des RQFT nur an einer einzigen Stelle benutzt, in Lemma 6.14.

Die Zeilen 7 und 8 erinnern zu Recht an den Miller-Rabin-Test. Es ist nämlich der gleiche mathematische Sachverhalt, der hier überprüft wird: in Körpern hat 1 als Quadratwurzeln nur 1 und -1 (Satz 3.8). Die Struktur $\mathbb{F}_{p^2} \simeq \mathbb{Z}_p[X]/(X^2 - bX - c)$ ist — wie $\mathbb{Z}_p$ — ein Körper. Wenn wir also gezeigt haben, dass $x^{(p^2-1)/2} = x^{2^{r-1}s} = 1$ ist in $\mathbb{Z}_p[X]/(X^2 - bX - c)$, dann folgt wie im Beweis von (5.2), dass die Bedingung in Zeile 8 nicht erfüllt ist, weil wir vorausgesetzt haben, dass der Algorithmus mit einer Primzahl $n = p$ arbeitet.

Wie zeigen wir $x^{(p^2-1)/2} = 1$? Satz 3.43 und Satz 3.44 auf $\mathbb{Z}_p[X]/(X^2 - bX - c)$ angewendet ergibt nur $x^{p^2-1} = x^{2^r s} = 1$. Aufgrund der Vorbedingung an c gilt jedoch $\left(\frac{-c}{p}\right) = 1$, was $x^{(p+1)/2} \in \mathbb{Z}_p$ impliziert, wie wir gesehen haben. Damit ergibt sich, wenn wir den Kleinen Satz von Fermat (Satz 3.45) auf $x^{(p+1)/2}$ anwenden,

$$x^{(p^2-1)/2} = (x^{(p+1)/2})^{p-1} = 1. \tag{6.8}$$

Nachdem wir gezeigt haben, dass Algorithmus 8 Primzahlen richtig klassifiziert, nämlich als „Primzahlkandidaten“, könnten wir aus den Tests in den Zeilen 2, 4 und 8 von Algorithmus 8 wieder eine mathematische Tatsache der Form (2.4) formulieren. Wir verzichten darauf, weil wir dafür schon drei Beispiele ((5.1) und Algorithmus 5; (6.2) und Algorithmus 7; (5.2), daraus abgeleitet Algorithmus 6) gesehen haben. Außerdem verzichten wir auf eine formale Definition der Begriffe *QFT-Primzahlkandidat*, *QFT-Pseudoprimzahl* und *QFT-Zeuge*. Deren Bedeutung ist mit einem Hinweis auf die Analogie zu den schon definieren Begriffen Fermat-Primzahlkandidat, Fermat-Pseudoprimzahl, Fermat-Zeuge, Miller-Rabin-Primzahlkandidat usw. sowie aufgrund von De-

finition 2.2 und Algorithmus 8 offensichtlich.

6.3. Anzahl der für den QFT zulässigen Paare (b, c)

Unser Ziel ist, den QFT zu randomisieren, indem b, c nicht mehr als Teil der Eingabe entgegengenommen, sondern vom Algorithmus zufällig gewählt werden. Über den so entstandenen Algorithmus, den RQFT, wollen wir zeigen, dass er ein Primzahltest ist. Dazu müssen wir für jede zu untersuchende zusammengesetzte Zahl n abschätzen, welchen Anteil die QFT-Zeugen für n unter allen Paaren (b, c), die für den QFT zusammen mit n als Eingabe zulässig sind, mindestens ausmachen. Anders als beim Fermat- und Miller-Rabin-Test ist für vorgegebenes n die Anzahl der Paare (b, c), so dass n, b, c zulässige Eingabe für den QFT ist — die Anzahl der *für den QFT mit n zulässigen Paare* (b, c) — nicht offensichtlich. Deshalb beschäftigt sich dieser Abschnitt genauer damit.

Außerdem ist das Problem, ein für den QFT mit n zulässiges Paar (b, c) zufällig zu wählen nicht so trivial, wie die zufällige Wahl eines a mit $2 \leq a \leq n-2$ (vgl. Algorithmus 6). Zu untersuchen ist hier nämlich, nach wie vielen Zufallsexperimenten, in denen jeweils ein Paar (b, c) mit $0 \leq b, c \leq n-1$ gewählt wird, man erwarten kann, ein zulässiges Paar zu finden. Auch zur Klärung dieser Frage dienen die Überlegungen des aktuellen Abschnittes.

Definition 6.3. *Sei $n > 2$ eine ungerade natürliche Zahl. Mit $Z(n)$ sei die Anzahl der Paare (b, c) modulo n bezeichnet, für die gilt*

1. $\left(\frac{b^2+4c}{n}\right) = -1$ *und* $\left(\frac{-c}{n}\right) = 1$ *oder*
2. $1 < \mathrm{ggT}(b^2 + 4c, n) < n$ *oder*
3. $1 < \mathrm{ggT}(c, n) < n$.

Wie zu sehen ist, bezeichnen wir mit $Z(n)$ nicht die Anzahl der für den QFT mit n zulässigen Paare (b, c), sondern die Mächtigkeit einer größeren Menge. Es bietet sich an, diese größere Menge zu nutzen, weil alle darin enthaltenen Elemente, die nicht für den QFT mit n zulässig sind, weil sie 2. oder 3. erfüllen, ebenso „gut“ sind wie QFT-Zeugen für n, denn sie offenbaren einen nichttrivialen Faktor von n. Ziel dieses Abschnittes ist nun der Beweis des folgenden Satzes aus [Gra98]:

Satz 6.4. *Sei n eine ungerade zusammengesetzte natürliche Zahl, aber keine Quadratzahl. Dann gilt: $Z(n) > \frac{n^2}{4}$.*

Zum Beweis dieses Satzes werden wir der Argumentation in [Gra98] folgen. Lemma 6.5 und Lemma 6.6 mit ihren Beweisen sind daher direkt [Gra98] entnommen, wobei die Beweise teilweise etwas aufbereitet wurden. Die Aussage von Lemma 6.8 wird in [Gra98] jedoch ohne Beweis verwendet, so dass sie hier selbständig nachgewiesen werden muss. Als Hilfe dazu wird Lemma 6.7 formuliert und bewiesen.

Lemma 6.5. *Sei p eine ungerade Primzahl und $s_1, s_2 \in \{-1, 1\}$. Sei l die Anzahl der Paare (b, c) modulo p, für die*

$$\left(\frac{b^2 + 4c}{p}\right) = s_1 \quad \textit{und} \quad \left(\frac{-c}{p}\right) = s_2 \tag{6.9}$$

ist. Dann gilt:

$$l = \begin{cases} \frac{(p-1)^2}{4} - s_1 \frac{p-1}{2}, & \textit{falls } s_1 = s_2 \textit{ und} \\ \frac{(p-1)^2}{4}, & \textit{falls } s_1 \neq s_2. \end{cases}$$

Beweis. Wir erinnern uns an Satz 3.49 und wählen einen beliebigen quadratischen Nichtrest $r \in \mathbb{Z}_p^*$, der für den Rest des Beweises fest ist. Außerdem setzen wir $R_1 = 1$, falls $s_1 = 1$, und $R_1 = r$, falls $s_1 = -1$. Analog definieren wir $R_2 = 1$, falls $s_2 = 1$, und $R_2 = r$ sonst.

Sei $(b, c) \in \mathbb{Z}_p \times \mathbb{Z}_p$ ein Paar, das (6.9) erfüllt. Nach Satz 3.49 gibt es zu diesem Paar genau 4 Paare $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$, für die gilt $b^2 + 4c \equiv R_1 x^2 \bmod p$ und $-c \equiv R_2 y^2 \bmod p$. Daraus folgt durch Einsetzen der zweiten Gleichung in die erste

$$b^2 \equiv R_2(2y)^2 + R_1 x^2 \bmod p. \tag{6.10}$$

Für den Rest des Beweises sei vereinbart, dass mit der Schreibweise $\{(x, y) \mid \dots\}$ bzw. $\{(x, y, z) \mid \dots\}$ stets Teilmengen von $\mathbb{Z}_p \times \mathbb{Z}_p$ bzw. $\mathbb{Z}_p^3$ gemeint und dass alle Kongruenzen als Kongruenzen modulo p zu lesen sind.

Wir kehren nun die Überlegungen um, die auf (6.10) geführt haben. Dazu definieren wir die Abbildung F, die einem Tripel (b, x, y) aus der Menge

$$M = \{(b, x, y) \mid x, y \neq 0, b^2 \equiv R_2(2y)^2 + R_1 x^2\} \tag{6.11}$$

das Paar $(b, -R_2 y^2)$ zuordnet. F ist so konstruiert, dass $F(b, x, y)$ die Gleichung (6.9) erfüllt. Außerdem ist jedes Paar (b, c), das diese Gleichung erfüllt, Bild eines Tripels

(b, x, y) aus M, genauer gesagt von genau vier solchen Tripeln. Schließlich gilt sogar $F^{-1}(b, c) \cap F^{-1}(b', c') = \emptyset$, falls $(b, c) \neq (b', c')$: Wenn $b \neq b'$ ist dies klar. Falls $b = b'$, muss $c \neq c'$ gelten, so dass für $(b, x, y) \in F^{-1}(b, c)$ und $(b', x', y') \in F^{-1}(b', c')$ nicht $y = y'$ gelten kann.

Die Abbildung F stellt also so etwas wie eine „4-zu-1-Korrespondenz“ her zwischen M und den Paaren (b, c), die (6.9) erfüllen. Deshalb gilt $l = |M|/4$.

Wir zählen im Folgenden M ab. Dazu bezeichnen wir $|M|$ mit n_1, falls $s_1 = s_2 = 1$, und mit n_2, falls $s_1 = s_2 = -1$. Wenn $s_1 \neq s_2$, sieht man an (6.11), dass die Kardinalität von M nicht davon abhängt, ob $(s_1, s_2) = (1, -1)$ oder $(-1, 1)$ ist. Deshalb bezeichnen wir $|M|$ mit n_3, falls $s_1 \neq s_2$.

Setzt man $s_1 = s_2 = 1$ in (6.11) ein, so ergibt sich

$$\begin{aligned}
n_1 &= |\underbrace{\{(x, y, z) \mid x, y \neq 0, x^2 + y^2 = z^2\}}_{=:A}| \\
&= |\{(x, y, z) \mid x, y \neq 0, (z - y)(z + y) = x^2\}| \\
&= \sum_{x=1}^{p-1} |\{(a, b) \mid a \neq b, ab = x^2\}| \\
&= \sum_{x=1}^{p-1} |\{a \in \mathbb{Z}_p \mid a \neq 0, a \neq x, a \neq -x\}| \\
&= (p - 1)(p - 3).
\end{aligned}$$

Man rechnet nach, dass $(p-1)(p-3)/4 = (p-1)^2/4 - (p-1)/2$ ist. Im Fall $s_1 = s_2 = 1$ ist damit die Behauptung gezeigt.

Um n_2 zu bestimmen setzen wir zunächst $s_1 = s_2 = -1$ in (6.11) ein und erhalten

$$\begin{aligned}
n_2 &= |\{(x, y, z) \mid x, y \neq 0, rx^2 + ry^2 = z^2\}| \\
&= |\{(x, y, z) \mid x, y \neq 0, x^2 + y^2 = r(zr^{-1})^2\}| \\
&= |\underbrace{\{(x, y, z) \mid x, y \neq 0, x^2 + y^2 = rz^2\}}_{=:B}|.
\end{aligned}$$

Wir wollen uns nun überlegen, dass $2|\{(x, y) \mid x, y \neq 0\}| = |A| + |B|$ ist. Dazu gehen wir wie in Abbildung 6.1 angedeutet vor: Sind $x, y \in \mathbb{Z}_p$ fest vorgegeben, so ist die Summe $x^2 + y^2$ entweder ein quadratischer Rest modulo p oder 0 oder ein quadratischer Nichtrest modulo p. Falls sie ein quadratischer Rest ist, so entspricht (x, y) genau zwei Tripeln in A, nämlich einem Tripel (x, y, z) und dem Tripel $(x, y, -z)$. Falls $x^2 + y^2 = 0$

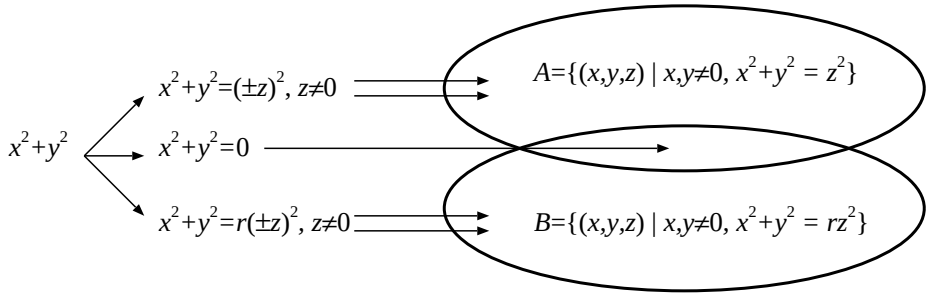

Abbildung 6.1.: Zum Abzählen von B.

ist, gibt es genau ein $z \in \mathbb{Z}_p$ mit $x^2 + y^2 = z^2$ oder rz^2, nämlich $z = 0$. Das Paar (x, y) entspricht in diesem Fall dem Tripel $(x, y, 0) \in A \cap B$ und wird in $|A| + |B|$ einmal bei A und einmal bei B gezählt. Falls $x^2 + y^2$ ein quadratischer Nichtrest ist, entspricht (x, y) zwei Tripeln in B. Damit gilt $n_2 = 2(p-1)^2 - n_1 = 2(p-1)^2 - (p-1)^2 + 2(p-1)$, also die Behauptung für den Fall $s_1 = s_2 = -1$.

Mit $(s_1, s_2) = (1, -1)$ bzw. $(-1, 1)$ in (6.11) ergibt sich:

$$\begin{aligned} n_3 &= |\{(x, y, z) \mid x, y \neq 0, x^2 + ry^2 = z^2\}| \\ &= |\{(x, y, z) \mid x, y \neq 0, rx^2 + (ry)^2 = rz^2\}| \\ &= |\{(x, y, z) \mid x, y \neq 0, x^2 + ry^2 = rz^2\}|. \end{aligned}$$

Mit einer ähnlichen Überlegung wie in Abbildung 6.1 folgt nun

$$\begin{aligned} &2|\{(x, y) \mid x, y \neq 0\}| \\ &= |\{(x, y, z) \mid x, y, z \neq 0, x^2 + ry^2 = z^2\}| \\ &\quad + 2|\{(x, y, 0) \mid x, y \neq 0, x^2 + ry^2 = 0\}| \\ &\quad + |\{(x, y, z) \mid x, y, z \neq 0, x^2 + ry^2 = rz^2\}| \\ &= n_3 + n_3, \end{aligned}$$

woraus sich $n_3 = (p - 1)^2$ und die Behauptung auch im letzten Fall ergibt. □

Lemma 6.6. *Sei n eine ungerade quadratfreie ganze Zahl und $s_1, s_2 \in \{-1, 1\}$. Mit l sei die Anzahl der Paare (b, c) modulo n bezeichnet, für die*

$$\left(\frac{b^2 + 4c}{n}\right) = s_1 \quad \textit{und} \quad \left(\frac{-c}{n}\right) = s_2 \tag{6.12}$$

gilt. Es ist

$$l = \begin{cases} \frac{1}{4}\varphi(n)^2 + \frac{s_1}{2}\mu(n)\varphi(n), & \textit{falls } s_1 = s_2 \\ \frac{1}{4}\varphi(n)^2, & \textit{sonst.} \end{cases}$$

Beweis. Sei $n = p_1 p_2 \cdots p_k$. Wir führen den Beweis durch Induktion nach $k \geq 1$. Den Induktionsanfang liefert Lemma 6.5.

Sei $N_{s_1,s_2}(n)$ die Anzahl der Paare (b, c) mod n, für die (6.12) gilt. Weiter sei p ein Primfaktor von n und $m = n/p$. Aufgrund des Chinesischen Restsatzes (Satz 3.40) und wegen Satz 3.50 gilt:

$$\begin{aligned} N_{s_1,s_2}(n) =& N_{s_1,s_2}(m)N_{1,1}(p) + N_{-s_1,s_2}(m)N_{-1,1}(p) \\ &+ N_{s_1,-s_2}(m)N_{1,-1}(p) + N_{-s_1,-s_2}(m)N_{-1,-1}(p). \end{aligned} \tag{6.13}$$

Damit vollziehen wir den Induktionsschritt. Zunächst setzen wir $s_1 = s_2$ in (6.13) ein:

$$\begin{aligned} N_{s_1,s_2}(n) =& \left(\frac{\varphi(m)^2}{4} + \frac{s_1}{2}\mu(m)\varphi(m)\right)\left(\frac{(p-1)^2}{4} - \frac{p-1}{2}\right) + \frac{\varphi(m)^2}{4}\cdot\frac{(p-1)^2}{4} \\ &+ \frac{\varphi(m)^2}{4}\cdot\frac{(p-1)^2}{4} + \left(\frac{\varphi(m)^2}{4} - \frac{s_1}{2}\mu(m)\varphi(m)\right)\left(\frac{(p-1)^2}{4} + \frac{p-1}{2}\right) \\ =& \frac{1}{4}\varphi(m)^2(p-1)^2 - \frac{1}{2}s_1\mu(m)\varphi(m)(p-1) \\ =& \frac{1}{4}\varphi(mp)^2 + \frac{1}{2}s_1\mu(mp)\varphi(mp). \end{aligned}$$

Für den letzten Schritt wurden Satz 3.41 und Lemma 3.39 benutzt.

Wir beenden den Beweis, indem wir $s_1 \neq s_2$ in (6.13) einsetzen:

$$\begin{aligned} N_{s_1,s_2}(n) =& \frac{\varphi(m)^2}{4}\left(\frac{(p-1)^2}{4} - \frac{p-1}{2}\right) + \left(\frac{\varphi(m)^2}{4} - \frac{s_1}{2}\mu(m)\varphi(m)\right)\frac{(p-1)^2}{4} \\ &+ \left(\frac{\varphi(m)^2}{4} + \frac{s_1}{2}\mu(m)\varphi(m)\right)\frac{(p-1)^2}{4} + \frac{\varphi(m)^2}{4}\left(\frac{(p-1)^2}{4} + \frac{p-1}{2}\right) \\ =& \frac{1}{4}\varphi(m)^2(p-1)^2 = \frac{1}{4}\varphi(mp)^2. \quad \square \end{aligned}$$

Die folgenden beiden Lemmas werden es uns ermöglichen, auch für nicht quadratfreie Zahlen n die Anzahl der für den QFT mit n zulässigen Paare (b, c) zu bestimmen.

Lemma 6.7. *Sei $p \neq 2$ eine Primzahl. Dann gibt es genau $\varphi(p^2)^2$ Paare (b, c) modulo p^2, für die gilt: $b^2 + 4c \not\equiv 0 \bmod p$ und $-c \not\equiv 0 \bmod p$.*

Beweis. Zuerst bemerken wir, dass nach Satz 3.41 gilt: $(p(p-1))^2 = \varphi(p^2)^2$.

Nun zählen wir Paare ab. Es gibt $(p^2)^2$ Paare modulo p^2. Von diesen gehören *nicht* zu den von uns betrachteten genau diejenigen, die entweder

(1) $b^2 + 4c \equiv 0 \bmod p$ und $c \not\equiv 0 \bmod p$ oder

(2) $b^2 + 4c \equiv 0 \bmod p$ und $c \equiv 0 \bmod p$ oder

(3) $b^2 + 4c \not\equiv 0 \bmod p$ und $c \equiv 0 \bmod p$

erfüllen.

Der Fall (1) ist für modulo p festgelegtes c äquivalent zu $b^2 \equiv -4c \bmod p$. Diese Gleichung ist lösbar genau dann, wenn $\left(\frac{-4c}{p}\right) = 1$. Wenn sie lösbar ist, hat sie für b genau zwei Lösungen modulo p. Da mit c auch $-4c$ alle Restklassen modulo p durchläuft, gibt es genau

$$\underbrace{\underbrace{\frac{p-1}{2}}_{\substack{c\text{'s modulo } p, \text{ die quadratische Reste sind} \\ c\text{'s modulo } p^2}} \cdot p \cdot \underbrace{2}_{\substack{b\text{'s modulo } p \text{ für festes } c}} \cdot p}_{\substack{b\text{'s modulo } p^2 \text{ für festes } c}} = p^2(p-1)$$

Paare (b, c) modulo p^2, die (1) erfüllen.

Genau p^2 Paare modulo p^2 erfüllen (2), weil dies äquivalent zu $(b, c) \equiv (0, 0) \bmod p$ ist.

Der letzte Fall ist äquivalent zu $b \not\equiv 0 \bmod p$ und $c \equiv 0 \bmod p$. Also gibt es genau $(p-1)p \cdot p = p^2(p-1)$ Paare (b, c) modulo p^2, die (3) erfüllen.

