Analyse der Sicherheit des RSA-Algorithmus

Mögliche Angriffe, Implementierungsfragen und ökonomische Konsequenzen

Diplomarbeit von Matthias Schneider

Erstprüfer:
Prof. Dr. sc. techn. Manfred Grauer

Zweitprüfer:
Bernhard Esslinger

Bearbeitungszeitraum:
09.06.04 – 09.10.04

Danksagung
Ich danke Prof. Dr. Manfred Grauer für seine Unterstützung, Bernhard Esslinger für die gute Betreuung und Henrik Koy für wertvolle Anregungen. Außerdem danke ich Dr. Alexander May für die Zurverfügungstellung des Quellcodes zu dem von ihm mitentwickelten Angriff.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Der von Ronald L. Rivest, Adi Shamir und Leonard Adleman entwickelte RSA-Algorithmus ist das bekannteste und meist verwendete Public Key Verschlüsselungsverfahren der Welt. Es stellt die tragende Säule für einen Großteil der implementierten IT-Sicherheitssysteme dar. Seit seiner Veröffentlichung im August 1977 wurde seine Sicherheit ausführlich untersucht. Dabei wurden viele zum Teil theoretische aber auch praktische Angriffe entwickelt.[1] Trotz der sehr einfachen Struktur des RSA-Algorithmus ist daher eine sichere Implementierung nicht trivial.

Die Sicherheit des zu Grunde liegenden mathematischen Verfahrens hängt maßgeblich von der Schwierigkeit ab, große Zahlen zu faktorisieren. Auch wenn in diesem Gebiet schon viel Forschungsarbeit geleistet wurde, geben aktuelle Ergebnisse Grund zur Sorge.

Auch Angriffe, die sich eines Seitenkanals wie etwa der Berechnungszeit oder dem Stromverbrauch einer RSA-Operation bedienen, müssen bei der Beurteilung der Sicherheit von RSA berücksichtigt werden.

In den letzten Jahren ist eine ganze Klasse von neuen Angriffen entwickelt worden, die auf dem mathematischen Verfahren der Gitterbasenreduktion fußen. Erstmals vorgestellt von Johan Håstad [Hås88], wurde die Idee weiterentwickelt von Don Coppersmith [Cop96a, Cop96b]. Mittels dieser Technik sind viele neuartige Angriffe entstanden, die bei einer sicheren Implementierung zu berücksichtigen sind.

In dieser Arbeit soll ein umfassendes Bild der publizierten Angriffe auf RSA gegeben werden. Neben der Faktorisierung und den verschiedenen Seitenkanalangriffen soll besondere Aufmerksamkeit den Gitterangriffen gelten, die bislang in der Öffentlichkeit keine sehr große Beachtung gefunden haben. Dabei soll neben der Darstellung der verschiedenen Verfahren auch gezeigt werden, welche Auswirkungen sie auf vorhandene Standards und Implementierungen hatten und wie groß die Bedrohung ist, die sie für RSA darstellen.

Um die praktische Relevanz der Gitterangriffe zu verdeutlichen, wurden ausgewählte

[1] Eine gute Übersicht zu den Angriffen bis 1999 gibt [Bon99].

Angriffe implementiert. Durch das Einbringen der bislang wenig bekannten Angriffe in das freie eLearning-Programm CrypTool[2] sollen sie mehr in den Fokus der Aufmerksamkeit der Anwender und Entwickler von RSA-basierten Lösungen gerückt werden. CrypTool wird als Open-Source-Projekt entwickelt und verwendet, um das Wissen um Kryptographie von seinen Anfängen bis zu hochaktuellen Themen einem breiten Publikum zu vermitteln.

1.2 Bedeutung von RSA

Dieser Abschnitt diskutiert die Bedeutung von RSA in seinen verschiedenen Einsatzbereichen. In diesem Zusammenhang werden zunächst seine zwei wichtigsten Anwendungen erläutert. Anschließend wird ein Überblick über deren Verwendung gegeben, um die große Bedeutung von RSA für die verschiedenen Bereiche der IT-Sicherheit aufzuzeigen.

1.2.1 Asymmetrische Verschlüsselung

Das RSA-Kryptosystem ist die Public Key Kryptosystem-Implementierung, die heute die größte Bedeutung erlangt hat. Dabei werden zum Ver- und Entschlüsseln zwei verschiedene Schlüssel verwendet, die als öffentlicher und privater Schlüssel bezeichnet werden. Eine zentrale Funktionsweise von RSA ist, dass eine Nachricht, die mit einem dieser Schlüssel verschlüsselt wurde, nur mit dem anderen wieder entschlüsselt werden kann. Das birgt einige Vorteile:

- Empfänger und Sender der Nachricht müssen sich nicht, wie bei symmetrischer Verschlüsselung, über einen sicheren Kanal auf einen gemeinsamen Schlüssel einigen. Der *Empfänger* schickt zunächst einen von ihm generierten öffentlichen Schlüssel authentisch an alle Kommunikationspartner. Diese können nun ihre Nachrichten an den Empfänger damit verschlüsseln und nur dieser kann sie entschlüsseln. Dadurch wird die Schlüsselverteilung gegenüber symmetrischen Verfahren sehr einfach.

- Es wird für jeden Kommunikationsteilnehmer nur ein Schlüsselpaar benötigt, während bei symmetrischer Verschlüsselung jeder Teilnehmer für jeden anderen Teilnehmer einen gemeinsamen geheimen Schlüssel braucht. Die Schlüsselverwaltung wird also ebenfalls stark vereinfacht.

In der Praxis werden Daten nicht direkt mit asymmetrischen Verfahren verschlüsselt, da diese langsamer sind als vergleichbar sichere symmetrische Verschlüsselungsverfahren.

[2] Informationen und die aktuelle Version der CrypTool-Software findet man auf http://www.CrypTool.de bzw. http://www.CrypTool.com.

Statt dessen werden so genannte Hybridverfahren eingesetzt, bei denen ein zufälliger Sitzungsschlüssel erzeugt wird, mit dem die Daten symmetrisch verschlüsselt werden. Dieser Sitzungsschlüssel wird dann mit dem öffentlichen Schlüssel verschlüsselt und übertragen. Die eigentliche Kommunikation wird mit dem schnelleren symmetrischen Verfahren verschlüsselt.

1.2.2 Digitale Signaturen

Wie eine händische Unterschrift, gewährleistet die digitale Signatur die Authentifizierung und Integrität einer Nachricht. Dazu kann man sich vereinfacht vorstellen, dass auf die Nachricht mit dem privaten Schlüssel des Absenders die Signaturoperation angewendet wird. Als Ergebnis erhält man die elektronische Signatur. Diese kann mit dem öffentlichen Schlüssel geprüft werden. Dazu wird die Signatur invertiert und man erhält die ursprüngliche Nachricht. Das Paar Signatur und Nachricht dient somit zur Prüfung der Integrität der Nachricht. Ist die Authentizität des öffentlichen Schlüssels sichergestellt, so gewährleistet die Signatur auch die Authentizität der Nachricht. Tatsächlich wird aus Gründen der Performance allerdings nicht die gesamte Nachricht, sondern ihr Hashwert signiert.[3] Der signierte Hashwert wird als Signatur der Nachricht bezeichnet und mit der unverschlüsselten Nachricht an den Empfänger gesendet. Der Empfänger vergleicht nun die invertierte Signatur mit dem von ihm erzeugten Hashwert und kann so die Integrität und Authentizität der Nachricht überprüfen.

Die authentische Verbreitung der öffentlichen Schlüssel geschieht über so genannte Public Key Infrastructures (PKI), wobei die Kombination aus Name und Schlüssel von anderen Teilnehmern oder Zertifizierungsstellen geprüft und mittels Zertifikat bestätigt wird.

1.2.3 Anwendungsbereiche

Das RSA-Kryptosystem durchdringt alle Bereiche des Internet. Es wird zur Sicherung der Kommunikation in verschiedenen Netzwerk Protokollschichten eingesetzt. In der Transportschicht kommt das von Netscape entwickelte Secure Session Layer (SSL) Protokoll zum Einsatz, das 1999 in den Transport Layer Security (TLS) Standard [DA99] einging. Dieses Protokoll nutzt zum Schlüsselaustausch RSA-Verschlüsselung. Zur Authentifizierung des Servers wird ebenfalls fast immer ein Zertifikat verwendet, dass auf dem

[3] Kryptographische Hashverfahren bilden Nachrichten beliebiger Länge auf einen Hashwert fester Länge ab, die eine Art Fingerabdruck der Nachricht darstellen. Diese Funktionen sind nicht umkehrbar und es ist praktisch unmöglich, zwei Nachrichten mit demselben Hashwert zu erzeugen.

RSA-Verfahren basiert. Es besteht zusätzlich auch die Möglichkeit die Authentizität des Clients mittels Zertifikat zu prüfen.