Insgesamt stellen wir nun fest, dass es $p^4 - p^2(p-1) - p^2 - p^2(p-1) = p^2(p^2 - 2(p-1) - 1) = p^2(p^2 - 2p + 1) = p^2(p-1)^2$ Paare mit der im Lemma betrachteten Eigenschaft gibt. □

Lemma 6.8. *Sei n eine ungerade natürliche Zahl, die keine Quadratzahl ist, $p \neq 2$ eine Primzahl und $s_1, s_2 \in \{-1, 1\}$. Weiter sei ein Paar (b', c') ganzer Zahlen gegeben, für das gilt*

$$\left(\frac{b'^2 + 4c'}{n}\right) = s_1 \quad \textit{und} \quad \left(\frac{-c'}{n}\right) = s_2.$$

Dann gibt es modulo np^2 genau $\frac{\varphi(np^2)^2}{\varphi(n)^2}$ Paare (b, c), für die gilt:

$$(b, c) \equiv (b', c) \bmod n, \quad \left(\frac{b^2 + 4c}{np^2}\right) = s_1 \quad \textit{und} \quad \left(\frac{-c}{np^2}\right) = s_2. \tag{6.14}$$

Beweis. Wir unterscheiden die Fälle $p|n$ und $p \not| n$. Gelte zunächst $p|n$. Dann muss $b'^2 + 4c' \not\equiv 0 \bmod p$ sein, weil sonst $\left(\frac{b'^2+4c'}{n}\right) = \left(\frac{b'^2+4c'}{n/p}\right)\left(\frac{b'^2+4c'}{p}\right) = \left(\frac{b'^2+4c'}{n/p}\right) \cdot 0 = 0 \neq s_1$ wäre. Analog gilt $c' \not\equiv 0 \bmod p$. Deshalb gilt $b^2 + 4c \not\equiv 0 \bmod p$ und $c \not\equiv 0 \bmod p$ auch für jedes Paar (b, c) ganzer Zahlen mit $(b, c) \equiv (b', c') \bmod n$. Jedes solche Paar erfüllt

$$\left(\frac{b^2 + 4c}{np^2}\right) = \left(\frac{b^2 + 4c}{n}\right)\left(\left(\frac{b^2 + 4c}{p}\right)\right)^2 = \left(\frac{b'^2 + 4c'}{n}\right) \cdot (\pm 1)^2 = s_1$$

und analog $\left(\frac{-c}{np^2}\right) = s_2$. Also ist jedes der $(p^2)^2$ Paare (b, c) modulo np^2 mit $(b, c) \equiv (b', c') \bmod n$, die es nach dem Chinesischen Restsatz gibt, eine Lösung von (6.14). Sei nun r die größte ganze Zahl mit $p^r|n$ und sei $k = n/p^r$. Es gilt $r \geq 1$ und $\mathrm{ggT}(p^r, k) = 1$. Nun folgt: $\frac{\varphi(np^2)}{\varphi(n)} = \frac{\varphi(kp^{r+2})}{\varphi(kp^r)} = \frac{\varphi(k)\varphi(p^{r+2})}{\varphi(k)\varphi(p^r)} = \frac{p^{r+1}(p-1)}{p^{r-1}(p-1)} = p^2$. Damit ist die Behauptung im Fall $p|n$ gezeigt.

Wir untersuchen nun den Fall $p \not| n$. Hier sind n und p teilerfremd und nach dem Chinesischen Restsatz ist ein Paar (b, c) modulo np^2 eindeutig bestimmt durch (b, c) modulo n und (b, c) modulo p^2. Damit (6.14) erfüllt sein kann, muss $(b, c) \equiv (b', c') \bmod n$ gelten. Wenn dies zutrifft, ist

$$\begin{aligned}
\left(\frac{b^2 + 4c}{np^2}\right) &= \left(\frac{b^2 + 4c}{n}\right)\left(\left(\frac{b^2 + 4c}{p}\right)\right)^2 \\
&= \left(\frac{b'^2 + 4c'}{n}\right) \cdot \left(\left(\frac{b^2 + 4c}{p}\right)\right)^2 \\
&= s_1 \cdot \begin{cases} 0, & \text{falls } b^2 + 4c \equiv 0 \bmod p, \\ 1, & \text{falls } b^2 + 4c \not\equiv 0 \bmod p. \end{cases}
\end{aligned}$$

Analog gilt

$$\left(\frac{-c}{np^2}\right) = \begin{cases} 0, & \text{falls } c \equiv 0 \bmod p, \\ s_2, & \text{sonst.} \end{cases}$$

Nach Lemma 6.7 gibt es also genau $\varphi(p^2)^2$ Paare (b, c) modulo np^2, die (6.14) erfüllen. Die Beobachtung $\frac{\varphi(np^2)}{\varphi(n)} = \frac{\varphi(n)\varphi(p^2)}{\varphi(n)} = \varphi(p^2)$ beschließt den Beweis. □

Beweis von Satz 6.4

Beweis. Wir interessieren uns zunächst für die Anzahl der Paare (b, c) modulo n, für die $\left(\frac{b^2+4c}{n}\right)$ und $\left(\frac{-c}{n}\right)$ beide nicht den Wert 0 annehmen, aber auch nicht genau die in der Definition von $Z(n)$ auf Seite 89 unter 1. angegebene Kombination. Erlaubt sind

also die Werte-Kombinationen $(1,1)$, $(1,-1)$ und $(-1,-1)$. Abkürzend nennen wir im Rest dieses Beweises ein Paar (b,c), das diese Bedingungen erfüllt, „interessant“.

In einem ersten Schritt betrachten wir den Fall, dass n quadratfrei ist. Nach Lemma 6.6 gibt es dann $\varphi(n)^2/4 + 1/2\mu(n)\varphi(n) + \varphi(n)^2/4 + \varphi(n)^2/4 - 1/2\mu(n)\varphi(n) = \frac{3}{4}\varphi(n)^2$ interessante Paare.

Wir wollen nun sehen, dass diese Formel auch dann gilt, wenn n nicht quadratfrei ist, aber den übrigen Einschränkungen des Satzes an n genügt. Wir halten fest, dass folgender Schritt möglich ist: Sei n gegeben, für das es genau $\frac{3}{4}\varphi(n)^2$ interessante Paare modulo n gibt, und $p \neq 2$ eine Primzahl. Da jedes interessante Paar modulo n nach Lemma 6.8 genau $\frac{\varphi(np^2)^2}{\varphi(n)^2}$ interessanten Paaren modulo np^2 entspricht, gibt es dann $\frac{3}{4}\varphi(n)^2\frac{\varphi(np^2)^2}{\varphi(n)^2} = \frac{3}{4}\varphi(np^2)^2$ interessante Paare modulo np^2. Damit können wir die Formel auf nicht notwendig quadratfreie n übertragen: Solche n lassen sich zerlegen in $n = m_1p_1^2 = m_2p_2^2p_1^2 = \ldots = m_kp_k^2p_{k-1}^2\cdots p_1^2$, wobei m_k quadratfrei ist und $p_1,\ldots,p_k$ (nicht notwendig verschiedene) Primzahlen sind. Für m_k gilt die Formel. Also ist unser Schritt möglich und die Formel gilt auch für $m_{k-1} = m_kp_k^2$. Ein weiterer Schritt ergibt, dass die Formel für $m_{k-2} = m_{k-1}p_{k-1}^2$ gilt. So fortfahrend gelangt man schließlich zu dem Ergebnis, dass auch für $n = m_1p_1^2$ die Formel richtig ist.

Nun konzentrieren wir uns wieder auf die Aussage des Satzes. Die Menge aller Paare (b,c) modulo n zerfällt in die Menge der interessanten Paare, die Menge der Paare, die 1. aus der Definition von $Z(n)$ erfüllen, und die Menge der Paare mit $\mathrm{ggT}(b^2+4c,n) \neq 1$ oder $\mathrm{ggT}(-c,n) \neq 1$. Letztere ist die Vereinigung der Menge der Paare, die 2. oder 3. aus der Definition von $Z(n)$ erfüllen, und der Menge A der Paare mit $\mathrm{ggT}(b^2+4c,n) = n$ oder $\mathrm{ggT}(-c,n) = n$. Modulo n ist $A = \{(b,-4^{-1}b^2) \mid 0 \leq b < n\} \cup \{(b,0) \mid 0 \leq b < n\}$. Beide Mengen überschneiden sich nur in $\{(0,0)\}$, also ist $|A| = 2n-1$.

Also gilt $Z(n) \geq n^2 - \frac{3}{4}\varphi(n)^2 - |A| \geq n^2 - \frac{3}{4}(n-2)^2 - 2n + 1 = \frac{n^2}{4} + n - 2 > \frac{n^2}{4}$. □

6.4. Der randomisierte starke quadratische Frobenius-Test (RQFT)

Nun sind wir bereit, den Primzahltest zu definieren und zu diskutieren, der im Mittelpunkt dieser Arbeit steht, nämlich den *randomisierten starken quadratischen Frobenius-Test* (*RQFT*, Algorithmus 9). Um eine Irrtumswahrscheinlichkeit von weniger als $\frac{1}{7710}$ zu erreichen, genügt es, dem Vorschlag aus [Gra98] zu folgen und $B = 50000$ zu wählen.

Algorithmus 9 Der randomisierte starke quadr. Frobenius-Test (RQFT) aus [Gra98].

Require: n. $n > 2$ ungerade, keine Quadratzahl und nicht teilbar durch Primzahlen $\leq B$.

1: Wähle zufällig Paare (b, c) mit $0 \leq b \leq n-1$ und $1 \leq c \leq n-1$, bis

   (a) $1 < \mathrm{ggT}(b, n)$, $1 < \mathrm{ggT}(c, n)$ oder $1 < \mathrm{ggT}(b^2 + 4c, n) < n$ oder (falls dies nicht zutrifft)

   (b) $\left(\frac{b^2+4c}{n}\right) = -1$ und $\left(\frac{-c}{n}\right) = 1$ oder (falls dies nicht zutrifft)

   (c) mehr als B Paare gewählt wurden.

2: **if** ein Paar mit (a) wurde gefunden **then**
3: **return** „n ist nicht prim.“
4: **else if** ein Paar (b, c) mit (b) wurde gefunden **then**
5: **if** QFT(n, b, c) erklärt n zum Primzahlkandidaten **then**
6: **return** „n ist prim.“
7: **else**
8: **return** „n ist nicht prim.“
9: **end if**
10: **else**
11: **return** „n ist prim.“
12: **end if**

Zunächst bemerken wir, dass die Laufzeit von Algorithmus 9 deterministisch beschränkt ist. Spätestens nachdem B mal ein Paar gewählt und einmal der QFT ausgeführt wurde, endet der Algorithmus — egal, welche Ergebnisse die Zufallsexperimente lieferten.

Weiterhin sehen wir, dass der Algorithmus Primzahlen stets richtig klassifiziert. Denn wir wissen schon, dass der QFT Primzahlen stets zu Primzahlkandidaten erklärt. Falls es nicht zu einem Aufruf der QFT kommt, weil der RQFT in Zeile 1 stets Paare wählt, die weder (a) noch (b) erfüllen, dann gelangt er zu Zeile 11, wo das für Primzahlen zutreffende Ergebnis „n ist prim“ ausgegeben wird. Wir bemerken, dass es unumgänglich ist, dass der Algorithmus in Zeile 11 diese Ausgabe tätigt, weil er sonst die Bedingung (2.2) nicht erfüllen würde und kein Primzahltest im Sinne von Definition 2.1 wäre.

Wie ist die Möglichkeit zu bewerten, dass Algorithmus 9 auch zusammengesetzte Eingaben in Zeile 11 zu Primzahlen erklären kann? Die Wahrscheinlichkeit, dass dies passiert, ist sehr gering: Unter den n^2 Paaren (b, c), $0 \leq b, c \leq n-1$, gibt es nach Satz 6.4 weniger als $\frac{3}{4}n^2$, die weder $\left(\frac{b^2+4c}{n}\right) = -1$ und $\left(\frac{-c}{n}\right) = 1$ noch $1 < \mathrm{ggT}(b^2 + 4c, n) < n$ oder $1 < \mathrm{ggT}(c, n)$ erfüllen. Nur wenn ein solches Paar gewählt wird, ist weder (a) noch (b) erfüllt. Damit[4] ist die Wahrscheinlichkeit, dass das Zufallsexperiment in Zeile 1 wiederholt werden muss, kleiner als $\frac{3}{4}n^2/n^2$. Wenn wir $B = 50000$ wählen, ist die Wahrscheinlichkeit, dass auch nach B unabhängigen Experimenten kein Paar gefunden ist, das (a) oder (b) erfüllt, kleiner als $0.75^B = 10^{B \log_{10} 0.75} < 10^{-50000 \cdot 0.124} = 10^{-6200}$. Diese Irrtumswahrscheinlichkeit ist winzig. Es wird sich herausstellen, dass sie die angestrebte Schranke von 1/7710 für die Irrtumswahrscheinlichkeit des RQFT nicht beeinflusst.

Anstatt in Zeile 1 eine Obergrenze für die Anzahl der Zufallsexperimente vorzugeben, könnte man auch solange Paare (b, c) wählen, bis eines gefunden ist, das (a) oder (b) erfüllt. Wegen Satz 6.4 ist ja zu erwarten, dass dies nach spätestens vier Versuchen der Fall ist. Durch diese Änderung würde die deterministische Laufzeitschranke von Algorithmus 9 aber verlorengehen.

Dass zusammengesetzte Zahlen mit kleinen Primteilern vor dem RQFT ausgeschlos-

[4] Um ganz exakt zu sein, muss noch bemerkt werden, dass die im Voraus ausgeschlossenen Paare $(b, 0)$ stets zu einer Wiederholung des Zufallsexperiments in Zeile 1 führen würden. Aufgrund dieser Tatsache ist die Wahrscheinlichkeit, dass das Experiment wiederholt werden muss, tatsächlich noch geringer als angegeben. Ebenfalls verträglich mit der Abschätzung ist die Tatsache, dass in (a) durch den Test $1 < \mathrm{ggT}(b, n)$ höchstens noch mehr Paare zum sofortigen Ende der Zufallsexperimente führen als ohne diesen Test.

sen werden sollen, ist zum einen darin begründet, dass die meisten zusammengesetzten Zahlen solche kleinen Teiler haben und in diesen Fällen einige Divisionen mit Rest schneller zum Ergebnis führen als ein Durchlauf des RQFT. Kleine Teiler müssen aber vor allem deshalb ausgeschlossen werden, weil für manche Eingaben n die Größe des kleinsten Primfaktors von n in die Abschätzung der Irrtumswahrscheinlichkeit des RQFT mit eingeht.

Die Irrtumswahrscheinlichkeit des RQFT lässt sich wie folgt abschätzen, wobei n eine den Vorbedingungen von Algorithmus 9 genügende zusammengesetzte Zahl sein soll:

$$
\begin{aligned}
&\text{Prob(Algorithmus 9 gibt „}n\text{ ist prim“ aus)}\\
\leq&\,\text{Prob(kein Paar } (b,c) \text{ gefunden, für das (a) oder (b) zutrifft)+}\\
&\text{Prob(gefundenes Paar } (b,c) \text{ führt zum Ergebnis „}n\text{ ist prim“)}\\
<&\,10^{-6200} + \frac{|\{(b,c) \mid \text{(b) trifft zu, aber nicht (a), QFT}(n,b,c)\text{: „Primzahlkandidat“}\}|}{|\{(b,c) \mid \text{(a) oder (b) trifft zu}\}|}\\
\leq&\,10^{-6200} + \frac{|\{(b,c) \mid \text{(b) trifft zu, aber nicht (a), QFT}(n,b,c)\text{: „Primzahlkandidat“}\}|}{|\{(b,c) \mid 1 < \text{ggT}(c,n) \text{ oder } 1 < \text{ggT}(b^2+4c,n) < n \text{ oder (b) trifft zu}\}|}\\
=&\,10^{-6200} + \frac{|\{(b,c) \mid \text{(b) trifft zu, aber nicht (a), QFT}(n,b,c)\text{: „Primzahlkandidat“}\}|}{Z(n)}\\
\leq&\,10^{-6200} + \frac{|\{(b,c) \mid \text{QFT}(n,b,c)\text{: „Primzahlkandidat“}\}|}{Z(n)}
\end{aligned}
$$

Solange der rechte Summand wesentlich größer ist als 10^{-6200}, ist also die folgende Definition sinnvoll:

Definition 6.9. *Sei n eine zusammengesetzte Zahl, die keine Quadratzahl ist und durch keine Primzahl ≤ 50000 teilbar ist. Wir sagen, dass n den RQFT mit Wahrscheinlichkeit $r \in \mathbb{R}$ besteht dann und nur dann, wenn es genau $rZ(n)$ Paare (b,c) ganzer Zahlen b,c, $0 \leq b,c \leq n-1$, gibt, so dass der QFT bei Eingabe (n,b,c) die Ausgabe „n ist QFT-Primzahlkandidat bezüglich (b,c)“ liefert.*

6.5. Irrtumswahrscheinlichkeit des RQFT

Das Ziel dieses Abschnitts ist es zu beweisen, dass der RQFT die Bedingung (2.3) mit $\varepsilon = 1/7710$ erfüllt. Damit ist dann insbesondere bewiesen, dass der RQFT überhaupt ein Primzahltest ist. Wir folgen dabei genau der Argumentation aus [Gra98], werden

die Beweise aber teilweise ausführlicher führen. Außerdem haben wir schon in Kapitel 3 eine Aussage bewiesen (Lemma 3.61), die in [Gra98] an zwei Stellen benutzt, aber nicht detailliert begründet wird.

Sei für den Rest des Abschnitts n eine zulässige Eingabe für den RQFT. Je nach Struktur von n geben wir verschiedene Schranken für die Irrtumswahrscheinlichkeit des RQFT auf n an. Wir beginnen mit nicht quadratfreien n.

Satz 6.10. *Wenn für eine Primzahl p gilt $p^2|n$, dann besteht n den RQFT mit Wahrscheinlichkeit kleiner als $\frac{4}{p}$.*

Beweis. Sei k die größte ganze Zahl mit $p^k|n$. Es gilt also $p^{k+1} \not| n$ und $k \geq 2$. Wenn $\mathrm{QFT}(n, b, c)$ die Zahl n zum Primzahlkandidaten erklärt, muss aufgrund von Zeile 4 in Algorithmus 8 die Gleichung $x^{n+1} = -c$ gelten. Der Test in Zeile 8 garantiert, dass $x^{n^2-1} = 1$. Damit folgt $c^{n-1} = (-c)^{n-1} = (x^{n+1})^{n-1} = 1$ in $\mathbb{Z}_n[X]/(X^2 - bX - c)$, also $c^{n-1} \equiv 1 \bmod n$ und insbesondere

$$c^{n-1} \equiv 1 \bmod p^k. \tag{6.15}$$

Wir erhalten nun die gewünschte Aussage, indem wir feststellen, wie viele Lösungen diese Gleichung höchstens haben kann. Nach Satz 3.43 ist $(\mathbb{Z}^*_{p^k}, \cdot)$ zyklisch. Wenn wir ein erzeugendes Element g dieser Gruppe wählen, gibt es eine ganze Zahl $d, 0 \leq d \leq \varphi(p^k) - 1$ mit $g^d \equiv c \bmod p^k$, so dass (6.15) äquivalent ist zu $(g^d)^{n-1} \equiv g^0 \bmod p^k$, was nach Satz 3.42 wiederum äquivalent ist zu $d(n-1) \equiv 0 \bmod \varphi(p^k)$. Nach Satz 3.24 hat diese Gleichung entweder gar keine oder genau $\mathrm{ggT}(n-1, \varphi(p^k)) = \mathrm{ggT}(n-1, p^{k-1}(p-1))$ Lösungen. Da n durch p^k teilbar ist, kann $n-1$ nicht durch p^k und folglich auch nicht durch p^{k-1} teilbar sein, so dass nach Satz 3.31 folgt $\mathrm{ggT}(n-1, p^{k-1}(p-1)) = \mathrm{ggT}(n-1, p-1) \leq p-1$. Also hat (6.15) modulo p^k höchstens $p-1$ Lösungen und es gibt für festes b modulo p^k höchstens $p-1$ Möglichkeiten, c so zu wählen, dass $\mathrm{QFT}(n, b, c)$ die Zahl n zum Primzahlkandidaten erklärt. Also ist die Anzahl aller Paare (b, c) modulo p^k, für die n den QFT bestehen kann[5], höchstens $p^k(p-1)$. Nach dem Chinesischen Restsatz (Satz 3.40) entspricht ein solches Paar modulo p^k genau $\left[\frac{n}{p^k}\right]^2$ Paaren modulo n. Somit ist die Anzahl aller Paare (b, c) modulo n, für die n den QFT besteht, nicht größer als

$$\frac{n^2}{p^{2k}} \cdot p^k(p-1) = n^2 \cdot \frac{1}{p^{k-1}}\left(1 - \frac{1}{p}\right) < n^2 \cdot \frac{1}{p^{k-1}} < \frac{4}{p^{k-1}} \cdot Z(n)$$

[5]Wir sagen „*n besteht den QFT* mit (b, c)“, wenn der QFT bei Eingabe (n, b, c) das Ergebnis „n ist QFT-Primzahlkandidat bezüglich (b, c)“ liefert.

Die letzte Ungleichung folgt aus Satz 6.4 und ergibt zusammen mit $k \geq 2$ die Behauptung. □

Nicht quadratfreie n, die der Vorbedingung von Algorithmus 9 genügen, haben einen Primfaktor $p > 50000$ mit $p^2|n$. Die Irrtumswahrscheinlichkeit des randomisierten quadratischen Frobenius-Test ist also $< 10^{-6200} + \frac{4}{50000} = 10^{-6200} + 8 \cdot 10^{-5} < 1.29 \cdot 10^{-4} < \frac{1}{7710}$. Ab jetzt können wir uns also auf quadratfreie n beschränken.