Im B2C sind hier die häufigsten Anwendungen sicherlich die Sicherung des elektronischen Zahlungsverkehrs und sicherheitsrelevanter Daten bei Online-Shopping und Homebanking. Der Kunde besucht die Webseite des Anbieters und der Browser kann anhand des Zertifikates dieser Seite die Authentizität des Anbieters überprüfen. Anschließend wird mit dem öffentlichen Schlüssel aus dem Zertifikat ein Sitzungsschlüssel vereinbart. Aufgrund mangelnder Verbreitung wird bei B2C bislang meist darauf verzichtet, die Identität des Kunden mittels Zertifikat zu überprüfen. Hier wird in den meisten Anwendungen lediglich eine Passwortabfrage durchgeführt.

SSL wird auch im Zusammenhang mit Web-Frontends von betrieblichen Anwendungssystemen verwendet, die zunehmend Verbreitung finden. Zum Beispiel stelle man sich einen Außendienstmitarbeiter vor, der aus dem Netzwerk eines Kunden heraus über das Internet auf aktuelle Verkaufzahlen aus dem ERP-System zugreift, ohne dass dabei die Kommunikation abgehört werden kann.

Da SSL auf der Transportschicht arbeitet, ist es unabhängig vom konkreten Anwendungsprotokoll. Daher können neben HTTP die verschiedensten Anwendungsprotokolle wie SMTP, FTP, TELNET u.a. oberhalb von SSL verwendet werden.

Auch in der Netzwerkschicht wird RSA verwendet. So kann es im Internet Protocol Security (IPSec) Framework verwendet werden, um beispielsweise Virtual Private Networks (VPN) zu sichern. Hier wird es zur Authentifizierung der Endpunkte und zum Schlüsselaustausch eingesetzt.

VPNs werden häufig zur Verbindung von Unternehmensstandorten über das Internet verwendet. Im Zuge des fortschreitenden Supplychain-Management[4] und der damit verbundenen geforderten Integration von Produktdaten über Unternehmensgrenzen hinweg ist eine Sicherung der Kommunikation zwischen Zulieferern und Kunden über VPN möglich.

In der Anwendungsschicht gibt es ebenfalls Protokolle, die unter anderem RSA verwenden. Dazu gehört zum Beispiel S/MIME, das zur Sicherung von E-Mail-Kommunikation verwendet wird, oder SET, ein von Visa und MasterCard entwickeltes Framework zum Schutz von Kreditkarteninformationen im Internet.

Neben dieser reinen Kommunikationssicherung kommt RSA im Zusammenhang mit digitalen Signaturen in vielen anderen Anwendungsbereichen eine tragende Rolle zu. Das am 1. August 1997 in Deutschland in Kraft getretene Gesetz zur digitalen Signatur

[4] Unter einer Supplychain (deutsch: *Versorgungskette*) versteht man die gesamte Kette der an der Herstellung eines Produktes beteiligten Firmen.

(SigG) regelt die rechtlichen Rahmenbedingungen für die Rechtsverbindlichkeit digitaler Signaturen in Deutschland. Die Regulierungsbehörde für Telekommunikation und Post empfiehlt in [Reg98] unter anderem das RSA-Verfahren für die Verwendung digitaler Signaturen nach diesem Gesetz.

Die Anwendungen für digitale Signaturen sind sehr vielfältig. Ohne Anspruch auf Vollständigkeit seien die folgenden Beispiele für den Einsatz digitaler Signaturen genannt:

- Elektronische Rechnungslegung
- Signierung von Softwareupdates und Downloads
- JobCard für Arbeitnehmer
- Elektronischer Arztausweis HPC (Health Professional Card)
- Signatur biometrischer Daten auf Personalausweisen

Die meisten Signaturverfahren, die heute im Einsatz sind, basieren auf RSA-Technik.

Frank Bourseau, Dirk Fox und Christoph Thiel kommen in [BFT02] zu dem Schluss, dass RSA heute der defacto-Standard für asymmetrische Kryptoverfahren ist und sich daran bis 2012 wenig ändern wird. Für die große Verbreitung von RSA machen sie unter anderem die vergleichsweise einfache Struktur des Verfahrens und das Einbringen von RSA in viele nationale und internationale Standards durch die Firma RSA Security verantwortlich. Sie beobachten eine zunehmend starke Verbreitung des Verfahrens seit Ende der 90er Jahre und führen dies auf das Auslaufen des von dieser Firma gehaltenen Patents im Jahre 2000 zurück. In dieser weiten Verbreitung sehen die Autoren allerdings auch eine große Gefahr: Da es keine etablierte Alternative zu RSA gibt, ist mit einem möglichen Bruch von RSA die Sicherheit aller Lösungen gefährdet, die asymmetrische Kryptographie verwenden. Dadurch motivieren sie die Dringlichkeit, Alternativen zu suchen und weiterzuentwickeln.

1.3 Überblick

In dieser Arbeit soll ein Gesamtbild über das RSA-Kryptosystem gegeben werden. Dazu wurde in Abschnitt 1.2 die Relevanz und Verbreitung des Verfahrens im Bereich der Informationsverarbeitung aufgezeigt, um die wirtschaftliche Bedeutung von RSA zu verdeutlichen.

Kapitel 2 befasst sich mit der theoretischen Sicherheit der dem RSA-Verfahren zugrunde liegenden Einwegfunktion. Dazu wird zunächst der RSA-Algorithmus vorgestellt und anschließend eine Abschätzung der gegenwärtigen und zukünftigen Sicherheit gegeben.

Kapitel 3 stellt anschließend die auf Gitterbasenreduktion basierenden Angriffe detailliert dar. Dazu wird zunächst eine mathematische Einführung gegeben und das Kernverfahren, auf dem die meisten dieser Angriffe basieren, vorgestellt. Darauf aufbauend werden alle bekannten Angriffe dieser Klasse gezeigt und mögliche Gegenmaßnahmen erläutert.

Kapitel 4 fasst diese und weitere bekannte Angriffe zusammen und geht auf ihren Einfluss auf eine sichere Implementierung ein. Zunächst werden Einschränkungen auf die Wahl der Parameter diskutiert, die sich unter anderem aus den Gitterangriffen ergeben. Anschließend werden die so genannten Seitenkanalangriffe vorgestellt und ihre Relevanz für eine sichere Implementierung gezeigt. Der letzte Abschnitt dieses Kapitels zeigt die Bedeutung eines geeigneten Füllschemas für die Sicherheit der Implementierung auf.

Kapitel 5 diskutiert die verschiedenen Arten von ökonomischen Konsequenzen, die die vorgestellten Angriffe haben können.

Abschließend wird in Kapitel 6 ein Fazit gezogen. Darin wird zusammenfassend eine Übersicht und Beurteilung der Bedrohung für das RSA-Kryptosystem und die sich ergebenden Implikationen für die Sicherheit gegeben. Anschließend wird auch eine Abschätzung auf zukünftige Entwicklungen in dem Betrachtungsraum aufgestellt.

Eine Aufgabe dieser Arbeit war es, ausgewählte Gitterangriffe zu implementieren und in das eLearning-Programm CrypTool einzubringen. Anhang A beschreibt die Implementierungen dieser Angriffe und stellt Laufzeiten vor.

2 Die Sicherheit des RSA-Kryptosystems

Um die Sicherheit des RSA-Algorithmus zu untersuchen, ist es zunächst notwendig, das RSA-Verfahren zu erläutern. Anschließend wird eine Beziehung zwischen der theoretischen Sicherheit von RSA und der Faktorisierung großer Zahlen hergeleitet. Angriffe auf die Implementierung des RSA-Kryptosystems werden in den folgenden Kapiteln beschrieben. Hier geht es darum, die theoretische Sicherheit der zugrunde liegenden mathematischen Trapdoorfunction (deutsch: *Falltürfunktion*) zu erläutern. Dieses Kapitel beschäftigt sich daher ausschließlich mit dem Faktorisierungsangriff und versucht eine Abschätzung auch der zukünftigen Sicherheit von RSA in Abhängigkeit von der Schlüssellänge zu geben.