Ein quadratfreies n kann geschrieben werden als $n = p_1 p_2 \cdots p_k$, wobei $p_1, \ldots, p_k$ paarweise verschiedene Primzahlen sind. Nach Satz 3.40 ist $\mathbb{Z}_n[X]/(X^2 - bX - c) \simeq \mathbb{Z}_{p_1}[X](X^2 - bX - c) \times \cdots \times \mathbb{Z}_{p_k}[X]/(X^2 - bX - c)$. Je nachdem, ob $\left(\frac{b^2+4c}{p_i}\right) = -1$ oder 1 ist, ist $\mathbb{Z}_{p_i}[X]/(X^2 - bX - c)$ isomorph zu $\mathbb{F}_{p_i^2}$ oder zu $\mathbb{Z}_{p_i} \times \mathbb{Z}_{p_i}$ (siehe Abschnitt 3.7.2). Wenn Letzteres der Fall ist, gibt es unter den etwa p_i^2 Paaren (b, c) mod p_i nicht viele „Lügner“, wie das folgende Lemma zeigt:

Lemma 6.11. *Sei p eine Primzahl und $p|n$. Wir schreiben „n besteht den QFT mit Parametern (b, c) modulo p“, falls es $b', c', 0 \leq b', c' \leq n-1$ gibt mit $(b, c) \equiv (b', c')$ mod p, so dass n ein QFT-Primzahlkandidat ist bezüglich (b', c'). Mit dieser Sprechweise gilt: Es gibt höchstens $\frac{p-1}{2}$ Paare (b, c) modulo p mit $\left(\frac{b^2+4c}{p}\right) = 1$, so dass n den QFT mit Parametern (b, c) modulo p besteht.*

Beweis. Das Lemma behauptet eine obere Schranke für die Mächtigkeit der Menge $A := \{(b, c) \mid b, c \in \mathbb{Z}_p, \left(\frac{b^2+4c}{p}\right) = 1,\ n$ besteht mit (b, c) den QFT modulo $p\}$. Jedes Element dieser Menge wird durch $(b, c) \mapsto X^2 - bX - c$ injektiv abgebildet in die Menge $B := \{X^2 - bX - c \in \mathbb{Z}_p[X] \mid b, c \in \mathbb{Z}_p, \left(\frac{b^2+4c}{p}\right) = 1, X^{n+1} \equiv -c \bmod (p, X^2 - bX - c)\}$. Besteht nämlich n mit (b, c) modulo p den QFT, so gibt es $b', c' \in \mathbb{Z}$ mit $(b', c') \equiv (b, c) \bmod p$, so dass der QFT n zum Primzahlkandidaten bezüglich (b', c') erklärt. Wegen Zeile 4 in Algorithmus 8 ist somit $X^{n+1} \equiv -c \bmod (n, X^2 - b'X - c')$, woraus $X^{n+1} \equiv -c \bmod (p, X^2 - bX - c)$ folgt. Es gilt also $|A| \leq |B|$.

Weiter wird die Menge B injektiv abgebildet in die Menge $C =: \{\{a_1, a_2\} \mid a_1, a_2 \in \mathbb{Z}_p, a_1 \neq a_2, a_1^n \equiv a_2 \bmod p, a_2^n \equiv a_1 \bmod p\}$, wenn man jedem Polynom $X^2 - bX - c$ seine beiden Nullstellen in $\mathbb{Z}_p$ zuordnet: In Abschnitt 3.7.2 ist ab Seite 54 dargelegt, dass unter der Voraussetzung $\left(\frac{b^2+4c}{p}\right) = 1$ diese Nullstellen tatsächlich verschieden sind und in $\mathbb{Z}_p$ liegen. Wegen Satz 3.35 ist die Abbildung injektiv. Aus $X^{n+1} \equiv -c \bmod (p, X^2 - bX - c)$ folgt wegen $X^2 - bX - c = (X - a_1)(X - a_2)$ die Kongruenz $X^{n+1} \equiv -c \bmod (p, X - a_1)$. Dies zieht $a_1^{n+1} \equiv -c \equiv a_1 a_2 \bmod p$ und analog $a_2^{n+1} \equiv a_1 a_2 \bmod p$ nach sich. Wäre $a_1 \equiv 0 \bmod p$, so folgte $a_2 \equiv 0 \bmod p$, was $a_1 \not\equiv a_2 \bmod p$ widerspricht.

Ebenso ist $a_2 \equiv 0 \bmod p$ ausgeschlossen. Also folgt $a_1^n \equiv a_2 \bmod p$ und $a_2^n \equiv a_1 \bmod p$, es wird also tatsächlich in C abgebildet. Wir erhalten $|B| \leq |C|$.

Die Mächtigkeit von C lässt sich bestimmen, indem wir uns klarmachen, dass jedes Element $\{a_1, a_2\} \in C$ den beiden Elementen

$$(a_1, a_2), (a_2, a_1) \in D := \{(a_1, a_2) \mid a_1, a_2 \in \mathbb{Z}_p, a_1 \neq a_2, a_1^n = a_2, a_2^n = a_1\}$$

zugeordnet ist. Damit gilt $2|C| = |D|$.

Die entscheidende Beobachtung ist nun, dass die Zuordnung $\psi : (a_1, a_2) \mapsto a_1$ die Menge D injektiv in die Menge $E := \{a \in \mathbb{Z}_p \mid a^{n^2-1} \equiv 1 \bmod p\}$ abbildet. Für $(a_1, a_2) \in D$ gilt nämlich $a_1^{n^2} \equiv a_2^n \equiv a_1$, woraus $a_1^{n^2-1} \equiv 1$ folgt, weil $a_1 \not\equiv 0$ sein muss. Ist weiterhin $\psi(a_1, a_2) = \psi(a_1', a_2')$, so gilt $a_1 \equiv a_1' \bmod p$. Damit ergibt sich aber auch $a_2 \equiv a_1^n \equiv a_1'^n \equiv a_2' \bmod p$, also ist ψ injektiv. Das begründet $|D| \leq |E|$.

Nun ist $0 \notin E$, so dass $|E| \leq p - 1$. Insgesamt ergibt sich also $|A| \leq |B| \leq |C| = \frac{|D|}{2} \leq \frac{|E|}{2} \leq \frac{p-1}{2}$. □

Lemma 6.12. *Sei n quadratfrei und $n = p_1 p_2 \cdots p_k$ die Primfaktorzerlegung von n. Wir betrachten die Anzahl l der Paare (b, c) modulo n, für die gilt:*

1. *(n, b, c) ist zulässige Eingabe für den QFT,*
2. *QFT(n, b, c) erklärt n zum Primzahlkandidaten und*
3. *für einen Primfaktor $p|n$ gilt $\left(\frac{b^2+4c}{p}\right) = 1$.*

Es gilt

$$l < \begin{cases} \frac{n\varphi(n)}{2B}, & \text{falls } k \text{ gerade,} \\ \frac{n\varphi(n)}{B^2}, & \text{falls } k \text{ ungerade.} \end{cases}$$

Beweis. Sei $0 \leq b, c \leq n - 1$ und (n, b, c) eine zulässige Eingabe für den QFT, so dass QFT(n, b, c) die Zahl n zum Primzahlkandidaten erklärt. Wäre $k = 1$, müsste $p = p_1$ sein und $\left(\frac{b^2+4c}{p_1}\right) = 1$ gelten, aber gleichzeitig $\left(\frac{b^2+4c}{p_1}\right) = \left(\frac{b^2+4c}{n}\right) = -1$, damit (n, b, c) zulässige Eingabe für den QFT ist. Das kann nicht sein, also ist $k > 1$.

Wir unterscheiden nun den Fall, dass $\left(\frac{b^2+4c}{p_i}\right) = 1$ für genau ein $i \in \{1, \ldots, k\}$, und den Fall, dass dies für mindestens zwei verschiedene $i \in \{1, \ldots, k\}$ gilt.

Fall 1. Sei p_i der einzige Primfaktor von n, für den $\left(\frac{b^2+4c}{p_i}\right) = 1$. Weil $-1 = \left(\frac{b^2+4c}{n}\right) = \prod_{j=1}^{k} \left(\frac{b^2+4c}{p_j}\right) = (-1)^{k-1}$, muss k gerade sein.

Nach Lemma 6.11 gibt es modulo p_i höchstens $\frac{p_i-1}{2}$ Möglichkeiten für (b, c).

Modulo $p_j, j \neq i$, gibt es $\frac{p_j-1}{2} \cdot p_j$ Möglichkeiten für ein Paar (b,c) mit $\left(\frac{b^2+4c}{p_j}\right) = -1$. Dies ist wie folgt einzusehen: Wenn b modulo p_j beliebig vorgegeben ist, kann jeder der genau $\frac{p_j-1}{2}$ quadratischen Nichtreste r modulo p_j, die der Ausdruck b^2+4c annehmen kann, durch ein modulo p_j eindeutig bestimmtes c realisiert werden, nämlich durch $c \equiv 4^{-1}(r-b^2) \bmod p_j$.

Nun folgt mit dem Chinesischen Restsatz (Satz 3.40), dass es für festes i höchstens $\frac{p_i-1}{2}\prod_{j\neq i}\frac{p_j(p_j-1)}{2}$ mögliche (b,c) modulo n gibt. Das sind nach Satz 3.41 $\frac{n\varphi(n)}{p_i 2^k}$ Stück. Insgesamt gibt es also höchstens $\sum_{i=1}^{k}\frac{n\varphi(n)}{2^k p_i} < \frac{n\varphi(n)k}{2^k B}$ Paare, weil jeder Primfaktor von n größer als B sein muss.

Fall 2. Für $0 \leq b,c \leq n-1$ sei $L(b,c) := \left(\left(\frac{b^2+4c}{p_1}\right), \left(\frac{b^2+4c}{p_2}\right), \ldots, \left(\frac{b^2+4c}{p_k}\right)\right)$. Die Menge der k-Tupel aus $\{1,-1\}^k$, die an mindestens zwei Stellen den Wert 1 aufweisen, sei mit V bezeichnet. Sie ist identisch mit der Menge aller $L(b,c)$, wobei (b,c) die Menge der hier in Fall 2 betrachteten Paare (b,c) durchläuft.

Für ein Element $L \in V$ schätzen wir die Anzahl aller (b,c) mit $L(b,c) = L$ ab. Seien dazu $i < j$ die größten Indizes, so dass $L_i, L_j = 1$ ist[6]. Nach Lemma 6.11 gibt es modulo p_i höchstens $\frac{p_i-1}{2}$ Paare (b,c), so dass n den QFT mit (b,c) besteht. Entsprechendes gilt für p_j. Für l, $l \neq i, l \neq j$, gibt es mit dem gleichen Argument wie in Fall 1 modulo p_l höchstens $\frac{p_l(p_l-1)}{2}$ Paare (b,c), so dass n den QFT mit (b,c) besteht. Damit gibt es nach dem Chinesischen Restsatz modulo n nicht mehr als $\frac{p_i-1}{2}\frac{p_j-1}{2}\prod_{l\neq i,j}\frac{p_l(p_l-1)}{2} = \frac{n\varphi(n)}{p_i p_j 2^k}$ Paare (b,c) mit $L(b,c) = L$, so dass n den QFT mit (b,c) besteht.

Insgesamt ist damit die Anzahl aller Paar (b,c), für die $\left(\frac{b^2+4c}{p}\right) = 1$ für mehr als einen Primfaktor p von n ist, nicht größer als $\sum_{L\in V}\frac{n\varphi(n)}{2^k B^2} \leq \frac{n\varphi(n)}{2^k B^2}|V| < \frac{n\varphi(n)}{B^2}$, wobei wir ausnutzen, dass $|V| < |\{1,-1\}^k| = 2^k$ ist. Damit die Untersuchung von Fall 2 abgeschlossen.

Wir zeigen nun die Behauptung des Lemmas, indem wir die Erkenntnisse aus beiden Fällen kombinieren. Wenn $k = 2$ gilt, muss $\left(\frac{b^2+4c}{p_1}\right) = 1$ und $\left(\frac{b^2+4c}{p_2}\right) = -1$ oder umgekehrt sein, so dass Fall 2 nicht eintreten kann. Die Abschätzung aus Fall 1 liefert jedoch genau die Aussage des Lemmas für $k = 2$.

Ist $k > 2$ und k gerade, dann kann sowohl Fall 1, als auch Fall 2 eintreten, so dass die Gesamtzahl aller (b,c), für die n den QFT mit (b,c) besteht, kleiner als $n\varphi(n)\left(\frac{k}{2^k B}+\frac{1}{B^2}\right)$ ist. Mit Lemma A.1 aus dem Anhang ergibt sich hier die zu beweisende Aussage.

Falls $k > 2$ ungerade ist, kann Fall 1 nicht eintreten und die Aussage des Lemmas ergibt sich aus der Abschätzung im Fall 2. □

[6] Wie üblich bezeichnet L_i die i-te Komponente des Vektors L.

Hat n eine gerade Anzahl von Primfaktoren und ist (n, b, c) eine zulässige Eingabe für den QFT, dann muss für einen Primfaktor p von n die Aussage $\left(\frac{b^2+4c}{p}\right) = 1$ gelten, weil sonst $\left(\frac{b^2+4c}{n}\right) = 1$ wäre. Damit folgt aus Lemma 6.12, Satz 6.4 und der Abschätzung $n\varphi(n) < n^2$ die Fehlerschranke für quadratfreie Zahlen mit gerader Anzahl Primfaktoren:

Satz 6.13. *Eine quadratfreie ganze Zahl mit einer geraden Anzahl Primfaktoren besteht den RQFT mit Wahrscheinlichkeit kleiner als* $\frac{2}{B}$.

Im Rest dieses Abschnitts beschäftigen wir uns mit quadratfreien Zahlen mit einer ungeraden Anzahl von Primfaktoren. Sei n eine solche Zahl und $n = p_1 p_2 \cdots p_k$ deren Primfaktorzerlegung. Um die Irrtumswahrscheinlichkeit des RQFT auf n abzuschätzen, müssen wir die Anzahl der (b, c) abschätzen, für die der QFT n zum Primzahlkandidaten bezüglich (b, c) erklärt. Die Anzahl der Paare mit $\left(\frac{b^2+4c}{p_i}\right) = 1$ für wenigstens ein $p_i | n$ wird durch Lemma 6.12 ausreichend beschränkt. Mit den Paaren (b, c), für die $\left(\frac{b^2+4c}{p_i}\right) = -1$ für alle $p_i | n$ gilt, werden wir uns im Rest dieses Abschnittes beschäftigen.

Wir definieren J als die größte ganze Zahl, für die gilt

$$2^{J+1} \mid \operatorname{ggT}(p_1^2 - 1, p_2^2 - 1, \ldots, p_k^2 - 1).$$

Es gilt $J \geq 2$ (Lemma A.2 aus dem Anhang auf $p_1, \ldots, p_k$ angewendet). Für jedes $i = 1, \ldots, k$ gilt $p_i^2 \equiv 1 \bmod 2^{J+1}$, also ist $n^2 \equiv 1 \bmod 2^{J+1}$. Demzufolge ist $2^{J+1} \mid n^2 - 1$, so dass $J + 1 \leq r$ gilt (r — und auch s, das im folgenden Lemma verwendet wird — sind in Zeile 7 von Algorithmus 8 definiert).

Lemma 6.14. *Für $j \geq J$ gibt es* keine *Paare (b, c), für die gilt:*

1. *der QFT erklärt n zum Primzahlkandidaten bezüglich (b, c),*
2. $\left(\frac{b^2+4c}{p}\right) = -1$ *für alle $p | n$ und*
3. $X^{2^j s} \equiv -1 \bmod (n, X^2 - bX - c)$.

Beweis. Wir nehmen an, dass es ein entsprechendes Paar (b, c) gibt, und wählen einen Primteiler $p | n$, für den $2^{J+2} \nmid p^2 - 1$ (möglich nach Definition von J). Aus $X^{2^j s} \equiv -1 \bmod (n, X^2 - bX - c)$ folgt $X^{2^j s} \equiv -1 \bmod (p, X^2 - bX - c)$. Der der Wert $y := X \bmod (p, X^2 - bX - c)$ ist folglich eine Lösung der Gleichung $y^{2^j s} = -1$ in $\mathbb{Z}_p[X]/(X^2 - bX - c) \simeq \mathbb{F}_{p^2}$. Nach Lemma 3.47 muss dann $2^{j+1} \mid p^2 - 1$ gelten. Für $j > J$ ist das ein Widerspruch zu $2^{J+2} \nmid p^2 - 1$.

Es bleibt noch der Fall $j = J$ auszuschließen. Da der QFT n zum Primzahlkandidaten bezüglich (b, c) erklärt, muss wegen Zeile 2 und Zeile 4 in Algorithmus 8 $-c$ modulo n eine Quadratzahl sein, also auch modulo p. Wie bei der Herleitung von (6.8) ergibt sich nun, dass $X^{(p^2-1)/2} \equiv 1 \bmod (p, X^2 - bX - c)$. Daraus folgt, dass 2^J die Ordnung von x in $\left((\mathbb{Z}_p[X]/(X^2 - bX - c))^*, \cdot\right)$ teilt, aber 2^{J+1} nicht. Deshalb kann nicht $x^{2^J s} \equiv -1$ modulo $(p, X^2 - bX - c)$ gelten, denn dann wäre 2^{J+1} ein Teiler der Ordnung von x. □

Satz 6.15. *Sei n quadratfrei mit Primfaktorzerlegung $n = p_1 p_2 \cdots p_k$, k ungerade. Dann ist die Wahrscheinlichkeit, dass n den RQFT besteht, kleiner als $\frac{1}{2^{3k-2}} + \frac{1}{2^{4k-3}} + \frac{4}{B^2}$.*

Beweis. Wir betrachten die Paare (b, c) mit $\left(\frac{b^2+4c}{p}\right) = -1$ für alle $p|n$. Um die Aussage des Satzes zu beweisen, genügt es nach Lemma 6.12 zu zeigen, dass die Anzahl dieser Paare nicht größer als $n^2(2^{-3k} + 2^{-4k+1})$ ist. Dies wollen wir im Folgenden tun, so dass wir ab jetzt $\left(\frac{b^2+4c}{p}\right) = -1$ für alle $p|n$ annehmen. Nach den Betrachtungen aus Abschnitt 3.7.2 ist folglich für jedes $i = 1, \ldots, k$ der Ring $\mathbb{Z}_{p_i}[X]/(X^2 - bX - c)$ isomorph zum Körper $\mathbb{F}_{p_i^2}$.

Wir setzen $G := \prod_{p_i|n} \mathrm{ggT}(p_i^2 - 1, s)$. Wenn der QFT n zum Primzahlkandidaten bezüglich (b, c) erklärt, muss wegen Zeile 8 des QFT und Satz 3.40

$$X^s \equiv 1 \bmod (p_i, X^2 - bX - c) \quad \text{für alle } p_i|n \tag{6.16}$$

oder für ein $j, 0 \leq j \leq r - 2$,

$$X^{2^j s} \equiv -1 \bmod (p_i, X^2 - bX - c) \quad \text{für alle } p_i|n \tag{6.17}$$

gelten. Wir wenden nun Satz 3.40, Lemma 3.61 mit $g = X^s - 1$ sowie Lemma 3.48 an. Die Anzahl der (b, c) modulo n, für die (6.16) gilt, ist also nicht größer als

$$\prod_{1 \leq i \leq k} \mathrm{ggT}(s, p_i^2 - 1)/2 = G/2^k.$$

Satz 3.40, Lemma 3.61 mit $g = X^{2^j s} + 1$ sowie Lemma 3.47 ergeben, dass für $j, 0 \leq j < J$, die Anzahl der Paare (b, c) modulo n, für die (6.17) zutrifft, nicht größer ist als

$$\prod_{1 \leq i \leq k} \mathrm{ggT}(2^j s, p_i^2 - 1)/2 \overset{\text{Satz 3.31, 2.}}{=} \prod_{1 \leq i \leq k} 2^j \,\mathrm{ggT}(s, p_i^2 - 1)/2 = G 2^{k(j-1)}.$$

Für $j \geq J$ gibt es nach Lemma 6.14 keine (b, c), so dass (6.17) gilt.

Zusammengenommen ist die Anzahl aller Paare (b, c), für die (6.17) für ein j, $0 \leq j \leq r-2$, oder (6.16) erfüllt ist, beschränkt durch

$$G/2^k + \sum_{0 \leq j < J} G2^{k(j-1)} = \Big(1 + \frac{2^{kJ}-1}{2^k-1}\Big) G/2^k.$$

Nun schätzen wir noch G ab. Da s ungerade ist, gilt

$$\mathrm{ggT}(p_i^2 - 1, s) = \mathrm{ggT}\left((p_i^2-1)/2^{J+1}, s\right) \leq (p_i^2-1)/2^{J+1} < p_i^2/2^{J+1},$$

so dass die Anzahl der Paare kleiner ist als $(1 + \frac{2^{Jk}-1}{2^k-1})\frac{n^2}{2^k 2^{(J+1)k}}$. Mit Lemma A.3 aus dem Anhang und Satz 6.4 folgt die zu zeigende Aussage. □

Für $k = 3$ ist die Fehlerschranke von Satz 6.15 größer als $1/128$. Deshalb führen wir speziell für $k = 3$ noch eine genauere Rechnung durch.

Satz 6.16. *Sei $n = p_1 p_2 p_3$ für drei paarweise verschiedene Primzahlen p_1, p_2, p_3. Dann ist die Wahrscheinlichkeit, dass n den RQFT besteht, kleiner als $\frac{4}{B^2} + \frac{3(B^2+1)}{2(B^4-3B^2)}$.*

Wenn dieser Satz bewiesen ist, haben wir in jedem Fall gezeigt, dass eine Iteration des RQFT mit Wahrscheinlichkeit $< \frac{1}{7710}$ eine zusammengesetzte Zahl als Primzahl identifiziert. Welcher Satz jeweils anzuwenden ist, kann man Tabelle 6.1 entnehmen.

Beweis von Satz 6.16. Wegen Lemma 6.12 genügt es zu zeigen, dass modulo n die Anzahl der Paare (b, c), für die n den QFT mit (b, c) besteht und für die $\left(\frac{b^2+4c}{p}\right) = -1$ für alle $p|n$ gilt, den Wert $\frac{3(B^2+1)}{2(B^4-3B^2)} \cdot \frac{n^2}{4}$ nicht überschreitet. Sei daher (b, c) ein solches Paar.