2.1 Das RSA-Verfahren

Wie im letzten Kapitel schon eingeführt wurde, gibt es beim RSA-Verfahren zwei Schlüssel: Einen öffentlichen und einen privaten. Der öffentliche Schlüssel besteht aus einer positiven ganzen Zahlen, die in der gesamten Arbeit immer mit e bezeichnet wird. Der private Schlüssel besteht ebenfalls aus einer positiven ganzen Zahl d. Darüber hinaus wird für die Ver- und Entschlüsselung noch ein Parameter N benötigt. Um ein neues Schlüsselpaar zu erzeugen, werden zunächst zwei Primzahlen p und q gewählt. Daraus wird $N = p \cdot q$ berechnet. Nun wird $\Phi(N) = (p-1)(q-1)$ berechnet, wobei $\Phi(N)$ die Eulersche Φ-Funktion bezeichnet. Entweder e oder d kann nun frei gewählt werden, es muss lediglich teilerfremd zu $\Phi(N)$ sein. Dann berechnet man den jeweils anderen Schlüssel e oder d so, dass die folgende elementare Gleichung des RSA-Systems gilt:

$$e \cdot d \equiv 1 \mod \Phi(N) \tag{2.1}$$

Somit sind d und e multiplikativ invers zueinander modulo $\Phi(N)$. Wird e gewählt, kann d mit dem erweiterten Euklidischen Algorithmus effizient berechnet werden und umgekehrt. Mit dem Satz von Euler kann gezeigt werden, dass für so gewählte e, d und N stets gilt:

$$(m^e)^d \equiv m \mod N, \qquad \text{für } m = 0, \ldots, N-1 \tag{2.2}$$

Zum Verschlüsseln einer Nachricht $m < N$ wird der Chiffretext c als $c = m^e \mod N$ berechnet. Der Chiffretext ist also die e-te Potenz von m modulo N. Der Empfänger berechnet aus diesem c wieder die Originalnachricht m, indem er seinerseits $m = c^d \mod N$ berechnet. Mit Gleichung 2.2 lässt sich der Vorgang nachvollziehen:

$$c = m^e \mod N \tag{2.3}$$

$$c^d = (m^e)^d = m \mod N \tag{2.4}$$

Die Anwendbarkeit dieses Verfahrens zur Kryptographie beruht im Wesentlichen darauf, dass die Berechnung von $m^e \mod N$ ohne die Kenntnis der Faktorisierung von N nicht umkehrbar ist. Man spricht hier auch von einer Trapdoorfunction. Jeder, der den öffentlichen Schlüssel (e, N) kennt, kann $c = m^e \mod N$ berechnen, aber nur wer den privaten Schlüssel (d, N) kennt, kann die Berechnung umkehren.

Bei der Signaturerstellung wird das Verfahren umgekehrt. Zunächst wird mit einem geeigneten Hash-Verfahren ein Fingerabdruck der Nachricht erzeugt. Auf diesen Fingerabdruck wendet der Unterzeichner nun die Verschlüsselungsfunktion mit dem *privaten* Schlüssel (d, N) an. Zur Überprüfung der Signatur genügt es, die Entschlüsselungsfunktion mit dem öffentlichen Schlüssel (e, N) auf die Signatur anzuwenden. Man erhält den Fingerabdruck der originalen Nachricht und vergleicht diesen mit dem Fingerabdruck der erhaltenen Nachricht.

Bei geeigneten Hashverfahren ist es praktisch ausgeschlossen, dass zwei Nachrichten denselben Hashwert haben. Somit kann nur der Besitzer des geheimen Schlüssels die Signatur so erstellen, dass sie, mit dem öffentlichen Schlüssel entschlüsselt, diesen Hashwert liefert.

2.2 Die Faktorisierung von N

Es ist klar, dass die Kenntnis der Faktoren p und q sehr schnell den geheimen Exponenten d liefert. Daraus folgt, dass RSA gebrochen ist, wenn der RSA-Modulus N nur mit Kenntnis von e und N effizient faktorisiert werden kann. Wir nehmen daher im Folgenden an, dass das Problem RSA zu brechen äquivalent ist zu dem Problem N zu faktorisieren. Diese Äquivalenz ist nicht bewiesen und es gibt Untersuchungen, die Zweifel daran rechtfertigen [BV98]. Allerdings ist kein Verfahren bekannt, dass $m^{\frac{1}{e}} \mod N$ ohne die Kenntnis der Faktoren von N effizienter berechnen kann, als die Faktorisierung selbst. Die theoretische Sicherheit des RSA-Kryptosystems hängt also von zwei grundlegenden Annahmen ab: Erstens, dass es keine Möglichkeit gibt, die e-te Wurzel modulo N effizi-

ent zu berechnen, ohne die Faktorisierung von N zu kennen und zweitens, dass es sehr schwer ist, N zu faktorisieren.

Dem Problem der Faktorisierung soll nun auf den Grund gegangen werden. In der Literatur versucht man die Fragen zu klären, welche Schlüssellängen benutzt werden sollten und wie lange diese voraussichtlich sicher sein werden. In [LV01] werden von Lenstra und Verheul verschiedene Hypothesen für zukünftige Entwicklungen formuliert und ein umfassenden Modell für Abschätzungen der Sicherheit von Schlüssellängen erarbeitet. Die dort getroffenen Annahmen werden auch von Ferguson und Schneier als die bei weitem besten bekannten bezeichnet [FS03, S. 217]. Hier wird ein sehr viel einfacherer Ansatz verwendet, um einen Überblick über die Entwicklung zu geben. Zunächst werden ähnlich wie in [BFT02] historische Ergebnisse betrachtet. Danach wird versucht daraus eine Abschätzung auf die Zukunft zu geben. Anschließend wird anhand aktueller Entwicklungen die erstellte Prognose nochmal kritisch geprüft.

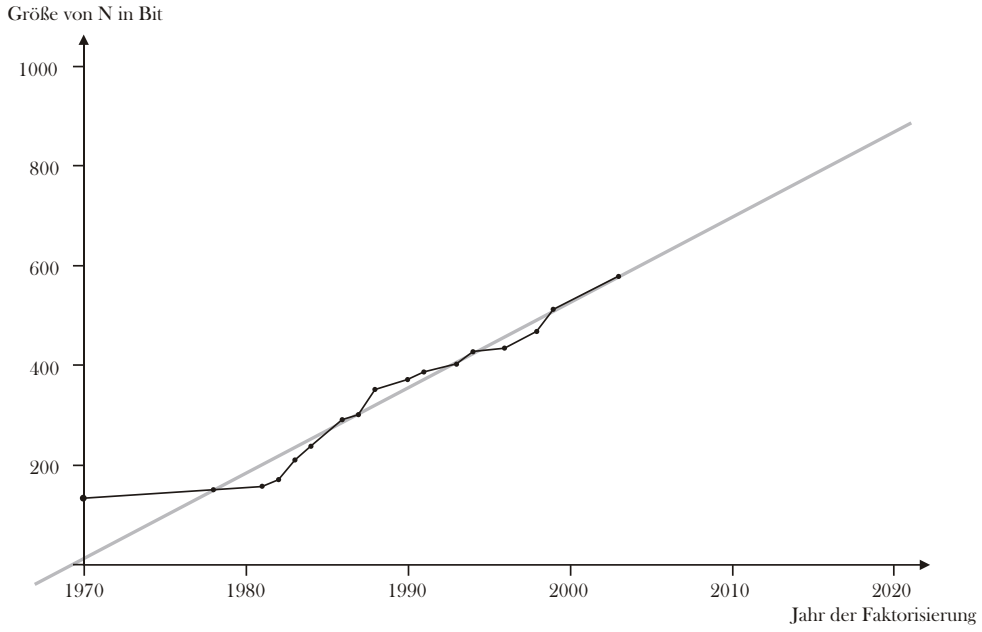

Abbildung 2.1: Faktorisierungsrekorde nach Jahren. Daten aus [BFT02], ergänzt um die Faktorisierung der 576 Bit Zahl vom 3. Dezember 2003.

Sieht man sich die publizierten Faktorisierungsrekorde an, so stellt man eine annähernd lineare Entwicklung zur Bitlänge des RSA-Schlüssels fest. Abbildung 2.1 zeigt die Rekorde für die Faktorisierungen der entsprechenden Jahre, die bis heute publiziert wur-

den. Die graue Gerade zeigt die lineare Ausgleichsgerade der Rekorde von 1984 bis heute. Nun soll untersucht werden, wie diese konstante Verbesserung zu erklären ist. Vor dem Hintergrund der vielen Faktoren, die für diese Entwicklung verantwortlich sind, scheint ein derart linearer Verlauf der Entwicklung eher zufällig zu sein. Im Wesentlichen gibt es zwei Faktoren, die den Fortschritt bei der Faktorisierung begünstigen:

1. Entwicklung neuer und Verbesserung bekannter Algorithmen zur Faktorisierung
2. Verbesserung der verfügbaren Hardware

Bei den **Algorithmen** hat es nach Weis, Lucks und Bogk [WLB02] in den letzten 25 Jahren große Fortschritte gegeben. Der für die Faktorisierung von RSA-Moduln stärkste Algorithmus ist das (Generalised) Number Field Sieve (GNFS) (deutsch: *Zählkörpersieb*). Es besitzt eine heuristische asymptotische Laufzeit von $o(n) = e^{(c+o(1))\cdot\ln(n)^{1/3}\cdot\ln(\ln(n))^{2/3}}$, wobei für hinreichend große Zahlen der Faktor $ln(n)^{\frac{1}{3}}$ dominiert. Die Laufzeit ist also subexponentiell und nimmt daher mit steigender Bitlänge sehr viel langsamer zu als die Laufzeit einer erschöpfenden Suche. In den vergangenen zehn Jahren gab es laut Weis, Lucks und Bogk aber kaum Verbesserungen auf diesem Gebiet. Das spricht eigentlich für eine Verlangsamung der Faktorisierungserfolge.

Betrachtet man die Entwicklung der **Hardware**, kann man das „Mooresche Gesetz“ zugrunde legen, das besagt, dass sich die Dichte integrierter Schaltungen etwa alle 18 Monate verdoppelt. Dieses Beobachtung hat sich bis heute bewahrheitet, für die Zukunft wird aber ein Rückgang dieses Wachstums erwartet.

Diese Beobachtungen könnten zu der Annahme führen, dass die Entwicklung dieser Kurve in Zukunft vergleichbar verläuft oder sogar abnimmt. Das würde bedeuten, dass heute stark verbreitete Schlüssel mit 1024 Bit Länge auch noch über das Jahr 2020 hinaus nicht faktorisiert werden könnten. Dabei ist allerdings die Annahme, dass die publizierten Faktorisierungserfolge die tatsächlichen Möglichkeiten reflektieren, vor dem Hintergrund von Geheimorganisationen mehr als fragwürdig.

Alle bisher angestellten Überlegungen basieren auf der Annahme, dass Standardhardware verwendet wird. Adi Shamir schlug 1999 ein optoelektronisches Gerät namens TWINKLE vor, mit dem der Siebungsschritt, der laufzeitintensivste Teil des GNFS, stark beschleunigt werden kann [Sha99]. Nach Weis, Lucks und Bogk galten die Kostenabschätzungen für dieses Gerät jedoch als zu niedrig. Lenstra und Verheul kamen in ihrer Veröffentlichung 2001 sogar zu dem Schluss, dass, nach damaligem Kenntnisstand, Spezialhardware wahrscheinlich keinen merklichen Einfluss auf die Sicherheit von RSA Moduln haben wird [LV01, S. 8].

Im Jahr 2003 veröffentlichten Adi Shamir und Eran Tromer in [AS03] ein neues Design für Hardware namens TWIRL, das unter anderem auf einer Veröffentlichung von Bernstein [Ber01] beruht. Dieses Design ist laut Weis, Lucks und Bogk Anlass zu einer Neubewertung der Sicherheit von RSA. So ist nach Shamir und Tromer ein auf dieser Technik basierendes Gerät für nur 10 Millionen EUR in der Lage, einen 1024 Bit langen Schlüssel in nur einem Jahr zu faktorisieren. Für einen 512 Bit Schlüssel braucht ein TWIRL-basiertes Gerät für 10 Tausend EUR weniger als 10 Minuten. Aufgrund der Parallelisierbarkeit des Designs ist mit einem 120 Millionen EUR teuren Gerät die Faktorisierung eine 1024 Bit langen Zahl in weniger als einem Monat möglich. Allerdings konnten Shamir und Tromer noch kein lauffähiges Gerät präsentieren und einige Teile des verwendeten Algorithmus können noch nicht in Spezialhardware realisiert werden. Wird jedoch eine andere Variante des GNFS verwendet, scheint das Design implementierbar, wobei sich dann die Kosten für ein Gerät, das eine 1024 Bit lange Zahl in einem Jahr faktorisiert, auf 50 Millionen EUR erhöht. Die geschätzten Kosten für dieses Gerät sind laut Weis, Lucks und Bogk jedoch noch Gegenstand der wissenschaftlichen Diskussion. Man kann jedoch davon ausgehen, dass 50 Millionen EUR und mehr für große Geheimdienste oder etwa das organisierte Verbrechen eine lohnende Investition darstellen können.

Weis, Lucks und Bogk leiten daraus die Empfehlung ab, Schlüssel von 1024 Bit Länge oder weniger durch solche mit mindestens 2048 Bit zu ersetzen. Die Regulierungsbehörde für Telekommunikation und Post nennt 1024 Bit als Mindestlänge für Schlüssel bis Ende 2007, die empfohlene Länge ist aber auch hier 2048 Bit. Die RSA Laboratories und das National Institute of Standards and Technology (NIST) der USA empfehlen spätestens 2010 bzw. 2015 Schlüssel mit 2048 Bit zu verwenden. Bei diesen Empfehlungen müssen natürlich die individuellen Sicherheitsanforderungen einbezogen werden. Allen Empfehlungen gemein ist jedoch, dass spätestens in elf Jahren 1024-Bit-Schlüssel nicht mehr verwendet werden sollten. Eine 50 Millionen EUR Maschine würde daher nur eine geringe Anzahl von Faktorisierungen durchführen können, bevor längere Schlüssel allgemein verwendet werden. Daher ist ein solcher Angriff wohl lediglich für Schlüssel interessant, die einen Wert von mehreren Millionen EUR haben.

Bei der Erhöhung der Schlüssellänge ist abhängig von der Anwendung zu prüfen, wann welche Schlüssellänge eingeführt werden sollte. So kann die Länge auch sukzessiv erhöht werden, um den Geschwindigkeitsvorteil kürzerer Schlüssel länger zu nutzen. Es gilt, die Anzahl der Umrüstungen unter Berücksichtigung des erhöhten Rechenaufwands bei Verwendung längerer Schlüssel zu mimieren. Langfristig wird eine Umstellung unvermeidlich sein, daher sollten alle neuen Sicherheitsprodukte laut [FS03] jetzt schon mindestens 2048 Bit verwenden und nach Möglichkeit 8192 Bit unterstützen, um auf zukünftige Entwick-

lungen flexibler reagieren zu können.

Eine weitere große Unbekannte in diesem Zusammenhang stellen so genannte Quantencomputer dar. Dabei handelt es sich um einen völlig neuartigen Computer, der auf den Gesetzen der Quantentheorie basiert. Peter W. Shor zeigte in [Sho94] wie eine solche Maschine aussehen kann und wie damit große Zahlen effizient faktorisiert werden können. Bis heute ist zwar noch nicht bekannt, ob solch eine Maschine in der entsprechenden Größe jemals gebaut werden kann, jedoch ist laut Buchmann und Takagi [BT04] klar, dass in diesem Fall völlig neue Public-Key-Verfahren benötigt würden. *„Wenn man die erst sucht, wenn große Quantencomputer schon gebaut werden können, ist es zu spät.“* [BT04, S. 6] Um nachhaltige Sicherheit zu gewährleisten, sollte man die heute als sicher geltenden Verfahren verwenden, jedoch Alternativen vorbereiten und Infrastrukturen schaffen, die einen einfachen Austausch unsicherer Komponenten ermöglichen. Interessanterweise wird in der gleichen Veröffentlichung ein quantensicheres Kryptoverfahren von Micciancio erläutert, das auf der Gittertheorie basiert, jedoch für die praktische Anwendung viel zu langsam ist.

3 Angriffe unter Zuhilfenahme von Gitterbasenreduktion

In diesem Kapitel werden die verschiedenen Angriffe erläutert, die sich der Gitterbasenreduktion bedienen. Obwohl sie in der Öffentlichkeit noch wenig Beachtung finden, stellen sie für einfache RSA-Implementierungen eine reale Bedrohung dar. Dieses Kapitel soll kurz in die zugrundeliegende Mathematik einführen und anschließend die Angriffe ausführlich beschreiben, um ein tieferes Verständnis für diese relativ neue Art der Angriffe zu schaffen.

Dazu wird in Abschnitt 3.1 zunächst eine kurze Einführung zur Gitterreduktion und ihre Anwendung zum Lösen von Polynomen modulo N gegeben. Unterabschnitt 3.1.6 fasst diese Verfahren noch einmal kurz zusammen und veranschaulicht ihre Verwendung in den verschiedenen Angriffen.

Die darauf folgenden Abschnitte beschreiben den aktuellen Stand der Veröffentlichungen zu diesen Angriffen. Diese lassen sich dabei in die folgenden vier Kategorien unterscheiden, die in dieser Reihenfolge behandelt werden:

1. Angriffe auf Schemata, die sehr kleine öffentliche Schlüssel e verwenden (z.B. $e = 3$).
2. Angriffe auf kurze geheime Exponenten (z.B. $d < N^{0,5}$).
3. Faktorisierung von N, wenn einer der Faktoren p oder q teilweise bekannt ist.
4. Angriffe, die voraussetzen, dass ein Teil des geheimen Schlüssels d bekannt ist.

3.1 Einführung in die Gitterreduktion

Dieser Abschnitt befasst sich mit dem mathematischen Grundlagen der Gitterangriffe. Zunächst werden die nötigen Definitionen und Sätze vorgestellt. Beweise werden nur dort geführt, wo sie dem Verständnis dienen. Für weiterführende Beweise und mathematische Ausführungen sei auf [Cas71] verwiesen. In diesem Abschnitt wird auch das Verfahren der Gitterbasenreduktion eingeführt. Im Besonderen wird der so genannte LLL-Algorithmus

von Lenstra, Lenstra und Lovasz [LLL82] in seinen Ergebnissen präsentiert. Im Abschnitt 3.1.4 wird das grundlegenden Verfahren vorgestellt, auf denen die späteren Angriffe aufbauen: Das Lösen univariater modularer Gleichungen. Dieses wird in Abschnitt 3.1.5 auf den multivariaten Fall erweitert.