Für $i = 1, 2, 3$ gilt: Wegen Zeile 4 des QFT muss $x^{n+1} = -c$ modulo $(n, X^2 - bX - c)$ sein, also auch modulo $(p_i, X^2 - bX - c)$. Weil p_i Primzahl ist und $\left(\frac{b^2+4c}{p_i}\right) = -1$ ist, gilt $x^{p_i+1} = -c$ modulo $(p_i, X^2 - bX - c)$ (gleiches Argument wie beim Nachweis der Tatsache, dass Primzahlen vom QFT stets zu Primzahlkandidaten erklärt werden). Wegen $\left(\frac{-c}{n}\right) \neq 0$ gilt $-c \not\equiv 0 \bmod p_i$, also ist $-c$ invertierbar. Deshalb folgt aus $x^{n+1} = -c$ und $x^{p_i+1} = -c$ die Gleichung

$$X^{n-p_i} \equiv 1 \bmod (p_i, X^2 - bX - c).$$

Mit $g = X^{n-p_i}$ folgt aus Lemma 3.61, dass modulo p_i die Anzahl der betrachteten Paare (b, c) höchstens halb so groß ist wie die Anzahl der Lösungen von $y^{n-p_i} = 1$

Struktur von n	Satz	Fehlerschranke	numerisch, $B = 50000$
n nicht quadratfrei	6.10	$4/B$	$8 \cdot 10^{-5}$
n quadratfrei und hat gerade Anzahl Primfaktoren	6.13	$2/B$	$4 \cdot 10^{-5}$
n quadratfrei und hat genau 3 Primfaktoren	6.16	$\frac{4}{B^2} + \frac{3(B^2+1)}{2(B^4-3B^2)}$	$< 2.221 \cdot 10^{-9}$
n quadratfrei und hat $k \geq 5$ Primfaktoren	6.15	$\frac{1}{2^{3k-2}} + \frac{1}{2^{4k-3}} + \frac{4}{B^2}$	$< 1.2970131 \cdot 10^{-4}$

Tabelle 6.1.: Die Irrtumswahrscheinlichkeit des randomisierten quadratischen Frobenius-Test ist stets kleiner als $\frac{1}{7710} > 1.297016 \cdot 10^{-4}$.

in $\mathbb{F}_{p_i^2}$. Wegen Satz 3.42 und Satz 3.24 hat diese Gleichung genau $\mathrm{ggT}(n - p_i, p_i^2 - 1)$ Lösungen in $\mathbb{F}_{p_i^2}$. Wir setzen $k_i = \mathrm{ggT}(n - p_i, p_i^2 - 1)$.

Damit und mit Satz 3.40 ergibt sich nun, dass modulo n die Anzahl der betrachteten Paare (b, c) höchstens $k_1 k_2 k_3/8$ ist. Der Rest dieses Beweises beschäftigt sich damit, diesen Ausdruck abzuschätzen.

Zunächst gilt $k_i = \mathrm{ggT}(n - p_i, p_i^2 - 1) = \mathrm{ggT}(n/p_i - 1, p_i^2 - 1)$, weil $p_i \nmid p_i^2 - 1$. Wir setzen $j_i := (p_i^2 - 1)/k_i$, $r_i := (n/p_i - 1)/k_i$ und $C = j_1 j_2 j_3$. Es gilt

$$\begin{aligned} k_1 k_2 k_3/8 &= (p_1^2 - 1)(p_2^2 - 1)(p_3^2 - 1)/(8C) \\ &= (n^2 - p_1^2 p_2^2 - p_1^2 p_3^2 - p_2^2 p_3^2 + p_1^2 + p_2^2 + p_3^2 - 1)/(8C) \\ &= (n^2 - p_1^2(p_2^2 - 1) - p_3^2(p_1^2 - 1) - p_2^2(p_3^2 - 1) - 1)/(8C) \\ &< \frac{n^2}{8C} < \frac{1}{2C} Z(n). \end{aligned}$$

Zur Abschätzung von $1/C$ unterscheiden wir drei Fälle.

Fall 1: $r_1 r_2 r_3 > C$**.** Weil $r_1 r_2 r_3$ eine ganze Zahl ist, gilt dann $\frac{r_1 r_2 r_3}{j_1 j_2 j_3} \geq \frac{C+1}{C} = 1 + \frac{1}{C}$.

Wenn wir r_i und j_i durch ihre Definition ersetzen, erhalten wir

$$\begin{aligned}\frac{r_1r_2r_3}{j_1j_2j_3} &= \frac{(p_2p_3-1)(p_1p_3-1)(p_1p_2-1)}{(p_1^2-1)(p_2^2-1)(p_3^2-1)} \\ &= \frac{n^2-p_1p_2p_3^2-p_1p_2^2p_3-p_1^2p_2p_3+p_2p_3+p_1p_3+p_1p_2-1}{n^2-p_1^2p_2^2-p_1^2p_3^2-p_2^2p_3^2+p_1^2+p_2^2+p_3^2-1} \\ &< \frac{n^2+p_1p_2+p_1p_3+p_2p_3}{n^2-p_1^2p_2^2-p_1^2p_3^2-p_2^2p_3^2} = \frac{1+(p_1p_2+p_1p_3+p_2p_3)/n^2}{1-(p_1^2p_2^2+p_1^2p_3^2+p_2^2p_3^2)/n^2} \\ &= \frac{1+1/(p_1p_2p_3^2)+1/(p_1p_2^2p_3)+1/(p_1^2p_2p_3)}{1-(1/p_3^2+1/p_2^2+1/p_1^2)} < \frac{1+3/B^4}{1-3/B^2}.\end{aligned} \tag{6.18}$$

Also gilt

$$1+\frac{1}{C} < \frac{1+3/B^4}{1-3/B^2} = \frac{B^4+3}{B^4-3B^2},$$

daraus folgt

$$\frac{1}{C} < \frac{3B^2+3}{B^4-3B^2}.$$

Damit ist im ersten Fall die Behauptung des Satzes gezeigt.

Fall 2: $r_1r_2r_3 < C$**.** Hier gilt $\frac{r_1r_2r_3}{j_1j_2j_3} \leq 1-\frac{1}{C}$ und

$$\frac{r_1r_2r_3}{j_1j_2j_3} > \frac{n^2-p_1p_2p_3^2-p_1p_2^2p_3-p_1^2p_2p_3}{n^2+p_1^2+p_2^2+p_3^2} > \frac{1-3/B^2}{1+3/B^4},$$

woraus $\frac{1}{C}-1 < \frac{3B^2-B^4}{B^4+3}$ und somit

$$\frac{1}{C} < \frac{3B^2+3}{B^4+3} < \frac{3B^2+3}{B^4-3B^2}$$

folgt. Damit ist die Behauptung auch im zweiten Fall gezeigt.

Fall 3: $r_1r_2r_3 = C$**.** Wir zeigen, dass dieser Fall nicht eintreten kann, weil er einen Widerspruch impliziert. Setzen wir nämlich $r_1r_2r_3 = j_1j_2j_3$ in (6.18) ein, so folgt einerseits

$$(p_2p_3-1)(p_1p_3-1)(p_1p_2-1) = (p_1^2-1)(p_2^2-1)(p_3^2-1).$$

Andererseits gilt jedoch $(p_2p_3-1)^2 > (p_2^2-1)(p_3^2-1)$ und entsprechende Ungleichungen für p_1, p_2 und p_1, p_3. Multipliziert man diese und zieht die Wurzel, so ergibt sich

$$(p_2p_3-1)(p_1p_3-1)(p_1p_2-1) > (p_1^2-1)(p_2^2-1)(p_3^2-1). \quad \square$$

„Irrtumswahrscheinlichkeit“ des einfachen quadratischen Frobenius-Tests

Wenn man die Beweise für die Abschätzung der Fehlerwahrscheinlichkeit des RQFT noch einmal anschaut, stellt man fest, dass nur für den Beweis von Satz 6.15 mehr als die Bedingung $x^{n+1} = -c$ benutzt wurde. Im Beweis von Satz 6.10 wird zwar $x^{n^2-1} = 1$ genutzt, das folgt aber in jedem Fall aus $x^n = b - x$, wie wir aus Lemma 6.2 schließen können. Damit kann man sagen, dass der einfache quadratische Frobenius-Test, der QFT und auch der RQFT im Wesentlichen deshalb gut Primzahlen von zusammengesetzten Zahlen unterscheiden können, weil in allen diesen Tests die Bedingung $x^n \stackrel{?}{=} b - x$ — also (6.1) — geprüft wird. Für viele zusammengesetzte n würde dieser Test — also der einfache quadratische Frobenius-Test — ausreichen, um mit großer Wahrscheinlichkeit n als zusammengesetzt zu erkennen. Allein für quadratfreie n mit $5, 7, 9, \ldots$ Primfaktoren konnte die Irrtumswahrscheinlichkeit von $< 1/7710$ für den RQFT nur nachgewiesen werden, weil der QFT auch noch nach nichttrivialen Quadratwurzeln von 1 sucht und den zusätzlichen Test in Zeile 2 durchführt. Zahlen n mit der erwähnten Struktur sind auch der Grund, warum eine Vergrößerung von B nicht automatisch zu einer Verringerung der Irrtumswahrscheinlichkeit des RQFT führt (vgl. Tabelle 6.1).

6.6. Laufzeit

Neben einer geringen Irrtumswahrscheinlichkeit muss ein Primzahltest auch effizient ausführbar sein, damit er für die Praxis nützlich ist. Deshalb diskutieren wir in diesem Abschnitt die Laufzeit des randomisierten starken quadratischen Frobenius-Tests. Dazu werden wir einerseits die Laufzeitanalyse aus [Gra98] nachvollziehen und kritisch beleuchten. Außerdem werden wir eine etwas schnellere Implementierungsmöglichkeit kennenlernen, indem wir die Implementierung des einfachen quadratischen Frobenius-Test aus [CP01] für den RQFT nutzbar machen.

Ein Lauf des RQFT auf n kann darin bestehen, dass ein Paar (b, c) gesucht und gefunden wird, so dass (n, b, c) zulässige Eingabe für den QFT ist, und anschließend der QFT gestartet wird. Die zweite Möglichkeit ist, dass ein Paar (b, c) gefunden wird, das einen Faktor von n offenbart. Schließlich ist es auch möglich, dass B Paare (b, c) gewählt und verworfen werden und der Algorithmus danach anhält. Alle diese Tätigkeiten

benötigen eine gewisse Laufzeit. Wir wollen uns in diesem Abschnitt hauptsächlich mit der Laufzeit des QFT beschäftigen. Der Zeitbedarf der übrigen Aktionen wird in Unterabschnitt 6.6.3 mit diskutiert.

Die Laufzeit des QFT wird im Wesentlichen davon bestimmt, dass in der Struktur $\mathbb{Z}_n[X]/(X^2 - bX - c)$ mehrmals bestimmte Potenzen x^t von x untersucht werden. Deshalb ist es entscheidend, wie schnell diese Potenzen ausgerechnet werden können. Satz 4.7 in Verbindung mit Satz 4.11 liefert sofort, dass eine solche Potenz in $5 \log t + o(\log t)$ Multiplikationen modulo n berechnet werden kann. Damit ist die behauptete Laufzeitschranke von 3 Miller-Rabin-Tests noch nicht erreicht (vgl. Satz 5.3), weil schon die Berechnung einer einzelnen Potenz fünf mal so lange dauert wie ein kompletter Miller-Rabin-Test. Zwei Ideen, die dieses Problem lösen, sind in [Gra98] dargelegt: Zum einen wird die Berechnung von Potenzen x^t beschleunigt. Zum andern werden aus der ersten berechneten Potenz mit geringem Aufwand die weiteren benötigten Potenzen gewonnen.

Bevor wir uns mit beiden Schritten jeweils in einem eigenen Unterabschnitt beschäftigen, müssen wir uns noch klar machen, wie wir die Laufzeit messen wollen (vgl. Abschnitt 4.1). Wir werden dazu im Wesentlichen die Zahl der Multiplikationen modulo n zählen. Gleichzeitig wollen wir jedoch auch die Anzahl der Additionen, Subtraktionen, Zuweisungen und Vergleiche in $\mathbb{Z}_n$ nicht unberücksichtigt lassen. Die genaue Anzahl ist für uns aber nicht entscheidend, weil wir wissen, dass diese Operationen nicht ins Gewicht fallen, solange sie nicht wesentlich häufiger auftreten als Multiplikationen. Gelegentlich müssen wir in $\mathbb{Z}_n$ auch invertieren. Auf Invertierungen müssen wir besonders Acht geben, weil eine Invertierung — abhängig von der Implementierung — asymptotisch mehr Zeit benötigen kann als eine Multiplikation. Aus diesem Grund übernehmen wir auch nicht das Zeitmaß „selfridge“ aus [Gra98], weder in der dort angegebenen Form, noch modifiziert. Denn dann müssten wir uns auf ein Verhältnis *festlegen*, nach dem wir jede Invertierung in eine zeitlich äquivalente Anzahl von Multiplikationen umrechnen.

6.6.1. Potenzen x^t bestimmen

Satz 4.11 bietet die Möglichkeit, die asymptotische Laufzeit für Potenzierungen zu verbessern, wenn man Quadrierungen um einen konstanten Faktor beschleunigt. Dies darf — ebenfalls bis zu einem konstanten Faktor — zu Lasten der sonstigen Multiplika-

tionen geschehen. Ein Ansatz dazu wäre, in Gleichung (4.1) $u_0 = v_0$ und $u_1 = v_1$ zu setzen und so im Fall der Quadrierung auf eine Verbesserung von Satz 4.7 zu hoffen. Leider hat dieser Ansatz zu keinem Erfolg geführt. In [Gra98] wird deshalb anders vorgegangen. Anstatt nach dem in Abschnitt 4.3 beschriebenen Verfahren viele kleine Potenzen von x zu x^t zusammenzusetzen, wird ein Element aus einer anderen Struktur M zur t-ten Potenz erhoben, aus der dann mit geringem Aufwand x^t bestimmt werden kann. Die Struktur M hat die hilfreiche Eigenschaft, dass dort „Quadrieren“ nur 3 Multiplikationen modulo n erfordert. Sonstige „Multiplikationen“ in M erfordern zwar jeweils 8 Multiplikationen modulo n, aber dieser Faktor macht sich asymptotisch nicht bemerkbar, weil die Anzahl dieser Multiplikationen $\frac{2\log t}{\log\log t} + \frac{1}{2}(\log t)^{3/5} = o(\log t)$ ist. Damit wird asymptotisch die Laufzeit des Miller-Rabin-Tests eingehalten.

Nachdem die Idee erläutert ist, können wir uns nun an die Details wagen. Wir erinnern uns an Abschnitt 3.8, wo es um die Lucas-Folgen ging, und definieren

$$M := \Big\{(V_j, U_j, C_j) \mid V_j = x^j + (b-x)^j, U_j = \frac{x^j - (b-x)^j}{x - (b-x)}, C_j = c^j, j = 0, 1, 2, \dots\Big\},$$

wobei $V_j = V_j(b, c)$ und $U_j = U_j(b, c)$ Glieder der Lucas-Folgen sind (siehe Abschnitt 3.8), und

$$(V_j, U_j, C_j) \circ (V_k, U_k, C_k) := (V_{j+k}, U_{j+k}, C_{j+k}).$$

Selbstverständlich ist $\circ$ assoziativ, denn $+$ ist assoziativ auf $\mathbb{N}$. (V_0, U_0, C_0) ist ein neutrales Element bezüglich $\circ$. Die nächsten beiden Sätze geben die Komplexität von Quadrierungen und sonstigen Multiplikationen in $(M, \circ)$ an.

Satz 6.17. *Seien $(V_j, U_j, C_j), (V_k, U_k, C_k) \in M$ sowie $2^{-1} \bmod n$ und $\Delta = b^2 + 4c \bmod n$ gegeben. Dann gilt:*

1. *$(V_j, U_j, C_j) \circ (V_k, U_k, C_k)$ kann mit 8 Multiplikationen modulo n und zwei Additionen modulo n bestimmt werden.*

2. *$(V_j, U_j, C_j)^2 := (V_j, U_j, C_j) \circ (V_j, U_j, C_j)$ kann mit 3 Multiplikationen und zwei Additionen bzw. Subtraktionen modulo n berechnet werden.*

3. *x^j lässt sich aus (V_j, U_j, C_j) mit 2 Multiplikationen und einer Subtraktion modulo n ermitteln.*

Beweis. Die Berechnung von U_{j+k} nach Gleichung (3.19) benötigt 3 Multiplikationen und eine Addition, die Berechnung von V_{j+k} nach (3.20) 4 Multiplikationen und eine Addition und die Berechnung von $C_{j+k} = C_j C_k$ eine Multiplikation modulo n.

Nach (3.21) kann U_{2j} mit einer Multiplikation modulo n berechnet werden. Falls j gerade ist, kann wegen (3.22) V_{2j} berechnet werden als $V_{2j} = V_j^2 - c - c$, also mit einer Multiplikation und zwei Subtraktionen modulo n. Entsprechend sind bei ungeradem j eine Multiplikation und zwei Additionen modulo n ausreichend. $C_{2j} = C_j^2$ lässt sich mit einer Multiplikation modulo n ermitteln.

Nach (3.18) ist der lineare Koeffizient von x^j durch U_j gegeben, steht also ohne Rechnung fest, und der konstante Koeffizient kann durch zwei Multiplikationen und eine Subtraktion modulo n bestimmt werden. □

Nun ergibt sich aus Satz 6.17 und Satz 4.11 die in [Gra98] angegebene Laufzeit für die Berechnung von x^t:

Satz 6.18. *Ist* $2^{-1} \bmod n$ *gegeben, so kann* x^t *mittels* $3\log t + o(\log t)$ *Multiplikationen modulo* n *und* $2\log t + o(\log t)$ *Additionen bzw. Subtraktionen modulo* n *bestimmt werden.*

Verbesserung der Laufzeit

Bei Experimenten stellt sich heraus, dass eine Frobenius-Test-Iteration [Gra98] entsprechend implementiert die Zeit von etwa 4 Miller-Rabin-Test-Iterationen benötigt (siehe Kapitel 7, Implementierungsvarianten `sfrob_gr01` und `frob_gr01`). Wie ist dies zu erklären? Wenn wir die Laufzeiten für eine Quadrierung und eine sonstige Multiplikation aus Satz 6.17 noch einmal genau in Satz 4.11 einsetzen, ergeben sich

$$\begin{aligned} &3(\log t + O(\log\log t)) + 8\left(\frac{2\log t}{\log\log t} + \frac{1}{2}(\log t)^{3/5}\right) \\ =&3\log t + \frac{16}{\log\log t}\log t + 4(\log t)^{3/5} + O(\log\log t) \end{aligned} \tag{6.19}$$

Multiplikationen modulo n zur Bestimmung von x^t. Die experimentell untersuchten Zahlen haben nicht mehr als 4096 Bit, so dass auch t in dieser Größenordnung liegt. Solange aber die Bitlänge von t kleiner als $2^{16} = 65536$ ist (was in der Primzahltest-Praxis utopisch ist), nimmt der Ausdruck $16/\log\log t$ noch mindestens den Wert 1 an, so die Anzahl der Multiplikationen modulo n mehr als $4\log t$ beträgt. Das Ziel für den

Rest dieses Abschnittes ist nun zu zeigen, wie man mit

$$3\log t + \frac{2\log t}{\log\log t} + \frac{1}{2}(\log t)^{3/5} + O(\log\log t) \tag{6.20}$$

Multiplikationen modulo n und vier Invertierungen modulo n zur Bestimmung von x^t auskommt. Das sind dann sogar ein paar Multiplikationen weniger als bei drei Miller-Rabin-Iterationen, die $3\log t + \frac{6\log t}{\log\log t} + \frac{3}{2}(\log t)^{3/5} + O(\log\log t)$ Multiplikationen benötigen. Allerdings kommt der Miller-Rabin-Test ohne Invertierungen aus.

Die Laufzeitschranke (6.20) wird dadurch erreicht, dass die Methode aus [CP01, Abschnitt 3.5.3] für Algorithmus 8 nutzbar gemacht wird. Wie bereits erwähnt wurde, wird in [CP01, Abschnitt 3.5] der einfache quadratischen Frobenius-Test behandelt, der ausschließlich darin besteht, für geeignete b, c die Bedingung $x^n = b - x$ zu prüfen. Es wird gezeigt, dass diese Bedingung genau dann erfüllt ist, wenn die Glieder $U_{n-\delta}, V_{n-\delta}$ der Lucas-Folgen bestimmte Werte annehmen, wobei $\delta = \left(\frac{b^2+4c}{n}\right)$ [CP01, Theorem 3.5.6]. Um dies effizient prüfen zu können, wird in [CP01, Abschnitt 3.5.3] eine Methode ausgearbeitet, mit der man schnell, nämlich mittels $2\log t + O(1)$ Multiplikationen modulo n und einer Invertierung modulo n, $V_t(b, c)$ und $U_t(b, c)$ bestimmen kann, wenn man $c^{t/2}$ zur Verfügung hat — allerdings nur für gerade t, was für [CP01, Theorem 3.5.6] ja auch ausreichend ist, weil $n - \delta$ gerade ist. In Experimenten hat sich gezeigt, dass mit dieser Implementierung eine Iteration des vereinfachten quadratischen Frobenius-Test tatsächlich nur die Zeit von drei Iterationen des Miller-Rabin-Test benötigt (siehe Kapitel 7, Implementierungsvarianten `frob_pn01` und `frob_pn02`). So lag es nahe, im Rahmen der vorliegenden Diplomarbeit diese Methode auch für Algorithmus 8 nutzbar zu machen und so dessen Effizienz zu steigern. Da ohnehin in [Gra98] vorgeschlagen wird, x^t aus V_t und U_t zu gewinnen (vgl. Satz 6.17, 3.), musste nur noch ein Weg gefunden werden, auch für ungerade t die Glieder U_t, V_t schnell zu bestimmen. Wie dies möglich ist, wird gleich erläutert, denn wir beschäftigen uns jetzt im Detail mit dem versprochenen schnelleren Verfahren zur Bestimmung von x^t.