3.1.1 Notationen

In dieser Arbeit wird die Menge der ganzen Zahlen mit $\mathbb{Z}$ und die Menge der reellen Zahlen mit $\mathbb{R}$ bezeichnet. $\mathbb{R}^d$ bezeichnet den euklidischen Vektorraum mit dem Euklidischen Skalarprodukt $\langle\ ,\ \rangle$, der Euklidischen Norm $\|v\| := \sqrt{\langle v, v\rangle}$ und der 1-Norm $\|v\|_1 = \sum |v_i|$. Eine n-elementige geordnete Menge von Vektoren aus $\mathbb{R}^d$ wird in der Form $b_1, \ldots, b_n$ geschrieben. Mit $\mathbb{R}[x]$ wird die Menge der univariaten Polynome über $\mathbb{R}$ bezeichnet, also Polynome der Form $\sum_{i=0}^{g} a_i x^i = a_g x^g + a_{g-1} x^{g-1} + \ldots + a_0$ mit $a_i \in \mathbb{R}$ in einer Veränderlichen. Entsprechend ist $\mathbb{R}[x, y]$ die Menge der bivariaten und allgemein $\mathbb{R}[x_n, \ldots, x_1]$ die Menge der multivariaten Polynome. Für $x \in \mathbb{R}$ bezeichnet $\lfloor x \rfloor$ die größte ganze Zahl kleiner als x entsprechend ist $\lceil x \rceil$ die kleinste ganze Zahl größer als x. $\binom{a}{b}$ bezeichnet den Binomialkoefizienten.

3.1.2 Grundlegendes über Gitter

Definition 3.1. *L heißt Gitter in $\mathbb{R}^d$, wenn es $n \leq d$ linear unabhängige Vektoren $b_1, b_2, \ldots, b_n$ linear aus $\mathbb{R}^d$ gibt mit*

$$L := \left\{ y \in \mathbb{R}^d \middle| y = \sum_{i=1}^{n} a_i b_i, a_i \in \mathbb{Z} \right\}.$$

Man bezeichnet diese b_i als die Basis des Gitters L. Ein Gitter ist also eine Menge aller Punkte, die durch ganzzahlige Linearkombinationen der Basisvektoren erreicht werden können. Man sagt L wird von $b_1, \ldots, b_n$ aufgespannt. n nennt man auch den Rang oder die Dimension des Gitters. Ist $d = n$, so sagt man das Gitter hat vollen Rang oder das Gitter ist volldimensional. Im Folgenden werden Basen auch in Matrixschreibweise verwendet, wobei die *Zeilen* der Matrix den Basisvektoren entsprechen.

Beispiel 3.1. *Abbildung 3.1 zeigt ein zweidimensionales Gitter L. Eine Basis dieses Gitters ist dargestellt durch die fett gedruckten Pfeile.*

$$B_1 = \begin{bmatrix} -b_1- \\ -b_2- \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ -2 & -1 \end{bmatrix}$$

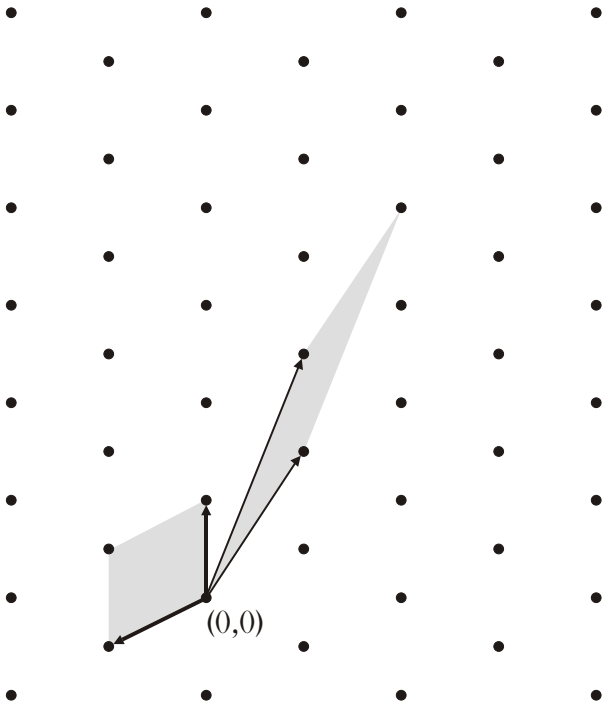

Abbildung 3.1: Zweidimensionales Gitter

Eine weitere Basis B_2 des gleichen Gitters L ist mit dünneren Pfeilen eingezeichnet:

$$B_2 = \begin{bmatrix} 2 & 3 \\ 2 & 5 \end{bmatrix}$$

Die grau hinterlegten Flächen kennzeichnen die Parallelogramme, die von den jeweiligen Basen aufgespannt werden. Man beachte, dass die Fläche dieser Parallelogramme identisch ist. Dieses Beispiel soll zwei Dinge verdeutlichen:

1. Gitter können mehrere Basen haben, es lässt sich sogar zeigen, dass jedes nicht triviale Gitter unendlich viele Basen hat.

2. Das n-dimensionale Volumen des Parallelogramms, welches von den Basisvektoren aufgespannt wird, ist unabhängig von der Basis für jedes Gitter eindeutig.

Diese Aussagen bleiben hier unbewiesen, mit dem Verweis auf [Cas71].

Definition 3.2. *Die Gitterdeterminante eines Gitters L mit Basis $b_1, \ldots, b_n$ ist definiert*

als Betrag der Determinante der Basismatrix:

$$\det(L) := |\det(b_1, \ldots, b_n)| \tag{3.1}$$

Die *Gitterdeterminante* ist eine *Invariante*, also unabhängig von der Basis der Matrix für jedes Gitter eindeutig. Anschaulich kann man sich vorstellen, dass sie dem oben genannten Volumen des Parallelepipeds entspricht. Für viele Anwendungen der Kryptographie und Kryptoanalyse[1] ist es wünschenswert Gitterbasen zu finden, die kurze Basisvektoren enthalten. Mit diesem Problem befassen sich die verschiedenen Verfahren der *Gitterbasenreduktion*.

3.1.3 Der LLL-Algorithmus

In diesem Abschnitt wird der so genannte LLL-Algorithmus von Lenstra, Lenstra und Lovàsz vorgestellt. Eine ausführlichere Darstellung findet der Leser in [LLL82], woraus die folgenden Ergebnisse übernommen wurden.

Wir schreiben $b_1^*, \ldots, b_n^* \in \mathbb{R}^d$ für die Vektoren, die das Gram-Schmidt'sche Orthogonalisierungsverfahren aus den Vektoren $b_1, \ldots, b_n \in \mathbb{R}^d$ erzeugt. Dabei werden die b_i^* folgendermaßen rekursiv definiert:

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^* \tag{3.2}$$

$$\mu_{i,j} = \frac{\left\langle b_i, b_j^* \right\rangle}{\left\langle b_j^*, b_j^* \right\rangle} \tag{3.3}$$

Dieses Verfahren kann nicht direkt zur Gitterbasenreduktion verwendet werden, da die entstehenden Vektoren im Allgemeinen keine Basis des Gitters mehr sind. Allerdings spielt es im Zusammenhang mit dem LLL-Algorithmus eine wichtige Rolle.

Definition 3.3. *Eine Basis $b_1, \ldots, b_n$ wird LLL-reduziert genannt, wenn für die durch das Gram-Schmidt'sche Verfahren entstehenden Vektoren $b_1^*, \ldots, b_n^*$ folgende Bedingungen gelten:*

$$|\mu_{i,j}| \leq 1/2 \quad \forall 1 \leq j < i \leq n \tag{3.4}$$

$$\left\| b_i^* + \mu_{i,i-1} b_{i-1}^* \right\|^2 \geq (3/4) \left\| b_{i-1}^* \right\|^2 \tag{3.5}$$

[1] Phong Q. Nguyen und Jacques Stern fassen in [NS01] die „zwei Gesichter der Gitter in der Kryptologie“ zusammen.

Man sagt, ein Vektor b_k ist längenreduziert, wenn für $1 \leq j < k$ gilt: $\mu_{k,j} \leq 1/2$. Es folgt damit $\|b_k\|^2 \leq 1/4 \sum_{j=1}^{k-1} \|b_j^*\|^2 + \|b_k^*\|^2$.

Satz 3.1 (Lenstra, Lenstra, Lovàsz). *Sei $b_1, \ldots, b_n$ die LLL-reduzierte Basis des Gitters L aus $\mathbb{R}^n$, so gilt:*

$$\|b_j\|^2 \leq 2^{i-1} \|b_i^*\|^2 \quad \forall 1 \leq j < i \leq n \tag{3.6}$$

$$\det(L) \leq \prod_{i=1}^{n} \|b_i\| \leq 2^{\frac{n(n-1)}{4}} \det(L) \tag{3.7}$$

$$\|b_1\| \leq 2^{\frac{(n-1)}{4}} \det(L)^{1/n} \tag{3.8}$$

Beweis siehe [LLL82][S. 517].