Zunächst folgen wir [CP01, Abschnitt 3.5.3] und betrachten die Struktur

$$N := \Big\{(V_j, V_{j+1}) \mid V_j := X^j - (B - X)^j \bmod (n, X^2 - BX + 1),\ j = 0, 1, 2, \ldots\Big\}.$$

Zu beachten ist, dass bei N Glieder der auf $b = B$ und $c = -1$ eingeschränkten Lucas-Folge $\big(V_n(B, -1)\big)_n$ beteiligt sind, während bei M Glieder der allgemeinen Lucas-Folge $\big(V_n(b, c)\big)_n$ genutzt werden. Wir wollen Elemente (V_j, V_{j+1}) aus N nach dem gleichen Prinzip wie beim schnellen Potenzieren (Algorithmus 3) ausrechnen. Dort wird ja die

Potenz ausgehend von einem Startwert ermittelt, indem mehrmals von $\mathtt{b} = a^k$ entweder zu $\mathtt{b}^2 = a^{2k}$ oder zu $\mathtt{b}^2 \cdot a = a^{2k+1}$ übergegangen wird. Auch in N sind diese Schritte möglich, sogar sehr effizient. Aus (3.23) erhalten wir nämlich, dass

$$(V_{2j}, V_{2j+1}) = (V_j^2 - 2, V_j V_{j+1} - B) \text{ und} \tag{6.21}$$

$$(V_{2j+1}, V_{2j+2}) = (V_j V_{j+1} - B, V_{j+1}^2 - 2). \tag{6.22}$$

Man kann also mit jeweils 2 Multiplikationen sowie 2 Subtraktionen modulo n von (V_j, V_{j+1}) zu (V_{2j}, V_{2j+1}) bzw. (V_{2j+1}, V_{2j+2}) gelangen. (V_0, V_1) steht ohne Berechnung fest, (V_1, V_2) kann dann mit einer Multiplikation und einer Subtraktion modulo n berechnet werden. Mit (V_1, V_2) beginnend benötigt Algorithmus 3 noch $\lfloor \log j \rfloor$ Schleifendurchläufe zur Bestimmung von (V_j, V_{j+1}). Somit gilt:

Satz 6.19. *Ist B gegeben, so kann das Paar $\big(V_j(B,-1), V_{j+1}(B,-1)\big)$ berechnet werden mit $2\lfloor \log j \rfloor + 1$ Multiplikationen modulo n und $2\lfloor \log j \rfloor + 1$ Subtraktionen modulo n.*

Bis zu dieser Stelle sind wir genau der Strategie aus [CP01, Abschnitt 3.5.3] gefolgt. Dort werden nun $U_{n-\delta}, V_{n-\delta}$ Abbildung 6.2 folgend ermittelt[7], wobei Satz 6.19 den Schritt von B zu $V_m(B,-1), V_{m+1}(B,-1)$ begründet. Diesen wichtigen Schritt übernehmen wir, gehen jedoch danach etwas anders vor, nämlich wie in Abbildung 6.3 dargestellt. Laufzeit und Details finden sich im Beweis des folgenden Satzes.

Satz 6.20. *Sei $t > 1$ eine natürliche Zahl und $m = \lfloor \frac{t}{2} \rfloor$. Weiter seien $B = (b^2 + 2c)c^{-1}$, das Paar $\big(V_m(B,-1), V_{m+1}(B,-1)\big)$ und der Wert $c^m \bmod n$ gegeben. Dann lässt sich $x^t = X^t \bmod (n, X^2 - bX - c)$ berechnen mittels 3 Invertierungen modulo n, 9 Multiplikationen modulo n und 4 Additionen bzw. Subtraktionen modulo n.*

Beweis. Wir folgen dem Schema aus Abbildung 6.3: Zunächst wenden wir zwei mal (3.25) an und erhalten $V_{2m}(b,c)$ und $V_{2m+2}(b,c)$. Das kostet 3 Multiplikationen. Dann bestimmen wir mit (3.26) $V_{2m+1}(b,c)$. Das kostet eine Invertierung, 2 Multiplikationen und eine Subtraktion. Je nachdem, ob $t = 2m$ oder $t = 2m+1$ ist, bestimmen wir nun $U_{2m}(b,c)$ bzw. $U_{2m+1}(b,c)$ mittels (3.24). Das kostet eine Invertierung, 2 Multiplikationen, eine Subtraktion und eine Addition ($2V_{m+1} = V_{m+1} + V_{m+1}$). Schließlich liefert (3.18) x^t. Dies kostet eine Invertierung, 2 Multiplikationen und eine Subtraktion. □

[7] Dies ist etwas vereinfacht. Tatsächlich werden in [CP01, Abschnitt 3.5.3] die in Abbildung 6.2 dargestellten Zusammenhänge ausgearbeitet und fließen in [CP01, Theorem 3.5.8] ein. [CP01, Algorithmus 3.5.9], der den einfachen quadratische Frobenius-Test implementiert, gründet sich dann auf dieses Theorem.

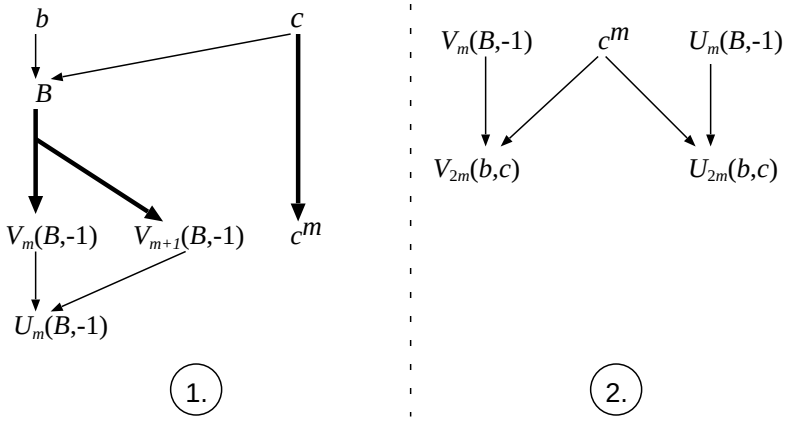

Abbildung 6.2.: Strategie aus [CP01, Abschnitt 3.5.3] zur effizienten Berechnung von $U_{n-\delta}$ und $V_{n-\delta}$, wobei $m := (n-\delta)/2$.

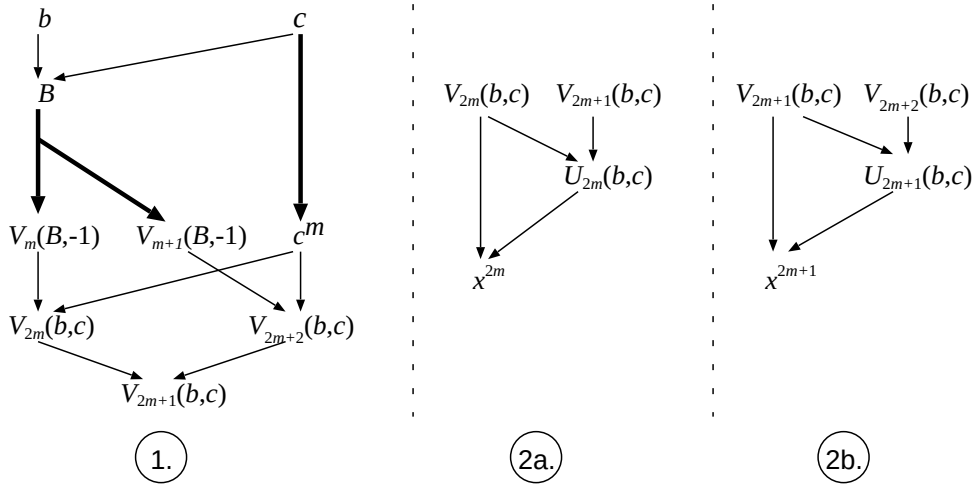

Abbildung 6.3.: Effiziente Berechnung von $x^t = x^{2m}$ bzw. $x^t = x^{2m+1}$.

Die Berechnung von c^m ist nach Satz 4.11 mit $\log m + \frac{2\log m}{\log\log m} + \frac{1}{2}(\log m)^{3/5} + O(\log\log m)$ Multiplikationen modulo n möglich. Um B zu beschaffen, genügen eine Invertierung, 2 Multiplikationen und 2 Additionen. Die in Abbildung 6.3 angedeutete Vorgehensweise führt also zu folgendem Ergebnis:

Satz 6.21. *Sei (n, b, c) zulässige Eingabe für den QFT und b invertierbar modulo n. Dann gilt: Das Element x^t in $\mathbb{Z}_n[X]/(X^2 - bX - c)$ kann berechnet werden mittels 4 Invertierungen, $3\log m + \frac{2\log m}{\log\log m} + \frac{1}{2}(\log m)^{3/5} + O(\log\log m)$ Multiplikationen, sowie $2\log m + O(1)$ Additionen bzw. Subtraktionen modulo n, wobei $m = \lfloor t/2 \rfloor$*

Diese Vorgehensweise ist nur möglich, wenn $\Delta = b^2 + 4c$, c und b modulo n invertierbar sind. Für die ersten beiden Werte können wir das aber voraussetzen, weil es aus $\left(\frac{\Delta}{n}\right) = -1$ und $\left(\frac{-c}{n}\right) = 1$ folgt. Sollte b modulo n nicht invertierbar sein, dann kann das wegen (a) aus Algorithmus 9 nur daran liegen, dass $b \equiv 0 \bmod n$. Damit vereinfacht sich die Bestimmung von $X^t \bmod (n, X^2 - bX - c)$ aber enorm: Sei $t = 2m + j$, $j = 0$ oder $j = 1$. Dann ist

$$X^t = (X^2)^m X^j \equiv c^m X^j \bmod (n, X^2 - c),$$

also entweder $x^t = c^{\lfloor t/2 \rfloor}$ oder $x^t = c^{\lfloor t/2 \rfloor} x$. Dies kann nach Satz 4.11 mittels $\log t + \frac{2\log t}{\log\log t} + \frac{1}{2}(\log t)^{3/5} + O(\log\log t)$ Multiplikationen modulo n berechnet werden.

6.6.2. Effiziente Durchführung aller Schritte des QFT

Wir kehren von unserm Ausflug in die Details der Berechnung von x^t zurück zum eigentlichen Objekt, dem QFT und seiner Laufzeit. Im aktuellen Abschnitt wollen wir sehen, wie die Argumente aus [Gra98] die Laufzeitschranke von asymptotisch 3 Miller-Rabin-Tests für den QFT begründen.

Zunächst halten wir uns wieder die Idee vor Augen. Der QFT untersucht die Potenzen $x^{\frac{n+1}{2}}$ und x^{n+1} sowie x^s und einige aus $\{x^{2^j s}, j = 1, 2, 3, \ldots, r-2\}$. Für eine schnelle Implementierung berechnet man zunächst $x^{\frac{n+1}{2}}$, was mit $3\log n + o(\log n)$ Multiplikationen modulo n schon den Löwenanteil der Laufzeit ausmacht. Aus den dabei ermittelten Zwischenergebnissen kann dann mit einer jeweils konstanten Anzahl von Multiplikationen und Invertierungen modulo n eine der anderen benötigten Potenzen berechnet werden. Die Anzahl der außer $x^{\frac{n+1}{2}}$ benötigten Potenzen ist nur $O(\log r) = O(\log\log n)$, weil die Frage, ob $x^{2^j s} \neq -1$ für alle $j \in \{0, 1, \ldots, r-2\}$, mit binärer Suche in diesem Bereich erledigt werden kann.

Kommen wir nun zu den Details. Die Berechnung von x^{n+1} aus $x^{\frac{n+1}{2}}$ ist offensichtlich: Ein Quadrierung $\mathbb{Z}_n[X]/(X^2 - bX - c)$, nach Satz 4.7 verbunden mit 5 Multiplikationen modulo n, leistet das Gewünschte. Das eigentliche Problem sind also die Potenzen $x^{2^j s}, j = 0, 1, 2, \ldots, r-2$. Hierfür hängt die konkrete Vorgehensweise davon ab, ob $n \equiv 1 \bmod 4$ oder $n \equiv 3 \bmod 4$. Wir behandeln zunächst ausführlich den Fall $n \equiv 3 \bmod 4$, dessen Prinzipien auch in dem anderen Fall angewendet werden[8].

Fall 1 Sei also $n \equiv 3 \bmod 4$. Nach Definition ist s der ungerade Anteil von $n^2 - 1 = (n+1)(n-1)$. Wir zerlegen nun auch $n+1$ in eine Zweierpotenz und einen ungeraden Anteil: Es sei $n + 1 =: 2^{r'} s'$ mit $2 \not| s'$. Da $n \equiv 3 \bmod 4$, ist $n+1$ durch 4 teilbar, also $r' \geq 2$. Deshalb können wir s und r mittels r' und s' ausdrücken:

$$2^r s = (n+1)(n-1) = 2^{r'} s'(2^{r'} s' - 2) = 2^{r'+1}(2^{r'-1} s'^2 - s').$$

Da $r' - 1 \geq 1$ ist, ist $2^{r'-1} s'^2 - s'$ ungerade. Der Wert 2^{r+1} ist gerade, also sind diese beiden Ausdrücke nichts anderes als 2^r und s. Somit gilt

$$r = r' + 1 \quad \text{sowie} \quad s = 2^{r'-1} s'^2 - s'.$$

Die Idee ist nun, $x^{\frac{n+1}{2}}$ nicht direkt zu bestimmen, sondern über $x^{s'}$ bzw. Vorstufen davon, die dann für die Berechnung von x^s genutzt werden können. Dazu wird nun noch eine weitere Variable eingeführt: Es sei $t := \frac{s'-1}{2}$. Es gelten nun folgende Zusammenhänge, nach denen die benötigten Potenzen $x^{\frac{n+1}{2}}, x^{n+1}, x^s$ und $x^{2^{e+1} s}$ ermittelt werden können:

$$\frac{n+1}{2} = 2^{r'-1}(2t+1), \tag{6.23}$$

$$n + 1 = 2\frac{n+1}{2}, \tag{6.24}$$

$$s = nt + t + \frac{n+1}{2} - s', \tag{6.25}$$

$$2^{e+1} s = n 2^e s' - 2^e s'. \tag{6.26}$$

Beweis. (6.23) gilt nach Definition von r', s' und t. (6.24) ist trivial. (6.25) ergibt sich durch

$$s = 2^{r'-1} s'^2 - s' = \frac{n+1}{2} s' - s' = \frac{n+1}{2}(2t+1) - s'$$
$$= (n+1)t + \frac{n+1}{2} - s' = nt + t + \frac{n+1}{2} - s',$$

[8] In [Gra98] wird $n \equiv 1 \bmod 4$ ausführlich behandelt, $n \equiv 3 \bmod 4$ nur kurz.

(6.26) durch

$$\begin{aligned} 2^{e+1}s &= 2^{e+1}(2^{r'-1}s'^2 - s') \\ &= 2^e(n+1)s' - 2 \cdot 2^e s' = 2^e n s' - 2^e s'. \quad \square \end{aligned}$$

An dieser Stelle ist zu bemerken, dass (6.25) in [Gra98] falsch angegeben ist, dort fehlt der Term „$-s'$“.

Der QFT führt im Fall $n \equiv 3 \bmod 4$ die folgenden Schritte aus: Zunächst wird

$$(V_t, U_t, C_t) \tag{6.27}$$

bestimmt. Dies benötigt $3\log t + o(\log t)$ Multiplikationen modulo n (vgl. Herleitung von Satz 6.18). Durch eine $\circ$-Quadrierung, eine $\circ$-Multiplikation in M und anschließend $r'-1$ $\circ$-Quadrierungen wird daraus (6.23) folgend $(V_{\frac{n+1}{2}}, U_{\frac{n+1}{2}}, C_{\frac{n+1}{2}})$ ermittelt. Danach wird $x^{\frac{n+1}{2}}$ bestimmt. Nach Satz 6.17 erfordert dies nur $3{+}8{+}3(r{-}1){+}2$ Multiplikationen und die Kosten für die Bereitstellung von 2^{-1} modulo n, so dass bis dahin $3\log t + 3(r'-1) + o(\log t) = 3\log n + o(\log n)$ Multiplikationen modulo n durchgeführt werden. Anschließend wird x^{n+1} aus $x^{\frac{n+1}{2}}$ bestimmt mit 5 weiteren Multiplikationen modulo n.

Falls n nun noch nicht als zusammengesetzt identifiziert wurde, gilt $x^{n+1} = -c$. In Abschnitt 6.2 haben wir gesehen, dass dies äquivalent ist zu $x^n = b - x = \sigma(x)$, wobei $\sigma : f \mapsto f(b-x)$ die Abbildung aus Satz 3.58 ist. Also ist $x^{nt} = (x^n)^t = \sigma(x)^t = \sigma(x^t)$, wobei im letzten Schritt Lemma 3.17 mit $g = b - X$ angewendet wurde. Damit ermöglicht (6.25) die schnelle Berechnung von x^s wie folgt:

$$x^s = \sigma(x^t)x^t x^{\frac{n+1}{2}}(x^{s'})^{-1}.$$

$x^{s'} = x^{2t+1}$ lässt sich aus $(V_{2t+1}, U_{2t+1}, C_{2t+1})$ mit 2 Multiplikationen modulo n bestimmen. Daraus das Inverse in $\mathbb{Z}_n[X]/(X^2 - bX - c)$ zu bestimmen[9], kostet eine Invertierung und 7 Multiplikationen modulo n (Satz 4.8). Der Wert $\sigma(x^t)$ wird nach (4.5)

[9] Dass so etwas nötig ist, wird in [Gra98] nicht erwähnt. Dort wird nämlich ausführlich nur der Fall $n \equiv 1 \bmod 4$ diskutiert wird, wo keine Invertierungen in $\mathbb{Z}_n[X]/(X^2 - bX - c)$ erforderlich sind. Die knappen Bemerkungen zu Fall $n \equiv 3 \bmod 4$ in [Gra98] sprechen dafür, dass der Autor entweder nicht bemerkt hat, dass die negativen Summanden in (6.25) und (6.26) Invertierung in $\mathbb{Z}_n[X]/(X^2 - bX - c)$ erforderlich machen, oder dass er es nicht für wesentlich gehalten hat. Für die letztere Möglichkeit würde seine Definition des „selfridge“ als Zeitmaß sprechen, in der er für Invertierung modulo n die Zeit von nur $O(1)$ Multiplikationen modulo n veranschlagt. Dies ist problematisch, wie auf Seite 66 erläutert wurde.

mittels einer Multiplikation modulo n aus x^t bestimmt, was seinerseits mit 2 Multiplikationen aus (V_t, U_t, C_t) zu berechnen ist. Der Wert $x^{\frac{n+1}{2}}$ wurde schon berechnet, so dass außerdem nur noch die Kosten für 3 Multiplikationen in $\mathbb{Z}_n[X]/(X^2 - bX - c)$ anfallen, also 15 Multiplikationen modulo n. Insgesamt kann x^s also mit 27 Multiplikationen und einer Invertierung modulo n berechnet werden.

Analog können die Potenzen $x^{2^j s}$ für $j = 1, 2, \ldots, r-2$ nach (6.26) berechnet werden als

$$x^{2^{e+1}s} = \sigma(x^{2^e s'})(x^{2^e s'})^{-1}, \quad e = 0, 1, 2, \ldots, r-2 = r'-1.$$

Dazu sind jeweils 2 Multiplikationen nötigt, um aus $(V_{2^r s'}, U_{2^r s'}, C_{2^r s'})$ die Potenz $x^{2^e s'}$ zu bestimmen, 1 Multiplikation für die Berechnung von $\sigma(x^{2^e s'})$, 7 Multiplikationen und eine Invertierung modulo n für die Invertierung in $\mathbb{Z}_n[X]/(X^2 - bX - c)$ und 5 Multiplikationen modulo n für die Multiplikation in $\mathbb{Z}_n[X]/(X^2 - bX - c)$. Insgesamt sind das eine Invertierung und 15 Multiplikationen modulo n.

Wir beschließen den Fall $n \equiv 3 \bmod 4$ mit einer detaillierten Beschreibung der binären Suche nach -1 in $\{x^{2^j s} \mid j = 0, 1, \ldots, r-2\}$. Wir beginnen mit $j := \lfloor \frac{r}{2} \rfloor$ und bestimmen $x^{2^j s}$. Falls dies den Wert -1 hat, ist die Suche beendet. Falls $x^{2^j s} = 1$, ist $x^{2^e s} = 1$ für alle $e \geq j$, so dass wir den Suchbereich halbiert haben. Falls $x^{2^j s} \neq 1$ und $x^{2^j s} \neq -1$, ist $x^{2^e s} \neq -1$ für alle $e \leq j$, so dass sich auch in diesem Fall der Suchbereich halbiert. Nun setzen wir die Suche in dem verbleibenden Suchbereich auf die übliche Weise rekursiv fort. Nach höchstens $\lfloor \log r \rfloor + 1$ Schritten ist der Wert -1 gefunden oder bewiesen, dass dieser Wert nicht auftaucht. Da $r \leq 2 \log n$, müssen nur $\lfloor \log r \rfloor + 1 \leq \lfloor \log \log n \rfloor + 2$ Potenzen $x^{2^{e+1} s}$ ausgerechnet werden.

Wir geben als Satz an, welche Laufzeit die Implementierung des QFT nach [Gra98] im Fall $n \equiv 3 \bmod 4$ benötigt, wobei wir für die Zahl der Multiplikationen anstelle des Ausdrucks $3 \log n + o(\log n)$ die genauere Angabe von (6.19) verwenden:

Satz 6.22. *Sei $n \equiv 3 \bmod 4$. Es sei $2^{r'} s' = n + 1$ die Zerlegung von $n + 1$ in eine Zweierpotenz und den ungeraden Anteil s', sowie $t = (s' - 1)/2$. Dann kann der QFT durchgeführt werden mittels*

$$\begin{aligned} \lfloor \log \log n \rfloor + O(1) \quad & \textit{Invertierungen modulo } n, \\ 3 \log n + \frac{16 \log t}{\log \log t} + 4(\log t)^{3/5} + O(\log \log n) \quad & \textit{Multiplikationen modulo } n \textit{ und} \\ O(\log n) \quad & \textit{Additionen/Subtraktionen modulo } n. \end{aligned}$$

Fall 2 Im Fall $n \equiv 1 \bmod 4$ geht man so vor: Die durch 4 teilbare Zahl $n-1$ wird zerlegt in $n-1 = 2^{r'}s'$ mit $2 \not| s'$. Dann gilt $r' \geq 2$, sowie $2^r s = (n+1)(n-1) = (2^{r'}s'+2)2^{r'}s = 2^{r'+1}(2^{r'-1}s'^2 + s')$, also $r = r'+1$ und $s = 2^{r'-1}s'^2 + s'$. Wieder wird $t = \frac{s'-1}{2}$ gesetzt und (V_t, U_t, C_t) berechnet. Daraus wird $(V_{\frac{n-1}{2}}, U_{\frac{n-1}{2}}, C_{\frac{n-1}{2}})$ bestimmt, woraus man $x^{\frac{n-1}{2}}$ ermittelt. Dies wird mit x multipliziert und ergibt $x^{\frac{n+1}{2}}$. Quadrieren ergibt x^{n+1}. Falls es nötig ist, x^s und einige $x^{2^j s}$ zu berechnen, kann dies geschehen, indem folgende Gleichungen ähnlich wie im Fall $n \equiv 3 \bmod 4$ benutzt werden:

$$s = nt + t + \frac{n-1}{2} + 1 \tag{6.28}$$

$$2^{e+1}s = n2^e s' + 2^e s'. \tag{6.29}$$

Beweis. (6.28) ergibt sich durch

$$\begin{aligned} s &= 2^{r'-1}s'^2 + s' = \frac{n-1}{2}s' + s' \\ &= \frac{n-1}{2}(2t+1) + (2t+1) = (n-1)t + \frac{n-1}{2} + 2t + 1 \\ &= nt + \frac{n-1}{2} + t + 1, \end{aligned}$$

(6.29) durch

$$\begin{aligned} 2^{e+1}s &= 2^{e+1}(2^{r'-1}s'^2 + s') \\ &= 2^e(n-1)s' + 2 \cdot 2^e s' = 2^e n + 2^e s'. \end{aligned}$$ □

Wie an (6.28) und (6.29) zu sehen ist, kommt man in diesem Fall ohne Invertierungen in $\mathbb{Z}_n[X]/(X^2 - bX - c)$ aus. Es muss lediglich für Satz 6.17 der Wert $2^{-1} \bmod n$ bestimmt werden[10]. Die Implementierung des QFT nach [Gra98] benötigt also im Fall $n \equiv 1 \bmod 4$ folgende Laufzeit:

Satz 6.23. *Sei $n \equiv 1 \bmod 4$. Es sei $2^{r'}s' = n-1$ die Zerlegung von $n-1$ in eine Zweierpotenz und den ungeraden Anteil s', sowie $t = (s'-1)/2$. Dann kann der QFT durchgeführt werden mittels einer Invertierung modulo n, nämlich der Berechnung von $2^{-1} \bmod n$, sowie*

$$3\log n + \frac{16\log t}{\log\log t} + 4(\log t)^{3/5} + O(\log\log n) \quad \textit{Multiplikationen modulo } n \textit{ und}$$

$$O(\log n) \quad \textit{Additionen/Subtraktionen modulo } n.$$

[10] Auch wenn mehrere Iterationen des RQFT durchgeführt werden, muss nur bei der ersten $2^{-1} \bmod n$ berechnet werden.

Multiplikationen vs. Invertierungen Wie wir gesehen haben, nutzt die Implementierungen des QFT im Fall $n \equiv 3 \bmod 4$ bis zu $\lfloor \log\log n \rfloor$ und mehr Invertierungen in $\mathbb{Z}_n$. Deshalb ist zunächst zu kritisieren, dass dies in [Gra98] nicht erwähnt wird. Weiter stellt sich nun die Frage, welche der Operationen die Bitkomplexität bestimmen. Wenn wir die asymptotisch besten bekannten Verfahren zur Multiplikation bzw. Invertierung in $\mathbb{Z}_n$ zugrunde legen (siehe Satz 4.4 und 4.5), dann stellen wir fest, dass die Multiplikationen den wesentlichen Anteil ausmachen: Die $3\log n + o(\log n)$ Multiplikationen in $\mathbb{Z}_n$ benötigen $\Theta(\log n \cdot M(\log n))$ Bitschritte, während $O(\log\log n)$ Invertierungen nur $O(\log\log n \cdot M(\log n) \cdot \log\log n)$ Bitschritte benötigen.

Verbesserte Laufzeit bei den Potenzen

Die auf [CP01, Abschnitt 3.5.3] beruhende, schnellere Methode zur Bestimmung der Potenzen kann eingesetzt werden, indem in (6.27) nicht $\big(V_t(b,c), U_t(b,c), C_t\big)$, sondern

$$\big(V_{\lfloor t/2 \rfloor}(B,-1), V_{\lfloor t/2 \rfloor+1}(B,-1)\big) \quad \text{und} \quad c^{\lfloor t/2 \rfloor} \bmod n$$

berechnet werden. Von dort aus können dann alle benötigten Potenzen von x bestimmt werden, ganz analog zu dem bisher Dargestellten. Allerdings muss bei der Implementierung darauf geachtet werden, ob t ungerade oder gerade ist. Es ergibt sich:

Satz 6.24. *Sei n eine zulässige Eingabe für den QFT. Weiter sei*

$$2^{r'} s' := \begin{cases} n-1, & \textit{falls } n \equiv 1 \bmod 4, \\ n+1, & \textit{falls } n \equiv 3 \bmod 4 \end{cases}$$

sowie $t := (s'-1)/2$ und $m := \lfloor t/2 \rfloor$. Dann kann der QFT durchgeführt werden mittels

- $3\lfloor \log\log n \rfloor + O(1)$ *Invertierungen modulo n, falls $n \equiv 3 \bmod 4$, bzw. $O(1)$ Invertierungen modulo n, falls $n \equiv 1 \bmod n$, sowie*
- $2\log n + \log m + \frac{2\log m}{\log\log m} + \frac{1}{2}(\log m)^{3/5} + O(\log\log n)$ *Multiplikationen modulo n sowie*
- $O(\log n)$ *Additionen/Subtraktionen modulo n.*

6.6.3. Gesamtaufwand zum Erkennen von Primzahlen

Nachdem wir nun die Laufzeit des QFT kennen, wollen wir den Gesamtaufwand diskutieren, der nötig ist, um mit Hilfe des RQFT Primzahlen von zusammengesetzten Zahlen zu unterscheiden.

Wie wir inzwischen wissen, benötigt der QFT $3 \log n \cdot M(\log n) + o(\log n \cdot M(\log n))$ Bitoperationen. Bevor der RQFT den QFT aufruft, müssen maximal B mal — bzw. nach Satz 6.4 durchschnittlich höchstens 4 mal — die drei größten gemeinsamen Teiler $\mathrm{ggT}(b, n)$, $\mathrm{ggT}(c, n)$ und $\mathrm{ggT}(b^2 + 4c, n)$ sowie die Jacobi-Symbole $\left(\frac{b^2+4c}{n}\right)$ und $\left(\frac{-c}{n}\right)$ berechnet werden. Beides ist möglich mit $O((\log n)^2)$ Bitoperationen[11]. Da die besten bekannten Verfahren zur Multiplikation ganzer Zahlen $M(l) = O(l \log l \log\log l)$ Bitoperationen benötigen, überwiegt der Aufwand für den QFT gegenüber dem Aufwand zum Finden von (b, c). In der Praxis ist überdies $M(l) = \Omega(l^{1.465})$ (Verfahren nach Toom-Cook, vgl. Abschnitt 7.1.1), so dass der Unterschied noch etwas deutlicher ausfällt.

Bevor n dem RQFT überhaupt als Eingabe übergeben werden kann, muss geprüft werden, ob n eine Quadratzahl ist. Dies erfordert nur $O(M(\log n))$ Bitschritte (siehe Abschnitt 4.4.2), fällt also nicht ins Gewicht. Außerdem müssen eventuelle Primteiler von n, die kleiner als B sind, gefunden werden. Dies erfordert bis zu $\pi(B)$ Divisionen mit Rest, in denen die Zahl n durch eine kleine Zahl geteilt wird, und den Aufwand zum Bestimmen aller Primzahlen $\leq B$ mittels eines Siebverfahrens. Da B konstant ist, kann aus theoretischer Sicht das Sieben mit konstantem Aufwand verbucht und das Dividieren als konstante Anzahl von Divisionen mit Rest betrachtet werden. Dies fällt gegenüber dem QFT nicht ins Gewicht fällt. Aus praktischer Sicht wäre es jedoch durchaus interessant zu untersuchen, ob die 5133 Divisionen durch alle Primzahlen ≤ 50000 gegenüber den ca. 3000 modularen Multiplikationen vernachlässigt werden können, die der QFT bei Eingabe von Zahlen der Länge beispielsweise 1024 Bit mindestens benötigt.

[11] [vzGG03] ist zu entnehmen, dass der erweiterte Euklidische Algorithmus sogar in $O(M(\log n) \log\log n)$ Schritten durchgeführt werden kann.

7. Miller-Rabin-Test und Frobenius-Test im experimentellen Vergleich

Nun sollen die Experimente präsentiert werden, in denen die Laufzeit des Miller-Rabin-Tests und des quadratischen Frobenius-Tests untersucht wurden. Diese waren nötig, um zu prüfen, ob die theoretischen Aussagen über die Laufzeit des Frobenius-Tests sich in der Praxis wiederfinden.

Auch wenn die Experimente nun für sich am Ende der Arbeit dargestellt werden, sind sie doch nicht so klar, wie es die Kapiteleinteilung vermuten lassen könnte, von den theoretischen Erwägungen zu trennen. Nur weil sich in den Experimenten herausstellte, dass der RQFT nach [Gra98] in der Praxis zeitlich nicht drei, sondern vier Miller-Rabin-Tests entspricht, während sich die Implementierung des einfachen quadratischen Frobenius-Tests nach [CP01, Abschnitt 3.5.3] tatsächlich zeitlich bei drei Miller-Rabin-Tests bewegt, erschien es lohnenswert, die Implementierung nach [CP01, Abschnitt 3.5.3] für den RQFT nutzbar zu machen.

7.1. Aufbau der Experimente

Alle den Tests vorgelegten Zahlen waren Primzahlen, genauer gesagt Zahlen, die von 10 Iterationen des Miller-Rabin-Tests als Primzahlen klassifiziert wurden. Wie zu erwarten, wurde später bei den Experimenten auch keine dieser Zahlen als zusammengesetzt erkannt. Man kann also davon ausgehen, dass in den Experimenten die Tests ausschließlich mit Primzahlen konfrontiert wurden. Dies war beabsichtigt, damit stets die Zeit für die Abarbeitung *aller* Teilschritte des quadratischen Frobenius-Tests gemessen wird.

Die Länge der untersuchten Zahlen beträgt 128 bis 4096 Bit, wobei mit einer Schrittweite von 64 Bit vorgegangen wurde. Für jede der untersuchten Bitlängen wurden fünfzehn „Primzahlen“ — im gerade beschriebenen Sinne — erstellt, die als Eingaben für die verschiedenen Primzahltests dienten.

Alle Experimente bestanden darin, mehrere Iterationen eines vorgegebenen Primzahltests auf eine der bereitgestellten Eingaben anzuwenden. Die dafür benötigte Zeit wurde gemessen und durch die Anzahl der Iterationen geteilt. So ergab sich als Messergebnis die für eine Iteration des jeweiligen Tests benötigte Zeit in Sekunden.

Die Experimente wurden auf jede der generierten Eingaben angewendet, teilweise mehrere Male. Für die Auswertung wurde für jede der betrachteten Bitlängen der Median aus allen Messergebnissen gebildet und graphisch dargestellt (siehe Abbildungen 7.1 bis 7.4 in Abschnitt 7.2).

Implementiert wurden die Primzahltests in ANSI C++, compiliert mit dem C++-Compiler[1] der GNU Compiler Collection [GCC], wie er von SuSE-Linux 9.2 installiert wird. Unter SuSE-Linux 9.2 wurden auch alle Experimente durchgeführt. Die Hardwareumgebung für die Untersuchungen bildete ein PC, der über einen Pentium-III-Prozessor mit 800 MHz und 256KByte Cache sowie 128 MByte Hauptspeicher verfügt. Auf einem Notebook mit Pentium-4-Prozessor (1.8 GHz) waren die gemessenen Zeiten nicht stabil, vermutlich weil der Rechner während der Experimente selbständig die Taktgeschwindigkeit änderte, um Überhitzung vorzubeugen. Deshalb wird darauf verzichtet, die dort erzielten Ergebnisse anzugeben.

7.1.1. Implementierung der arithmetischen Operationen — die GMP-Bibliothek

Die in den Primzahltests auftretenden Zahlen benötigen sehr viel mehr Platz, als von den in C++ eingebauten Datentypen für ganze Zahlen (**int**, **long int**) zur Verfügung gestellt wird. Deshalb muss auf eine geeignete Datenstruktur für große ganze Zahlen zurückgegriffen werden, sowie auf Methoden, die in dieser Datenstruktur repräsentierte Zahlen manipulieren. Dafür wurde eine vorhandene freie Bibliothek eingesetzt, nämlich GMP[2], die GNU Multiple Precision Arithmetic Library [GMP].

GMP repräsentiert lange Zahlen durch ein Feld von sogenannten „limbs“. Das sind

[1] Version 3.3.4, pre 3.3.5 20040809
[2] Version 4.1.3

Teile einer großen ganzen Zahl, die auf dem jeweiligen Rechner in ein Maschinen-Wort passen, zum Beispiel 32 Bit. Für eine 2048-Bit-Zahl würden also etwa 64 Limbs benötigt. Addition und Subtraktion von ganzen Zahlen wird nach der Schulmethode durchgeführt. Zur Multiplikation großer ganzer Zahlen werden abhängig von der Bitlänge der Operanden verschiedene Verfahren genutzt. Für die von uns betrachteten Zahlengrößen sind die Schulmethode sowie die Verfahren nach Karatsuba bzw. Toom-Cook relevant. Bei letzteren handelt es sich um Divide-and-Conquer Verfahren, die die Multiplikation zweier langer Zahlen geschickt zurückführen auf mehrere Multiplikationen von Teilen dieser Zahlen. Für die Details sei auf die die GMP-Dokumentation und auf [Knu98, Abschnitt 4.3.3] verwiesen. Die Schulmethode benötigt für die Multiplikation zweier n-Bit-Zahlen $O(n^2)$ Multiplikationen einzelner Limbs, das Verfahren nach Karatsuba asymptotisch $O(n^{\log_2 3}) = O(n^{1.585})$ Limb-Multiplikationen und das Verfahren nach Toom-Cook $O(n^{\log_3 5}) = O(n^{1.465})$. Da die asymptotisch schnelleren Verfahren aufwendige Nebenrechnungen erfordern, sind sie erst ab einer bestimmten Bitlänge auch in der Praxis schneller als die einfacheren Verfahren. Deshalb werden in der GMP-Bibliothek Grenzen für die Bitlänge der zu multiplizierenden Zahlen definiert. Ja nachdem, ob die Bitlänge sich oberhalb der Grenze für Toom-Cook, unterhalb davon, oder sogar unterhalb der Grenze für das Karatsuba-Verfahren befindet, wird das Verfahren nach Toom-Cook, Karatsuba oder die Schulmethode eingesetzt. Die rekursiven Verfahren (Toom-Cook und Karatsuba) rufen das jeweils eine Stufe einfachere Verfahren (Karatsuba bzw. die Schulmethode) auf, sobald die Länge der durch den Divide-Schritt entstandenen Zahlen die entsprechende Grenze unterschreitet. Die konkreten Werte der Schwellen[3] wurden abhängig vom eingesetzten Prozessor von den Autoren der GMP-Bibliothek experimentell optimiert.

Auch für die Division mit Rest wird je nach Bitlänge die Schulmethode oder ein Divide-and-Conquer-Verfahren eingesetzt. Asymptotisch reichen mit dem Divide-and-Conquer-Verfahren $O(M(n) \log n)$ Limb-Operationen, wobei n die Bitlänge der beteiligen Zahlen und $M(n)$ die Zahl der Limb-Operationen für eine Multiplikation von n-Bit-Zahlen ist. In dem Bereich, in dem für die Multiplikation Toom-Cook eingesetzt wird, gibt die GMP-Dokumentation $2.63M(n)$ als realistischen Wert für den Aufwand einer n-Bit-Division an. Für weitere Details sei auf die GMP-Dokumentation und die dort zitierte Literatur verwiesen.

[3] Für Pentium-4 verwendet GMP 4.1.4 die Schulmethode bei Zahlen mit bis zu 576 Bit, bis 4448 Bit Karatsuba, bis ca. 180000 Bit Toom-Cook, danach schnelle Fourier-Transformation.

Die Berechnung von Inversen modulo einer ganzen Zahl erfolgt mit dem erweiterten Euklidischen Algorithmus. In GMP wird dieser verwendet, wobei eine Idee von Lehmer für die Vereinfachung des Aufwands bei großen Zahlen eingesetzt wird. Das Ergebnis ist ein Verfahren, dass $O(n^2)$ Limb-Operationen benötigt, wobei n wieder die Bitlänge der beteiligten Zahlen ist.

Die Berechnung von $a^b \bmod k$ führt GMP nach der 2^m-ären Methode durch, wie sie in Abschnitt 4.3 beschrieben wurde. Der Parameter m wird dabei abhängig von der Größe des Exponenten b gewählt.

7.1.2. Implementierungsvarianten

Der Frobenius-Test wurde in drei verschiedenen Varianten implementiert. In der einfachsten werden die Potenzen x^t durch Algorithmus 3 berechnet, wobei modulo $(n, X^2 - bX - c)$ multipliziert und quadriert wird. Jede dieser beiden Operationen benötigt also 5 Multiplikationen modulo n, so dass in dieser Variante der Frobenius-Test die Laufzeit von 5 Miller-Rabin-Tests benötigt (vgl. die Bemerkungen am Beginn von Abschnitt 6.6). Dies ist die langsamste, aber auch die einfachste Implementierungsvariante. In den Diagrammen mit den Ergebnissen wird sie mit „`exp`“ abgekürzt.

Außerdem wurde der quadratische Frobenius-Test in der Weise implementiert, wie es in [Gra98] vorgeschlagen ist. Diese Variante erhält die Abkürzung „`gr`“.

Die implementierungstechnisch komplizierteste Variante ist die mit „`pn`“ abgekürzte. Hier wird — wie ab Seite 112 erläutert — die Technik aus [CP01] genutzt, um die Potenzen x^t zu bestimmen.

Sowohl der einfache quadratische Frobenius-Test (Implementierungsvarianten mit „`frob`“ im Namen) als auch der QFT (Implementierungsvarianten mit „`sfrob`“ im Namen) wurden implementiert. Naturgemäß ist der einfache quadratische Frobenius-Test leichter zu implementieren, weil dort die Berechnung der Potenz x^n bzw. x^{n+1} nicht so organisiert werden muss, dass bestimmte Zwischenergebnisse aufbewahrt und später zur schnellen Berechnung von $x^s, x^{2s}, x^{4s}, \ldots$ genutzt werden können. Wie die Analyse der Laufzeit und die Messergebnisse zeigen, ist der zeitliche Unterschied zwischen dem QFT und dem einfachen quadratischen Frobenius-Test nicht relevant.

Bei den Ergebnissen der Experimente tauchen auch für den Miller-Rabin-Test zwei Implementierungsvarianten auf, `mr01` und `mr02`. Sie unterscheiden sich nur darin, dass in der einen Variante die genaue Stelle, an der der Miller-Rabin-Test das Ergebnis fest-

stellt, festgehalten und aufbewahrt wird, so dass sie mitprotokolliert und ausgewertet werden könnte. Auf die gemessene Laufzeit hat das aber keine Auswirkungen.

7.1.3. Details der Implementierung

Der Miller-Rabin-Test wurde Algorithmus 6 folgend implementiert. Die Anweisungen dort lassen sich direkt in Aufrufe von GMP-Routinen übertragen.

In [CP01, Algorithmus 3.5.9] ist der einfache quadratische Frobenius-Test so weit ausgearbeitet, dass die einzelnen Anweisungen ebenfalls direkt in GMP-Aufrufe umgesetzt werden können. Dies ist in [Lav] geschehen, der Quellcode von dort wurde übernommen für die Implementierungsvariante `frob_pn01`. Die Variante `frob_pn02` unterscheidet sich von `frob_pn01` nur darin, dass anstelle des C-Typs `mpz_t` die C++-Klasse `mpz_class`, die diesen Typ kapselt, verwendet wurde. So konnte festgestellt werden, ob sich dieser Unterschied in der Geschwindigkeit bemerkbar macht.

Die Implementierung des QFT ist softwaretechnisch aufwendiger. Dort wurde für den Ring $R := \mathbb{Z}_n[X]/(X^2 - bX - c)$ ein eigenes Modul geschaffen. Diese Modul muss initialisiert werden, indem einer entsprechenden Funktion geeignete Werte für n, b und c übergeben werden. Nachdem das Modul initialisiert wurde, kann ein als C++-Klasse realisierter Datentyp genutzt werden, der Elemente dieses Rings repräsentiert. Die Ringelemente können durch entsprechende Methoden addiert, subtrahiert, multipliziert und quadriert werden. Für die σ-Operation und Ein-/Ausgabe sind ebenfalls C++-Methoden vorhanden. Der Grundgedanke war, den Code des Primzahltests übersichtlicher zu machen, indem ein effizienter benutzerdefinierter Datentyp[4] für Elemente von $\mathbb{Z}_n[X]/(X^2 - bX - c)$ geschaffen wurde.