Ungleichung 3.8 erlaubt es, die Norm des ersten Vektors der reduzierten Basis nach oben abzuschätzen. Für einige der in dieser Arbeit vorgestellten Angriffe wird zusätzlich eine ähnliche Schranke für andere Vektoren benötigt. Dazu verwenden wir den folgenden Satz:

Satz 3.2 (Lenstra, Lenstra, Lovàsz). *Sei $b_1, \ldots, b_n$ die LLL-reduzierte Basis des Gitters L aus $\mathbb{R}^n$, so gilt:*

$$\|b_j\| \leq 2^{\frac{n(n-1)}{4(n-j+1)}} \det(L)^{1/n-j+1} \ \forall j = 1, \ldots, n \tag{3.9}$$

Beweis siehe [May03][S. 23].

Der *LLL-Algorithmus* stellt nun ein Verfahren dar, mit dem aus jeder Gitterbasis eine LLL-reduzierte Basis erstellt werden kann. Hier soll nun der Algorithmus soweit beschrieben werden, wie es dem Verständnis dienlich ist. In [LLL82, Kap. 1] findet man dazu eine ausführliche und gut verständliche Darstellung.

Abbildung3.2 zeigt den LLL-Algorithmus in vereinfachter Form in Pseudocode. Zunächst werden alle $\mu_{i,j} (1 \leq j < i \leq n)$ und $b_i^* (1 \leq i \leq n)$ berechnet. Der Algorithmus ändert nun die nicht reduzierte Basis $b_1, \ldots, b_n$ in eine LLL-reduzierte um, indem er für die Teilmenge $b_1, \ldots, b_k$ mit $k \leq n$ die Bedingungen aus Definition 3.3 herstellt. Dazu wird k initial auf 2 gesetzt, da für $k = 1$ die Bedingungen 3.4 und 3.5 bereits gelten. Jetzt wird in jedem Schleifendurchlauf in Zeile 5 der aktuelle Vektor b_k so längenreduziert, dass Bedingung 3.4 gilt und die entsprechenden Gram-Schmidt-Koeffizienten $\mu_{i,j}$ werden angepasst. Stimmt dann noch Bedingung 3.5, so wird k um eins erhöht, da die Vektoren $b_1, \ldots, b_k$ jetzt reduziert sind. Stimmt die Bedingung nicht, wird b_k mit dem letzten Vektor getauscht und k um eins vermindert. Dabei müssen b_k^* und b_{k-1}^* sowie alle zugehörigen Gram-Schmidt-Koeffizienten μ aktualisiert werden.

```
LLL(b_1, ..., b_n)
 1  berechne alle b_i^* und μ_{i,j}
 2  n ← 2
 3  while k ≤ n
 4  do
 5      Längenreduktion von  b_k
 6      aktualisiere μ_{k,j} für j < k
 7      if (Bed. 3.5 für i = k efüllt)
 8        then k ← k + 1
 9        else  SWAP(b_k, b_k − 1)
10              k ← MAX(k − 1, 2)
11  return b_1, ..., b_n
```

Abbildung 3.2: LLL-Algorithmus

Der Algorithmus nimmt also in jeder Iteration entweder einen Vektor hinzu oder tauscht den letzten reduzierten Vektor mit dem neuen und verliert dadurch einen reduzierten Vektor.

An dieser Stelle wurde noch nicht geklärt, wie die Basis im Einzelnen modifiziert wird, um die Bedingungen der Definition für LLL-reduzierte Basen zu erfüllen. Diese Erläuterung würde hier allerdings zu weit führen, der Interessierte sei auf [LLL82, S. 519ff] verwiesen. Dort wird auch gezeigt, dass der Algorithmus in jedem Falle terminiert. Zur Laufzeit gilt folgender Satz:

Satz 3.3 (Lenstra, Lenstra, Lovàsz). *Sei $L \subset \mathbb{Z}^n$ ein Gitter mit Basis $b_1, \ldots, b_n$ und sei $B \in \mathbb{R}, B \geq 2$ so, dass $\left\|b_i^2\right\| \leq B$. Dann ist die Anzahl der ganzzahligen arithmetischen Operationen, die von dem Algorithmus benötigt werden, in $O(n^4 \log B)$ und die Zahlen, mit denen diese Operationen durchgeführt werden, haben die Länge $O(n \log B)$.*

Unter Verwendung von klassischen arithmetischen Operationen ergeben sich daraus $O(n^6 log^3 B)$ Bitoperationen.

Praktische Algorithmen nutzen Gleitkommaarithmetik [SE94] und besitzen in der Praxis deutlich bessere Laufzeiten, als die in Satz 3.3 bewiesene [Sho].

3.1.4 Das Lösen univariater modularer Gleichungen

Ein Teil der in diesem Kapitel vorgestellten Angriffe führt das Problem geheime Informationen zu erhalten zurück auf das Problem die Nullstellen eines univariaten modularen Polynoms zu finden. In diesem Abschnitt werden zunächst wichtige Begriffe geklärt und

anschließend wird das Verfahren von Nick Howgrave-Graham [HG97] vorgestellt. Das Verfahren basiert auf dem von Don Coppersmith [Cop96b] präsentierten und hat eine vergleichbare Laufzeit. Allerdings ist die Darstellung von Howgrave-Graham verständlicher, sodass auch Coppersmith in [Cop01] diese Darstellung wählt.

Wir werden uns im Folgenden mit Polynomen der Form

$$p(x) = x^k + a_{k-1}x^{k-1} + \ldots + a_1 x + a_0 \mod N$$

befassten. Es handelt sich also um univariate modulare Polynome. Wir nehmen an, dass N keine Primzahl, jedoch schwer zu faktorisieren ist. Eine Bedingung für dieses Verfahren ist, dass das Polynom monisch ist, also 1 als Leitkoeffizienten hat. Es stellt jedoch kein Problem dar, diese Eigenschaft herzustellen. Man multipliziert dazu einfach alle Koeffizienten mit dem multiplikativen Inversen des Leitkoeffizienten. Dieses kann zum Beispiel mit dem erweiterten Euklidischen Algorithmus leicht gefunden werden, da ein möglicher Leitkoeffizient in der Regel teilerfremd zu N sein wird[2].

In folgenden werden Polynome häufig auch in Vektorschreibweise verwendet. Dabei stellen die Koeffizienten der entsprechenden Potenzen der Unbekannten die Einträge der Vektoren dar. Die Potenz der Unbekannten bestimmt den Index des Eintrages im Vektor für den zugehörigen Koeffizienten.

Das Problem, mit dem sich dieser Abschnitt befasst, ist es, die Nullstellen eines solchen Polynoms in $\mathbb{R}_N$ zu finden. Das ist nicht so leicht möglich, wie bei Polynomen aus $\mathbb{R}$.

Beispiel 3.2. *Das Polynom $p(x) = 4x - 4$ hat in $\mathbb{R}$ die einzige Nullstelle 1. Diese Nullstelle würde auch durch das Newtonsche Iterationsverfahren angenähert werden können. Betrachtet man das Polynom jedoch modulo 17, so sind alle x_i eine Nullstelle $p(x_i) \equiv 0 \mod 17$, für die $p(x_i)$ ein ganzzahliges Vielfaches von 17 ergibt. Dazu zählen neben der 1 zum Beispiel auch $\frac{55}{4}$. Klassische Verfahren zur Nullstellensuche versagen hier.*

Nun kann man fragen, welche Nullstellen x_0 mit $p(x_0) \equiv 0 \mod N$ auch Nullstellen von $p(x)$ in $\mathbb{R}_N$ sind. Im Beispiel gab es mit $x_0 = 1$ eine Nullstelle, die sowohl für $p(x)$ als auch für $p(x) \mod 17$ galt. Dazu kann man sich überlegen, dass alle diejenigen Nullstellen auch in $\mathbb{R}$ gelten, für die alle Berechnungen Ergebnisse liefern, die dem Betrag nach kleiner als der Modulus sind. Dann ist der Modulusoperator wirkungslos und die Berechnungen liefern in $\mathbb{R}$ das selbe Ergebnis. Dazu wir hier der Begriff der gewichteten Norm von Polynomen eingeführt.

Definition 3.4. *Sei $h(x) \in \mathbb{R}[x]$ ein Polynom vom Grad k und $X \in \mathbb{R}, X > 0$ gegeben,*

[2] Wäre dies nicht der Fall, so hätte man einen nicht trivialen Faktor von N gefunden.

dann ist die gewichtete Norm *von* $h(x)$ *definiert als* $\|h(xX)\| = \sum_{i=0}^{k} \left|X^i a_i\right|$

Der folgende Satz von Howgrave-Graham nutzt die gewichtete Norm von Polynomen, um zu zeigen, welche Nullstellen sowohl für $p(x) \mod N$ als auch für $p(x)$ gelten.

Satz 3.4 (Howgrave-Graham). *Sei* $h(x) \in \mathbb{R}[x]$ *ein Polynom vom Grad* n *und sei* $X \in \mathbb{R}$ *gegeben. Gibt es ein* $|x_0| < X$ *mit*

$$h(x_0) \equiv 0 \mod N \text{und}$$

$$\|h(xX)\| < N/\sqrt{n},$$

so ist $h(x_0) = 0$ *in ganz* $\mathbb{R}$.

Beweis 3.1. *Es gilt:*

$$|h(x_0)| = \left|\sum a_i x_0^i\right| = \left|\sum a_i X^i \left(\frac{x_0}{X}\right)^i\right|$$

$$\leq \sum \left|a_i X^i \left(\frac{x_0}{X}\right)^i\right| \leq \sum \left|a_i X^i\right|$$

$$\leq \sqrt{n}\, \|h(xX)\| < N$$

Aus $|h(x_0)| < N$ *und* $h(x_0) \equiv 0 \mod N$ *folgt jetzt* $h(x_0) = 0$.

□

Wenn wir also die Nullstelle x_0 hinreichend beschränken können und das Polynom $p(x) \mod N$ eine kleine gewichtete Norm hat, ist auch $p(x_0) = 0$. Die Idee, die diesem Angriff zu Grunde liegt ist also, ausgehen von einem Polynom $p \in \mathbb{Z}_N$ ein Polynom $b(x)$ zu finden, dass eine kleine Norm hat. Dafür werden zu $p(x) \mod N$ weitere Polynome $q_i(x)$ aufgestellt, die alle notwendigerweise die gleiche Nullstelle x_0 modulo N besitzen. Wenn $q_i(x_0) \equiv 0 \mod N$ für alle $i = 1, \ldots, n$ gilt, dann gilt das auch für alle Linearkombinationen $\sum_{i=1}^{n} c_i q_i(x_0)$ mit $c_i \in \mathbb{Z}$. Gesucht wird also eine Linearkombination der $q_i(x)$ mit kleiner gewichteter Norm.

Dazu wird der LLL-Algorithmus verwendet. Zunächst werden die Koeffizienten der Polynome $q_i(xX)$ als Vektoren dargestellt, die die Basis des Gitters L bilden. Das Gewicht X wird passend gewählt, so dass die LLL-Reduktion kurze Vektoren bestimmt. Im Verlaufe der Arbeit wird weiter darauf eingegangen. Jeder Vektor in diesem Gitter repräsentiert nun ein Polynom, das die Nullstelle x_0 modulo N besitzt. Der LLL-Algorithmus reduziert die Basis nun und man erhält eine kurzen ersten Vektor. Daraus erstellt man wieder ein Polynom $h(x)$, für das $h(x_0) = 0$ in $\mathbb{R}$ gilt.

Beispiel 3.3. *Abbildung 3.3 zeigt das Gitter, das von den Polynomen* $q_1(x) = 3x + 1$ *und* $q_2(x) = 4x - 1$ *aufgespannt wird. Jedes Polynom, das aus einem beliebigen Gittervektor gebildet wird hat genau wie* $q_1(x)$ *und* $q_2(x)$ *in* $x_0 = 2$ *eine Nullstelle modulo* 7*. Wir wählen* $X = 3$ *als Schranke für* x_0*. Wenn wir ein Polynom* $h(x)$ *finden können, dessen gewichtete Norm* $\|h(xX)\|$ *kleiner als* 7 *ist, dann hat dieses eine Nullstelle in* x_0 *in* $\mathbb{R}$*. So ein Polynom stellt das eingezeichnete* $h(x) = x - 2$ *dar, seine gewichtete Norm beträgt* 5*. Tatsächlich werden bei diesem Verfahren die Koeffizientenvektoren von* $q_1(xX)$ *und* $q_2(xX)$ *verwendet um das Gitter aufzuspannen, was hier aus Gründen der Übersichtlichkeit nicht getan wurde.*

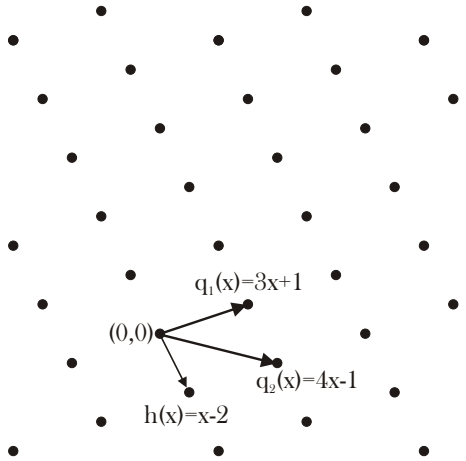

Abbildung 3.3: Gitter zu Beispiel 3.3

Hier wird nun das Verfahren für beliebige $p(x)$ vorgeführt. Wir werden später sehen, dass es auch Basen gibt, die den Eigenschaften von $p(x)$ besser Rechnung tragen. Zunächst muss ein $X > 0$ gewählt werden, sodass $|x_0| < X$ gilt. Dann wählt man ein $h \geq 2 \in \mathbb{Z}$, dessen Funktion später genauer erklärt wird. Nun werden die Polynome

$$q_{u,v}(x) = N^{h-1-v} x^u (p(x))^v, \qquad u > 0, v > 0 \in \mathbb{Z}$$

aufgestellt. Da alle $q_{u,v}(x)$ den Faktor $p(x)$ enthalten, haben sie alle in x_0 eine Nullstelle modulo N. Jetzt wird eine $(hk) \times (hk)$ Matrix M aufgestellt, deren Einträge $m_{i,j}$ definiert werden als $m_{i,j} = e_{i,j} X^{j-1}$, wobei $e_{i,j}$ der Koeffizient von x^{j-1} von $q_{u,v}$ mit $v = \lfloor (i-1)/k \rfloor$ und $u = (i-1) - kv$ ist. Es wird also eine Matrix aus den Koeffizienten von $q_{u,v}(xX)$ aufgestellt.

Der LLL-Algorithmus liefert uns nun eine Basis mit einem kurzen ersten Vektor b_1, der die Koeffizienten des gesuchten Polynoms $h(xX)$ repräsentiert. Aus Satz 3.4 wissen wir, dass aus $h(x_0) = 0 \mod N$ und $\|h(xX)\| < N/\sqrt{hk}$ folgt, dass $h(x_0) = 0$ ist. Um nun $\|h(xX)\| < N/\sqrt{hk}$ sicherzustellen, brauchen wir eine obere Grenze für $\|h(xX)\| = \|b_1\|$. Diese liefert uns Ungleichung 3.8:

$$\|b_1\| \leq 2^{\frac{(hk-1)}{4}} \det(L)^{1/hk}$$

Da die entstehende Matrix M eine untere Dreiecksmatrix ist, lässt sich ihre Determinante sehr einfach als Produkt Einträge der Diagonalen berechnen. Auf der Diagonalen stehen immer die Leitkoeffizienten von $q_{u,v}(xX)$. Da sich diese durch Multiplikation des Leitkoeffizienten von $p(x)$ mit $X^{j-1}N^{h-1-v}$ errechnen und $p(x)$ monisch ist, erhält man als Determinante:

$$\det(L) = X^{hk(hk-1)/2} N^{hk(h-1)/2}$$

Daraus ergibt sich

$$\|h(xX)\| < 2^{\frac{(hk-1)}{4}} X^{(hk-1)/2} N^{(h-1)/2}$$

Damit nun $\|h(xX)\| < N/\sqrt{hk}$ gilt, ergibt sich X maximal zu

$$X < 2^{-\frac{1}{2}} N^{\frac{h-1}{hk-1}} (hk)^{-\frac{1}{hk-1}}$$

Man sieht also, dass X für $h \to \infty$ gegen $2^{-\frac{1}{2}} N^{\frac{1}{k}}$ strebt.

Beispiel 3.4. *Mit $p(x) = x^2 + a_1x^1 + a_0$ und $h = 3$ erhält man nach diesem Verfahren die folgende $(hk) \times (hk) = 6 \times 6$ Matrix:*

$$\begin{bmatrix} N^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & XN^2 & 0 & 0 & 0 & 0 \\ a_0N & a_1XN & X^2N & 0 & 0 & 0 \\ 0 & a_0XN & a_1X^2N & X^3N & 0 & 0 \\ a_0^2 & 2a_1a_0X & (2a_0 + a1)X^2 & 2a_1X^3 & X^4 & 0 \\ 0 & a_0^2X & 2a_1a_0X^2 & (2a_0 + a1)X^3 & 2a_1X^4 & X^5 \end{bmatrix}$$

Die Matrix hat die Determinante $X^{hk(hk-1)/2} N^{hk(h-1)/2} = X^{15}N^6$.

Die Laufzeit dieses Verfahrens liegt nach [Sch99, S. 47] in

$$O(h^9 k^6 \log^3 N),$$

wenn N gegenüber k groß ist, wovon wir im Folgenden ausgehen werden.

3.1.5 Erweiterung auf den bivariaten Fall

In diesem Abschnitt wird die Erweiterung des Verfahrens auf Gleichungen in zwei Unbekannten vorgestellt. Dabei handelt es sich nicht um das von Coppersmith in [Cop96a] gezeigte Verfahren, welches Gleichungen in $\mathbb{Z}$ behandelt. Vielmehr geht es hier um die Verallgemeinerung des in Abschnitt 3.1.4 vorgestellten modularen Verfahrens für den bivariaten Fall. Vorgreifend sei jedoch erwähnt, dass es sich hier um ein heuristisches Verfahren handelt, wie später gezeigt wird.