Für die Implementierungsvariante `sfrob_pn01` wurde ein weiteres Modul geschaffen, das auf dem R realisierenden Modul aufbaut. Es enthält Funktionen, die die Berechnungsschritte durchführen, die durch die Gleichungen (6.21), (6.22), (3.25), (3.26), (3.24) und (3.18) beschriebenen sind. Eine weitere Funktion dieses Moduls berechnet ein beliebiges Paar $\big(V_m(B,-1), V_{m+1}(B,-1)\big)$ ausgehend von B (vgl. Satz 6.19). Schließlich gibt es Funktionen, die mehrere dieser Schritte kombinieren, indem sie ausgehend von $\big(V_m(B,-1), V_{m+1}(B,-1)\big)$ die Potenz x^{2m} bzw. x^{2m+1} ausrechnen oder für gegebenes j das Paar $\big(V_{\lfloor j/2\rfloor}(B,-1), V_{\lfloor j/2\rfloor+1}(B,-1)\big)$ sowie $c^{\lfloor j/2\rfloor}$ ausrechnen, aus denen dann später x^j bestimmt werden kann. Zur Repräsentation solcher Paare gibt es

[4] Mehr zu effizienten benutzerdefinierten Datentypen in C++ steht in [Str00], Abschnitt 10.3.

weiterhin einen Datentyp, die einfache Klasse **struct** `chainel`.

Auch bei `sfrob_gr01`, der Implementierung des RQFT nach [Gra98], gibt es ein zusätzliches Modul. Dies enthält eine Klasse, die einen einfachen benutzerdefinierten Datentyp für die Elemente $(V_j, U_j, C_j) \in M$ bereitstellt. Dort ist auch das Potenzieren von Elementen aus M[5] nach Satz 4.11[6] implementiert.

Wie den Ausführungen in Abschnitt 6.6.2 zu entnehmen ist, müssen die Elemente $(V_s, U_s, C_s), (V_{2s}, U_{2s}, C_{2s}), (V_{4s}, U_{4s}, C_{4s}), \ldots, (V_{2^{r'-1}s}, U_{2^{r'-1}s}, C_{2^{r'-1}s})$ bzw. die entsprechenden Elemente von N aufbewahrt werden. Dazu werden diese in einem `vector`[7] abgelegt. Das Element $(V_t, U_t, C_t) \in M$ bzw. $(V_{\lfloor t/2 \rfloor}, V_{\lfloor t/2 \rfloor+1}) \in N$ wird natürlich ebenfalls nach der Berechnung aufbewahrt.

Ebenfalls Abschnitt 6.6.2 ist zu entnehmen, dass die Implementierung von Algorithmus 8 im Detail davon abhängt, ob die zu untersuchende Zahl n bei Division durch 4 den Rest 1 oder 3 lässt. Folglich wurden beide Fälle mit einer eigenen Funktion realisiert, die zwar beide die GMP-Bibliothek sowie die Hilfsfunktionen aus den bereits beschriebenen Modulen und die dort bereitgestellten einfachen benutzerdefinierten Datentypen verwenden, ansonsten aber keinen Code gemeinsam benutzen.

Quadratfreie Zahlen werden mit der entsprechenden Routine der GMP-Bibliothek ausgeschlossen. Diese erledigt auch die Berechnung des Jacobi-Symbols, was in Algorithmus 9 nötig ist, um ein geeignetes Paar (b, c) zufällig zu wählen. Die Bedingung 1(c) aus Algorithmus 9 wird nicht überprüft, sondern es werden solange (b, c) gewählt, bis geeignete gefunden sind. Nach Satz 6.4 ist zu erwarten, dass das spätestens der Fall ist, nachdem vier mal zufällig (b, c) gewählt wurde. Im Übrigen folgt die Implementierung des RQFT genau Algorithmus 9.

[5] definiert auf Seite 111

[6] Allerdings mit einer kleinen Abweichung: In der Vorberechnung wird für *alle* Exponenten $b \in \{2, 3, 4, 5, \ldots, 2^m - 1\}$ der Wert a^b ausgerechnet — nicht nur für ungerade Exponenten. Anschließend wird nach (4.9) — und nicht nach (4.10) — die gewünschte Potenz ausgerechnet. Dadurch ist die Anzahl der sonstigen Multiplikationen geringfügig höher als in Satz 4.11 angegeben, es entfällt der Faktor $\frac{1}{2}$ vor dem Term $(\log k)^{3/5}$. Es ist aber nicht zu erwarten, dass die Messergebnisse sich spürbar verändern würden, wenn die Implementierung exakt dem Beweis von Satz 4.11 folgen würde.

[7] Siehe [Str00, Abschnitt 3.7.1] und [Str00, Kapitel 16 und 17].

7.2. Ergebnisse der Experimente

Die Ergebnisse der Experimente sind in Abbildung 7.1 bis 7.4 zu sehen. Die Abszisse ist stets mit der Bitlänge der untersuchten Primzahlen bezeichnet, während auf der Ordinate die Zeit in Sekunden abgetragen ist, die eine Iteration des jeweiligen Tests benötigt. Wie man an der Beschriftung erkennen kann, ist die für einen Miller-Rabin-Test benötigte Zeit schon verdrei- bzw. vervierfacht, damit man sie mit der Laufzeit der jeweiligen Variante des quadratischen Frobenius-Tests gut vergleichen kann.

Die Abbildungen 7.1 und 7.2 zeigen die Laufzeiten der einzelnen Tests auf Primzahlen $p \equiv 1 \bmod 4$, einerseits für Zahlen der Länge 128 bis 2048 Bit, andererseits für Zahlen der Länge 2048 bis 4096 Bit. In Abbildung 7.3 sind dieselben Laufzeiten noch einmal in einem einzigen Diagramm dargestellt. Abbildung 7.4 zeigt die Ergebnisse für Primzahlen $p \equiv 3 \bmod 4$.

Wir kommen zur Interpretation der Messergebnisse. Versieht man die Messergebnisse der beiden Miller-Rabin-Varianten mit dem gleichen Faktor, so kommt man zu dem Ergebnis, dass die Kurven als deckungsgleich angesehen werden können.

Wie man sieht, benötigt die naive Implementierung des Frobenius-Test, `frob_exp01`, etwas mehr als die Zeit von 5 Miller-Rabin-Tests und entspricht somit den Erwartungen. Ob man den starken quadratischen Frobenius-Test (`sfrob_gr01` bzw. `sfrob_pn01`) oder den vereinfachten quadratischen Frobenius-Test (`frob_gr01` bzw. `frob_pn01/02`) durchführt, hat auf die Laufzeit kaum messbare Auswirkungen. Auch das entspricht den Erwartungen, denn für den starken Teil werden nur $O(\log \log n)$ zusätzliche Multiplikationen und Invertierungen benötigt, während der übrige Teil des Tests $3 \log n + O(\frac{\log n}{\log \log n})$ Multiplikationen benötigt.

Der Unterschied zwischen der Implementierung nach [Gra98] (`xxx_gr0x`) und der aus [CP01, Abschnitt 3.5.3] entwickelten Methode (`xxx_pn0x`) beträgt etwa die Zeit eines Miller-Rabin-Test, was im Wesentlichen dem entspricht, was die Ausführungen auf Seite 112 erwarten lassen. Die leichte Überlegenheit von `sfrob_pn01` gegenüber dreimaligem Miller-Rabin-Test könnte man mit dem Unterschied zwischen $\frac{2 \log n}{\log \log n}$ und $\frac{6 \log n}{\log \log n}$ erklären: Wenn $\log \log n \approx 10$ ist, beträgt der Unterschied etwa $0.133 \cdot 3 \log n$. Allerdings beträgt der in den Diagrammen zu erkennende Vorsprung nicht 13%, sondern eher 5%. Das könnte durch die Invertierungen modulo n verursacht sein. $3 \log n$ Multiplikationen modulo n ergeben für eine 1024-Bit-Zahl n ungefähr 3000 Multiplikationen. Modulo n invertiert werden muss im schlimmsten Fall (`xxx_pn0x`-Verfahren,

$p \equiv 3 \bmod 4$)

$$\underbrace{4}_{\text{Berechnung von } (V_{\lfloor t/2 \rfloor}, V_{\lfloor t/2 \rfloor + 1})} + \underbrace{1}_{\text{für } x^s} + \underbrace{\lfloor \log \log n \rfloor + 2}_{\text{während der binären Suche}}$$

mal. Das sind etwa 17 Invertierungen. Da eine Invertierung modulo n bis zu 10 Multiplikationen modulo n entspricht (Abbildung 4.2), können die Invertierungen die gleiche Zeit wie 170 Multiplikationen benötigen. Das sind etwa 6% der 3000 Multiplikationen modulo n.

Prinzipiell sollten die Invertierungen die Laufzeit des Frobenius-Test nicht beeinflussen: Es werden mehr als $3 \log n$ Multiplikationen modulo n durchgeführt, aber nur $O(\log \log n)$ Invertierungen modulo n. Eine Multiplikation nach Toom-Cook benötigt etwa $O((\log n)^{1.465})$ Bitoperationen. Für die Multiplikationen ergibt sich also eine Bitkomplexität von $O((\log n)^{2.465})$. Eine Invertierung modulo n hat praktisch Bitkomplexität $O((\log n)^2)$, so dass die Bitkomplexität für alle Invertierungen nur $O(\log \log n \cdot (\log n)^2)$ ist. Auch wenn wir nicht die praktisch relevanten, sondern die asymptotisch schnellsten bekannten Verfahren zugrundelegen, überwiegt im Frobenius-Test der Aufwand für die Multiplikationen, im Vergleich zum Aufwand für die Invertierungen, wie wir auf Seite 121 gesehen haben.

Die Laufzeit von `sfrob_gr01` und vier Miller-Rabin-Iterationen kann man so vergleichen: etwa $3 \log n + \frac{16 \log n}{\log \log n} \approx 3000 + \frac{16}{10} \cdot 1000 = 4600$ Multiplikationen modulo n (invertiert werden muss nur $2 \bmod n$, wenn $n \equiv 1 \bmod 4$ ist, der Aufwand dafür kann ignoriert werden) stehen ungefähr $4 \log n + \frac{8 \log n}{\log \log n} \approx 4000 + \frac{8}{10} \cdot 1000 = 4800$ Multiplikationen modulo n gegenüber. Es ist $4800/4600 \approx 1.043$. Möglicherweise entspricht der der in den Diagrammen zu erkennende Unterschied dieser Abweichung von etwas 4%.

Bei dem mit `eqft01` bezeichneten Verfahren in Abbildung 7.4 handelt es sich um den Algorithmus EQFTac aus [DF03] in einer naiven Implementierung.

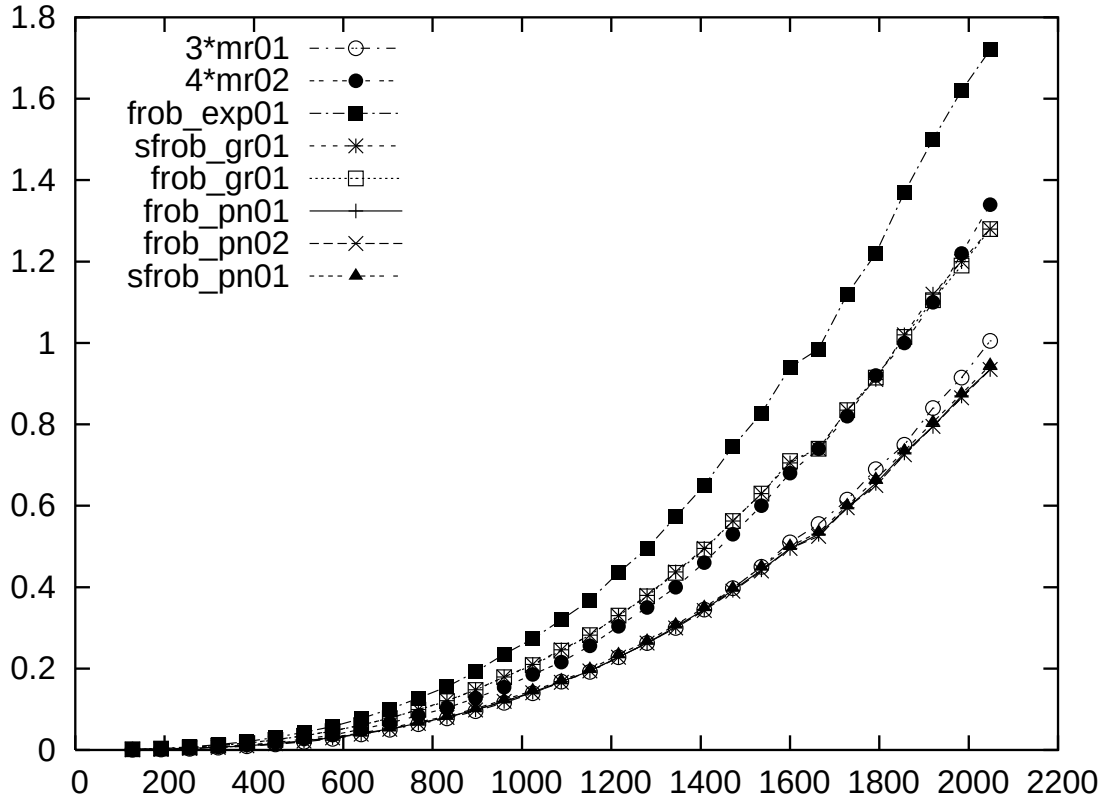

Abbildung 7.1.: Laufzeit des Miller-Rabin-Tests im Vergleich mit verschiedenen Varianten des quadratischen Frobenius-Test. Untersuchte Zahlen: Primzahlen $p \equiv 1 \bmod 4$.

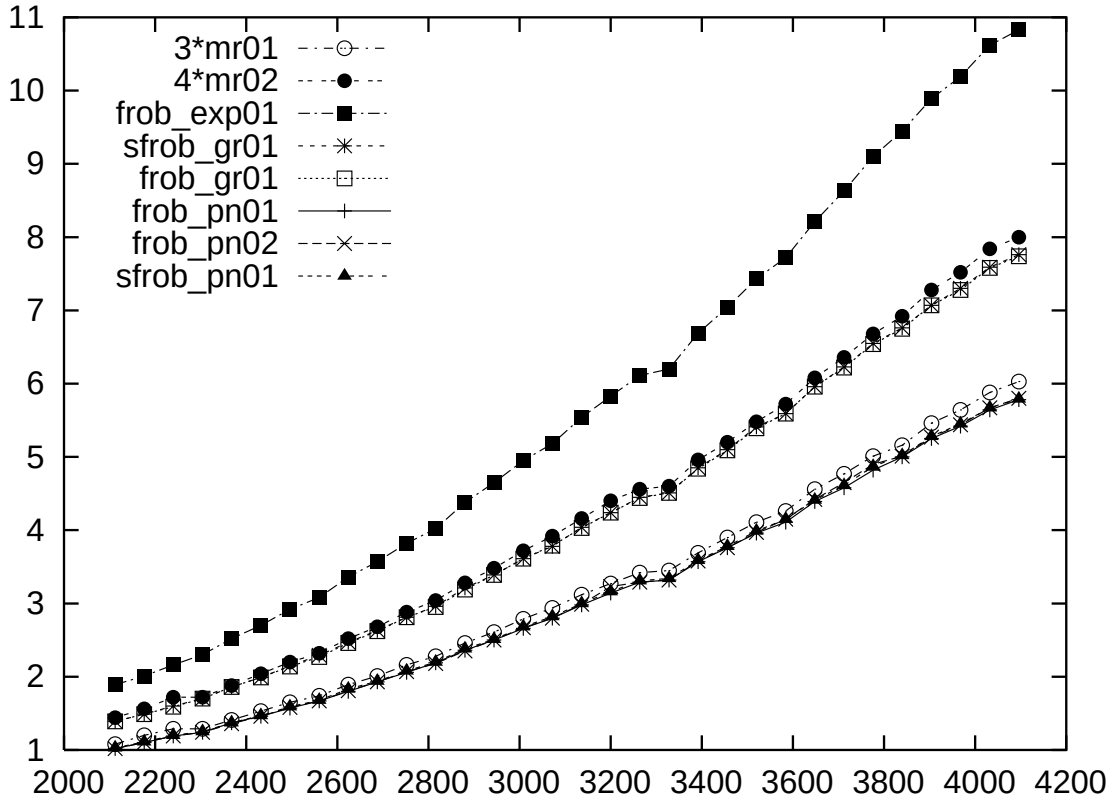

Abbildung 7.2.: Laufzeit des Miller-Rabin-Tests im Vergleich mit verschiedenen Varianten des quadratischen Frobenius-Test. Untersuchte Zahlen: große Primzahlen $p \equiv 1 \bmod 4$.

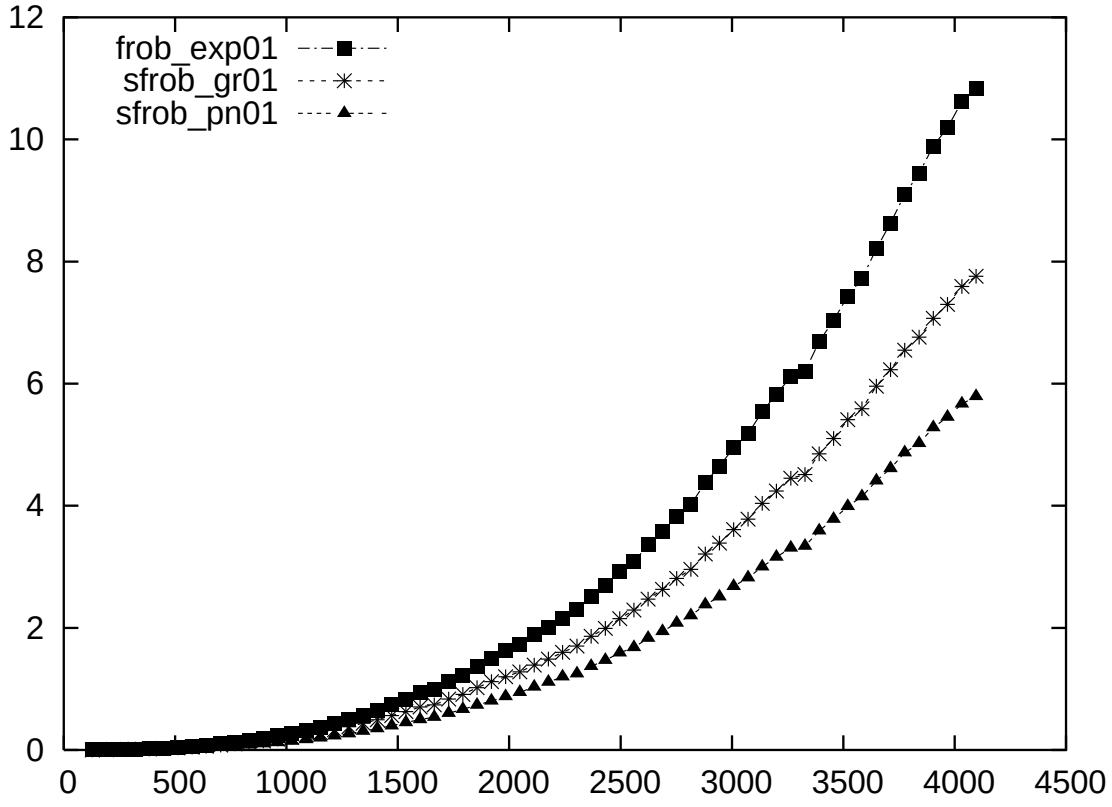

Abbildung 7.3.: Laufzeit des Miller-Rabin-Tests im Vergleich mit verschiedenen Varianten des quadratischen Frobenius-Test. Untersuchte Zahlen: Primzahlen $p \equiv 1 \bmod 4$.

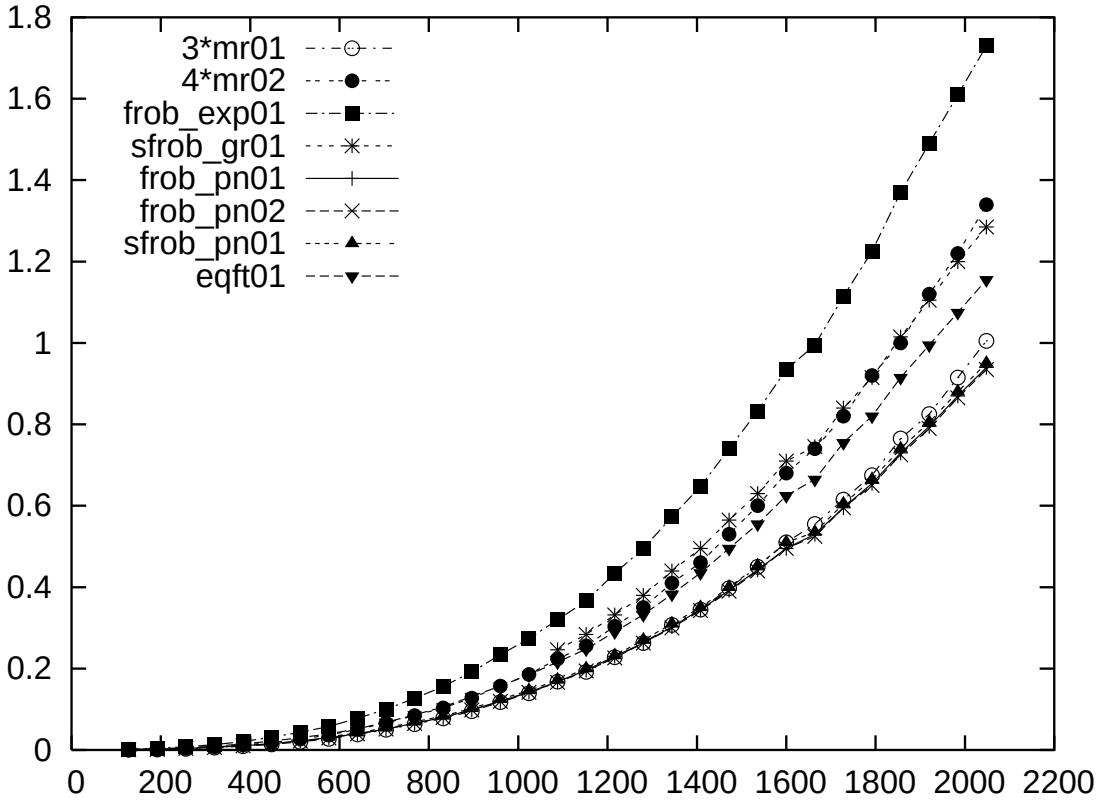

Abbildung 7.4.: Laufzeit des Miller-Rabin-Tests im Vergleich mit verschiedenen Varianten des quadratischen Frobenius-Test. Untersuchte Zahlen: Primzahlen $p \equiv 3 \bmod 4$.

8. Bewertung und Ausblick

Wir sind nun am Ende der Untersuchungen des quadratischen Frobenius-Tests angelangt. So ist es an der Zeit, Ergebnisse zusammenzufassen und ihre Bedeutung in einem größeren Zusammenhang zu erörtern.

Das wichtigste Ergebnis ist, dass der RQFT tatsächlich so implementiert werden kann, dass er die Zeit von nur drei Miller-Rabin-Test-Iterationen benötigt. Nach der Rechnung aus Abschnitt 2.2.2 ist es also gerechtfertigt zu sagen, dass der RQFT doppelt so schnell ist wie der Miller-Rabin-Test. Nun stellt sich die Frage nach den praktischen Auswirkungen.

Für die Primzahlerzeugung dürften diese kaum spürbar sein. Denn der größte Teil der Zeit wird hier damit zugebracht, eine große Zahl zufällig zu wählen, festzustellen, dass diese zusammengesetzt ist, und mit der nächsten Zahl weiterzumachen. In fast allen Fällen wird die Zahl als zusammengesetzt erkannt, weil sie einen kleinen Teiler hat oder weil sie einen einzelnen Fermat-Test nicht besteht. Die erste Zahl, die auf diese Weise nicht als zusammengesetzt erkannt wird, wird dann meistens auch von vielen Iterationen des Miller-Rabin- oder Frobenius-Tests nicht mehr als zusammengesetzt eingestuft, was Satz 5.1 genau so auch erwarten lässt. Damit fällt die Geschwindigkeitssteigerung, die man durch Verwendung des RQFT anstelle des Miller-Rabin-Tests erzielt, insgesamt so gering aus, dass sie in den Messfehlern untergeht. — Leider ist die Implementierung des RQFT um einiges komplizierter als die des Miller-Rabin-Tests, so dass es eigentlich keinen Grund gibt, bei der Primzahlerzeugung vom Miller-Rabin-Test auf den RQFT umzusteigen.

Wie sieht es mit der anderen Anwendung, dem Schutz vor Betrug in kryptographischen Protokollen aus? In diesem Szenario ist durchaus damit zu rechnen, dass der Einsatz des RQFT Vorteile bringt. Denn es sind hier nicht viele „durchschnittliche“ Zahlen zu untersuchen, sondern wenige, möglicherweise „speziell konstruierte“ Zahlen

müssen geprüft werden. Dadurch wirkt sich zunächst die Laufzeit des verwendeten Primzahltests viel stärker aus als bei der Primzahlerzeugung. Außerdem bekommt die Frage große Bedeutung, welche Struktur die Pseudoprimzahlen — also Zahlen, die den jeweiligen Test bezüglich bestimmter Parameter bestehen — aufweisen. Es ist nicht unrealistisch anzunehmen, dass zusammengesetzte Zahlen zur Untersuchung vorgelegt werden, die keine kleinen Teiler haben und auch bezüglich mehrerer Basen den Fermat-Test und vielleicht sogar den Miller-Rabin-Test bestehen[1]. Man kann nun hoffen[2], dass solche „betrügerischen“ Zahlen mit Hilfe des quadratischen Frobenius-Tests leichter als zusammengesetzt entlarvt werden, als es mit dem Miller-Rabin-Test der Fall wäre. Dann wäre folgende Vorgehensweise sinnvoll: Zunächst sollte man kleine Teiler ausschließen. Anschließend könnten einige Fermat- bzw. Miller-Rabin-Tests sowie Frobenius-Tests folgen. Wenn die Zahl dann immer noch nicht als zusammengesetzt identifiziert ist, werden so viele Iterationen des schnellsten Tests — also des RQFT — ausgeführt, wie nötig sind, um die Irrtumswahrscheinlichkeit auf den gewünschten Wert zu senken.

Falls sich der quadratische Frobenius-Test auf diese Weise als nützlich erweisen wird, könnten wir mit ihm ein weiteres Werkzeug in die Sammlung der Verfahren und Ideen aufnehmen, die Sicherheit auch für digitale Kommunikation möglich machen. Zu wünschen bleibt, dass von dieser Möglichkeit Gebrauch gemacht wird, in einer guten und verantwortungsvollen Weise.

[1] Genaueres und Hinweise auf Literatur dazu in [Mül01], Abschnitt 1.1.

[2] Numerische Untersuchungen wie in [PSW80] legen nahe, dass Zahlen, die mit dem Miller-Rabin-Test schwer als zusammengesetzt identifiziert werden können, mit dem quadratischen Frobenius-Test bzw. dem Lucas-Test schnell als solche erkannt werden.

A. Definitionen und Nebenrechnungen

A.1. Definitionen

Hier werden einige Schreibweisen vereinbart und verschiedene allgemeine Definitionen gegeben.

Zunächst kommen wir zu einigen Dingen, die Mengen betreffen. Die Menge der natürlichen Zahlen ist in dieser Arbeit die Menge $\mathbb{N} = \{0, 1, 2, 3, 4, 5, 6, \ldots\}$, also *einschließlich* der Null. Wie üblich sind die Elemente der Menge $\mathbb{Z} = \{0, \pm 1, \pm 2, \pm 3, \ldots\}$ die ganzen Zahlen. $\mathbb{P} = \{2, 3, 5, 7, 11, 13, 17, 19, \ldots\}$ ist die Menge der Primzahlen (vgl. Abschnitt 3.4.3). $\mathbb{Q}$ ist die Menge der rationalen Zahlen, $\mathbb{R}$ die Menge der reellen Zahlen, $\mathbb{C}$ die Menge der komplexen Zahlen. Wenn jedes Element a einer Menge A in einer anderen Menge B enthalten ist, nennt man A eine Teilmenge von B. Diese Beziehung wird in der vorliegenden Arbeit mit $A \subset B$ bezeichnet. Ist A Teilmenge von B und darüber hinaus A verschieden von B, dann bezeichnet man A als *echte* Teilmenge von B. Das wird geschrieben als $A \subsetneq B$.

$\log k$ steht in dieser Arbeit immer für $\log_2 k$, den Logarithmus zur Basis 2. Die Bitlänge einer natürlichen Zahl n wird mit $\|n\|$ bezeichnet und lässt sich ausrechnen nach der Formel

$$\|n\| = \begin{cases} 1, & \text{falls } n = 0, \\ \lfloor \log_2 n \rfloor + 1, & \text{sonst.} \end{cases}$$

$\|n\|_1$ bezeichnet die Anzahl der 1en in der Binärdarstellung von n, $\|n\|_0$ die Anzahl der 0en. Ausrechnen lassen sich diese Größen mittels $\|n\|_0 = \|n\| - \|n\|_1$ und $\|0\|_1 = 0, \|2k\|_1 = \|k\|_1, \|2k+1\|_1 = \|k\|_1 + 1$.

Nun soll noch die Schreibweise für das asymptotische Verhalten von Funktionen festgelegt werden. Sei $A \subset \mathbb{R}$ eine Menge, in der der Grenzübergang $a \to \infty$ möglich ist, die also ∞ als Häufungspunkt hat. ($\mathbb{R}$ selbst, $\mathbb{N}$, die Menge der positiven reellen

und die Menge der positiven ganzen Zahlen erfüllen beispielsweise diese Voraussetzung.) $f, g : A \to \mathbb{R}$ seien Abbildungen. Die O- und o-Notation hat folgende Bedeutung:

$$\begin{aligned} f = o(g) &:\Leftrightarrow \lim_{a\to\infty} f(a)/g(a) = 0. \\ f = O(g) &:\Leftrightarrow \text{Es gibt } c \in \mathbb{N}, \text{ so dass } |f(a)| < c|g(a)| \text{ für alle } a \in A \text{ gilt.} \end{aligned}$$

Anstatt von Funktionen schreibt man gewöhnlich einen Term in einer Variable, der beschreibt, welchem Wert ein durch die Variable bezeichneter Wert durch die entsprechende Funktion zugeordnet wird. $n = O(n^2)$ ist also eine Abkürzung für $f = O(g)$, wobei $f = n \mapsto n$ und $g = n \mapsto n^2$ ist. Es ist zu beachten, dass „Gleichungen“, in denen O oder o vorkommt, nicht symmetrisch sind, sondern nur von links nach gelesen werden dürfen. Es gilt zwar $n = O(n^2)$, aber **nicht** $n^2 = O(n)$.

Eine passende Umschreibung für $f = o(g)$ ist „asymptotisch verschwindet f im Vergleich zu g“. Für $f = O(g)$ kann man sagen „asymptotisch wächst f höchstens so schnell wie g“. Die Situation „asymptotisch wächst f mindestens so schnell wie g“ wird mit $f = \Omega(g)$ bezeichnet, was als $g = O(f)$ definiert ist. Analog setzt man $f = \omega(g) \;:\Leftrightarrow\; g = o(f)$. Es gibt auch Funktionen f, g, die man als „asymptotisch äquivalent“ bezeichnen kann, was bedeutet, dass sowohl $f = O(g)$ als auch $g = O(f)$ ist. Hierfür schreibt man $f = \Theta(g)$.

Schließlich werden die Ausdrücke $o(g), O(g), \omega(g), \Omega(g)$ und $\Theta(g)$ auch benutzt, um *irgendeine* Funktion der entsprechenden Größenordnung zu bezeichnen. Eine Aussage wie „der Algorithmus benötigt $n^2 + o(\log n)$ Multiplikationen“ bedeutet also: es gibt eine Funktion f, für die $f = o(\log n)$ gilt, so dass der Algorithmus gerade $n^2 + f(n)$ Multiplikationen benötigt.

A.2. Technische Nebenrechnungen

In diesem Abschnitt werden einige Aussagen nachgerechnet.

Lemma A.1. *Seien $k > 2$ und $B > 8$ ganze Zahlen. Dann gilt:*

$$\frac{k}{2^k B} + \frac{1}{B^2} < \frac{1}{2B}$$

Beweis. Die zu beweisende Aussage ist äquivalent zu

$$kB + 2^k < 2^{k-1}B. \tag{A.1}$$

Für $k = 3, 4$ rechnen wir direkt nach, dass dies gilt: $3B+2^3 = 3B+8 < 3B+B = 2^{3-1}B$ und $4B + 2^4 = 4B + 16 < 4B + 2B \leq 2^{4-1}B$.

Nun behandeln wir den Fall $k \geq 5$. Durch vollständige Induktion nach k sieht man sofort, dass gilt

$$k < 2^{k-2} \text{ für } k \geq 5. \tag{A.2}$$

Damit ist (A.1) äquivalent zu $\frac{2^k}{2^{k-1}-k} < B$, das heißt $\frac{1}{\frac{1}{2}-\frac{k}{2^k}} < B$. Dies aber trifft tatsächlich zu, denn wegen (A.2) ist $\frac{k}{2^k} < \frac{1}{4}$, also $\frac{1}{\frac{1}{2}-\frac{k}{2^k}} < \frac{1}{\frac{1}{2}-\frac{1}{4}} = 4 < B$. □

Lemma A.2. *Sei n eine ungerade ganze Zahl. Dann ist $n^2 - 1$ durch 8 teilbar.*

Beweis. $n-1$ und $n+1$ sind zwei aufeinanderfolgende gerade Zahlen. Deshalb ist eine davon durch 4 teilbar, während die andere immerhin durch 2 teilbar ist. Damit ist das Produkt $n^2 - 1 = (n-1)(n+1)$ durch 8 teilbar. □

Lemma A.3. *Sei $k > 1$ ein ganze Zahl. Unter allen ganzen Zahlen $J > 1$ nimmt der Ausdruck*

$$\Big(1 + \frac{2^{Jk} - 1}{2^k - 1}\Big)\frac{n^2}{2^k 2^{(J+1)k}}$$

für $J = 2$ sein Maximum an, nämlich $\Big(1 + 2^k + 1\Big)\frac{n^2}{2^{4k}} = n^2\Big(\frac{1}{2^{4k-1}} + \frac{1}{2^{3k}}\Big)$.

Beweis. Der angegebene Ausdruck ist äquivalent zu

$$n^2/2^k\left(\frac{1}{2^{(J+1)k}} + \frac{2^{Jk}-1}{2^{(J+1)k}(2^k-1)}\right) = \frac{n^2}{2^k} \cdot \frac{2^k - 1 + 2^{Jk} - 1}{2^{(J+1)k}(2^k-1)}.$$

Wir untersuchen den rechten Faktor, indem wir Zähler $Z(J) := 2^{Jk} + 2^k - 2$ und Nenner $N(J) := 2^{Jk+2k} - 2^{Jk+k}$ betrachten. Es gilt $Z(J+1) = 2^{Jk+k} + 2^k - 2 < 2^{Jk+k} + 2^k(2^k - 2) = 2^k Z(J)$. Außerdem benutzen wir $N(J+1) = 2^k N(J)$. Damit folgt $\frac{Z(J)}{N(J)} > \frac{Z(J+1)/2^k}{N(J+1)/2^k} = \frac{Z(J+1)}{N(J+1)}$, also die Behauptung. □

Abbildungsverzeichnis

Verzeichnis der Algorithmen

Literaturverzeichnis

[AGP94] Alfrod, W. R., Andrew Granville und Carl Pomerance: *There are infinitely many Carmichael numbers.* Annals of Mathematics, 139(3):703–722, 1994.

[AKS02] Agrawal, Manindra, Kayal, Neerja und Saxena, Nitin: *PRIMES is in P.* `http://www.cse.iitk.ac.in/news/primality.html`, August 2002.

[Arn97] Arnault, François: *The Rabin-Monier Theorem for Lucas pseudoprimes.* Mathematics of Computation, 66(218):869–881, April 1997.

[Ble96] Bleichenbacher, Daniel: *Efficiency and Security of Cryptosystems based on Number Theory.* Doktorarbeit, ETH Zürich, 1996.

[Bos04] Bosch, Siegfried: *Algebra.* Springer, 5. Auflage, 2004.

[Coh96] Cohen, Henri: *A Course in Computational Algebraic Number Theory.* Nummer 138 in *Graduate Texts in Mathematics.* Springer, 1996.

[CP01] Crandall, Richard und Carl Pomerance: *Prime Numbers — A Computational Perspective.* Springer, 2001.

[DA99] Dierks, T. und C. Allen: *The TLS Protocol Version 1.0.* RFC 2246 (Proposed Standard), Januar 1999. Updated by RFC 3546.

[DF03] Damgård, Ivan B. und Gudmund Skovbjerg Frandsen: *An Extended Quadratic Frobenius Primality Test with Average and Worst Case Error Estimates.* Technischer Bericht RS-03-9, BRICS, Department of Computer Science, Universtity of Aarhus, Februar 2003.

[Die04] Dietzfelbinger, Martin: *Primality Testing in Polynomial Time*. Nummer 3000 in *Lecture Notes in Computer Science*. Springer, 2004.

[FS03] Ferguson, Niels und Bruce Schneier: *Practical Cryptography*. Wiley Publishing, Inc., Indianapolis, Indiana, 2003.

[GCC] *GCC Home Page*. `http://gcc.gnu.org/`.

[GMP] *The GNU MP Bignum Library*. `http://www.swox.com/gmp/`.

[GPG] *The GNU Privacy Guard*. `http://www.gnupg.org`.

[Gra98] Grantham, Jon: *A Probable Prime Test With High Confidence*. Journal of Number Theory, 72:32–47, 1998.

[Gra01] Grantham, Jon: *Frobenius Pseudoprimes*. Mathematics of Computation, 70(234):873–891, 2001.

[IR90] Ireland, Kenneth und Michael Rosen: *A Classical Introduction to Modern Number Theory*. Nummer 84 in *Graduate Texts in Mathematics*. Springer, 2. Auflage, 1990.

[Knu98] Knuth, Donald E.: *Seminumerical Algorithms*, Band 2 der Reihe *The art of computer programming*. Addison-Wesley, 3. Auflage, 1998.

[Lav] Lavoie, Rick. `http://cs-people.bu.edu/coldfury/frobenius.c`.

[Mil76] Miller, G.: *Riemann's hypothesis and tests for primality*. Journal of Computer and System Sciences, 13:300–317, 1976.

[Mül01] Müller, Siguna: *A Probable Prime Test with Very High Confidence for* $n \equiv 1 \bmod 4$. In: *ASIACRYPT 2001*, Nummer 2248 in *Lecture Notes in Computer Science*, Seiten 87–106. Springer, 2001.

[Mül03] Müller, Siguna: *A Probable Prime Test with Very High Confidence for* $n \equiv 3 \bmod 4$. Journal of Cryptology, 16:117–139, 2003.

[Mon80] Monier, L.: *Evaluation and comparison of two efficient probabilistic primality testing algorithms*. Theoretical Computer Science, 12:97–108, 1980.

[Mon92] Montgomery, Peter L.: *Evaluating recurrences of form* $X_{m+n} = f(X_m, X_n, X_{m-n})$ *via Lucas chains.* `ftp://ftp.cwi.nl:/pub/pmontgom/Lucas.ps.gz`, 1992.

[Pin98] Pinch, Rchard G. E.: *All Carmichael numbers up to* 10^{16}. `http://www.chalcedon.demon.co.uk/rgep/carmichael-16.gz`, 1998.

[PSW80] Pomerance, Carl, J. L. Selfridge und Samuel. S. Wagstaff, Jr.: *The Pseudoprimes up to* $25 \cdot 10^9$. Mathematics of Computation, 35(151):1003–1026, Juli 1980.

[Rab80] Rabin, M.: *Probabilistic algorithm for testing primality.* Journal of Number Theory, 12:128–138, 1980.

[SS71] Solovay, Robert und Volker Strassen: *A fast Monte-Carlo test for primality.* SIAM Journal on Computing, 6:74–86, 1971.

[SSH] *OpenSSH.* `http://www.openssh.org`.

[Str00] Stroustrup, Bjarne: *Die C++ Programmiersprache.* Addison-Wesley, 4. Auflage, 2000. Deutsche Ausgabe der Special Edition.

[vzGG03] Gathen, Joachim von zur und Jürgen Gerhard: *Modern Computer Algebra.* Cambridge University Press, 2. Auflage, 2003.

[Wol96] Wolfart, Jürgen: *Einführung in die Zahlentheorie und Algebra.* Aufbaukurs Mathematik. Vieweg, Braunschweig/Wiesbaden, 1996.

Thesen

1. Primzahltests sind Verfahren, die entscheiden, ob eine natürliche Zahl eine Primzahl ist. Sie sind wichtig, um Primzahlen zu finden und sich in kryptographischen Protokollen gegen Betrug abzusichern. In der Praxis werden häufig Monte-Carlo-Algorithmen mit einseitigem beschränkten Fehler als Primzahltests eingesetzt. Irrtumswahrscheinlichkeit und Laufzeit sind charakteristische Kennwerte dieser Primzahltests.

2. Der wichtigste Primzahltest ist der Miller-Rabin-Test. Die Irrtumswahrscheinlichkeit einer Iteration des Miller-Rabin-Tests beträgt höchstens $\frac{1}{4}$.

3. Der einfache quadratische Frobenius-Test besteht darin zu prüfen, ob die Gleichung $X^n \equiv b - X \bmod (n, X^2 - bX - c)$ erfüllt ist. Dabei ist n die zu untersuchende Zahl. Die Werte b, c sind Parameter, die geeignet gewählt werden müssen, was algorithmisch leicht möglich ist.

4. Der starke quadratische Frobenius-Test (QFT), der in [Gra98] beschrieben wird, geht im Wesentlichen über den einfachen quadratischen Frobenius-Test dadurch hinaus, dass er nach nichttrivialen Quadratwurzeln von 1 sucht. Man kann ihn randomisieren, so dass man den randomisierten starken quadratischen Frobenius-Test (RQFT) aus [Gra98] erhält.

5. In [Gra98] werden die Irrtumswahrscheinlichkeit und die Laufzeit des RQFT analysiert. Diese Analyse zeigt, dass die Irrtumswahrscheinlichkeit des RQFT kleiner als $\frac{1}{7710}$ ist. Auch wenn in [Gra98] bei der Analyse der Laufzeit nicht berücksichtigt wird, dass in $\mathbb{Z}_n[X]/(X^2 - bX - c)$ invertiert werden muss, wenn der QFT wie angegeben implementiert werden soll, zeigen die vorgetragenen Argumente, dass die Laufzeit des RQFT asymptotisch der von drei Iterationen des Miller-Rabin-Tests entspricht.

6. Implementiert man den RQFT wie in [Gra98] vorgeschlagen und unterzieht Zahlen mit Bitlänge bis zu 4096 diesem Test, dann stellt man fest, dass eine Iteration des Tests praktisch die Zeit von vier Miller-Rabin-Test-Iterationen benötigt.

7. In [CP01] wird eine Technik für die Implementierung des einfachen quadratischen Frobenius-Tests beschrieben. Bei Experimenten mit dieser Implementierung und Zahlen der Länge bis 4096 Bit benötigt der einfache quadratische Frobenius-Test etwa die Zeit von drei Iterationen des Miller-Rabin-Tests.

8. Man kann die Implementierungstechnik für den einfachen quadratischen Frobenius-Test aus [CP01] in den zeitkritischen Teil der Implementierung des RQFT nach [Gra98] einsetzen und erhält so eine etwas schnellere Implementierung des RQFT. In Experimenten mit Zahlen der Länge bis 4096 Bit benötigt dann eine Iteration des RQFT nicht mehr Zeit als drei Iterationen des Miller-Rabin-Tests.

[CP01] Crandall, Richard und Carl Pomerance: *Prime Numbers — A Computational Perspective*. Springer, 2001.

[Gra98] Grantham, Jon: *A Probable Prime Test With High Confidence*. Journal of Number Theory, 72:32–47, 1998.

..... Ilmenau, 2. September 2005
Christian Hoffmann

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich diese Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe.

..... Ilmenau, 2. September 2005
Christian Hoffmann