Zunächst wird Satz 3.4 generalisiert und auf den multivariaten Fall erweitert.

Satz 3.5. *Sei $h(x_1, \ldots, x_k)$ ein Polynom vom Grad n und $X_1, \ldots, X_k \in \mathbb{R}$ gegeben. Gibt es $|x_i| < X_i$ für $i = 1, \ldots, k$ mit*

$$h(x_i, \ldots, x_k) \equiv 0 \mod N \text{ und}$$

$$\|h(x_1 X_1, \ldots, x_k X_k)\| < N/\sqrt{n},$$

so ist $h(x_1, \ldots, x_k) = 0$ in ganz $\mathbb{R}$.

Da der Beweis genau wie der zu Satz 3.4 verläuft, wird er hier nicht geführt.

Gesucht werden die Nullstellen $(x_0, y_0) \in \mathbb{R} \times \mathbb{R}$ des Polynoms $p(x, y) \mod N$ zu gegebenem $p(x, y)$ und N. Das Vorgehen verläuft sehr ähnlich zu dem aus Abschnitt 3.1.4. Zunächst definiert man die Polynome

$$q_{i,j,k}(x, y) := x^i y^j (p(x, y))^k N^{m-k}. \tag{3.10}$$

Es gilt $g_{i,j,k}(x_0, y_0) = 0 \mod N$ für alle $i, j \geq 0, k \leq m$. Nachdem die Schranken X und Y aufgestellt wurden, bildet man aus den Koeffizienten von $g_{i,j,k}(xX, yY)$ eine Gitterbasis. Die Berechnung von i, j und k ist abhängig von $p(x, y)$ und wird jeweils bei der Anwendung des Verfahrens erläutert. Wie im univariaten Verfahren wird auch hier das Gitter LLL-reduziert.

Da wir es hier jedoch mit Gleichungen in mehreren Unbekannten zu tun haben, gilt im Allgemeinen, dass eine Gleichung nicht ausreicht, um die gesuchte Nullstelle zu finden. Daher werden die ersten beiden Vektoren der reduzierten Basis verwendet, um daraus die Polynome $h_1(x, y)$ und $h_2(x, y)$ aufzustellen. Um die Norm des zweiten Polynoms

abschätzen zu können, verwenden wir Gleichung 3.9. Zusammen mit Satz 3.5 ergibt sich:

$$\det(L) < 2^{-\frac{n(n-1)}{4}} N^{n-1} n^{-\frac{n-1}{2}} \Rightarrow \|h_2(xX, yY)\| < \frac{N}{\sqrt{n}}$$

Man erhält also bei geeigneter Gitterdeterminante zwei Polynome $h_1(x, y), h_2(x, y) \in \mathbb{R}[x, y]$ mit $h_1(x_0, y_0) = 0 \wedge h_2(x_0, y_0) = 0$. Von diesen Polynomen ist bekannt, dass sie linear unabhängig sind, allerdings sind sie nicht notwendigerweise algebraisch unabhängig. Diese Eigenschaft ist jedoch notwendig, um die gesuchte Nullstelle (x_0, y_0) zu finden. Daher ist das Verfahren, wie oben bereits erwähnt, *heuristisch*. Es ist nicht sichergestellt, dass die gefundenen Polynome diese Eigenschaft erfüllen und es lassen sich Beispiele finden, in denen die Heuristik versagt. So zeigten Blömer und May in [BM01], dass für einen Sonderfall des von Boneh und Durfee in [BD00] vorgestellten Verfahrens die Heuristik immer versagt.

Sollten die gefundenen Polynome algebraisch unabhängig sein, können die Nullstellen leicht berechnet werden. Dazu wird in der Computeralgebra die so genannte Resultante berechnet. Für eine gute Einführung zu Resultanten siehe [GCL92]. Im Folgenden werden nur die relevanten Eigenschaften dargestellt. Zu den Polynomen $p(x_1, \ldots, x_m)$ und $q(x_1, \ldots, x_m)$ in m Veränderlichen ist die Resultante bezüglich x_i ein Polynom in $m - 1$ Veränderlichen, das genau dort eine Nullstelle besitzt, wo auch p und q eine gemeinsame Nullstelle besitzen. Man kann damit also aus zwei Polynomen in zwei Unbekannten $p(x, y)$ und $q(x, y)$ ein Polynom $r(x) = \mathrm{res}_y(p, q)$ in einer Unbekannten erstellen. Zu jeder Nullstelle x_0 dieses Polynoms sucht man dann die Nullstellen von $p(x_0, y)$ oder $q(x_0, y)$ und findet so alle gemeinsamen Nullstellen von p und q. Um die Resultante zu berechnen kann zum Beispiel die Determinante der so genannten Sylvestermatrix berechnet werden, aber es gibt schnellere Verfahren, wie zum Beispiel den Subresultanten-Algorithmus aus [Coh96]. Sind die Polynome nicht algebraisch unabhängig, so ergibt sich für die Resultante immer das Nullpolynom.

3.1.6 Übergang von den Grundlagen

Die meisten in diesem Kapitel vorgestellten Angriffe gehen nach einem ähnlichen Schema vor, das hier kurz zusammenfassend dargestellt werden soll. In diesem Abschnitt wird auch das zuvor gezeigte Verfahren von Coppersmith noch einmal kurz zusammengefasst.

Ausgangspunkt für die gezeigten Angriffe sind die elementaren Funktionen des RSA-Kryptosystems. Daraus wird zunächst ein Polynom p aus $\mathbb{Z}_N[x]$ aufgestellt, das eine Nullstelle $x_0 \in \mathbb{Z}_N$ besitzt. Diese Nullstelle stellt die gesuchten geheimen Informationen dar, wie z.B. der unbekannte Teil einer Nachricht oder ein Teil der Faktoren p oder q. Es

wird also ein $x_0 \in \mathbb{Z}_N$ gesucht mit $p(x_0) \equiv 0 \mod N$, was nicht trivial ist, wie bereits erläutert wurde. Daher bedienen sich die Angriffe des Verfahrens von Coppersmith:

- Zunächst wird ein $X \in \mathbb{R}$ passend gewählt.
- Dann wird aus dem Polynom $p(x)$ eine Menge weiterer Polynome $q_i(x),\ i = 1, \ldots, k$ aufgestellt, die alle ebenfalls in x_0 eine Nullstelle modulo N besitzen.
- Aus diesen $q_i(x)$ werden die zugehörigen $q_i(xX)$ gebildet. Die Koeffizienten der Polynome $q_i(xX)$ bilden die Zeilenvektoren der Basis des Gitters

$$L(q_1(xX), \ldots, q_k(xX)).$$

- Dabei muss darauf geachtet werden, dass die Determinate des Gitters L nicht zu groß wird.
- L wird nun LLL-reduziert und man erhält einen kurzen ersten Vektor b.
- Ist dieser Vektor b klein genug, gilt also $\|b(xX)\| \leq \frac{N}{\sqrt{n}}$ und existiert x_0 mit $p(x_0) \equiv 0 \mod N$ und $x_0 < X$, dann gilt auch $b(x_0) = 0$.
- Es werden nun Nullstellen von $b(x)$ bestimmt, unter ihnen ist das gesuchte x_0.

Einige der Angriffe nutzen auch multivariate Varianten des hier erläuterten Verfahrens, generell ist der Ablauf jedoch immer ähnlich. Allerdings stützen sie sich, bis auf eine Ausnahme, auf die Heuristik, dass die gefundenen kurzen Vektoren algebraisch unabhängig sind. Daher können viele Angriffe nicht garantieren, eine Lösung zu finden.

3.2 Angriffe auf kleine öffentliche Exponenten e

Dieser Abschnitt befasst sich mit dem Angriff auf Nachrichten m, die mit sehr kleinem öffentlichen Exponenten e verschlüsselt wurden. Da e frei gewählt werden kann, ist man versucht, einen kleinen Wert zu verwenden, um die Verschlüsselung zu beschleunigen. Hier soll untersucht werden, welche Risiken für kleine öffentliche Exponenten bestehen.

Erste Angriffe auf RSA mit kleinem e zeigte Håstad [Hås88] schon 1988. Dabei wird eine Nachricht vollständig entschlüsselt, wenn viele verschiedene Chiffretexte des selben Klartextes vorliegen. Abschnitt 3.2.1 zeigt, wie sich die Originalergebnisse von Håstad verbessern, wenn die Verfahren zum Lösen univariater modularer Gleichungen nach Coppersmith verwendet werden.

In Abschnitt 3.2.2 wird gezeigt, dass ein unbekannter Teil der Nachricht herausgefunden werden kann, falls der überwiegende Teil bekannt ist.

Einen weiteren Angriff stellt Coppersmith 1997 in [Cop97] vor. Er zeigt, dass einfaches zufälliges Padding der Nachricht vor dem Verschlüsseln ein Sicherheitsrisiko darstellt. In 3.2.3 werden dieser Angriff und einfache Gegenmaßnahmen erläutert.

3.2.1 Broadcasted Messages

Das von Håstad in [Hås88] vorgestellte Verfahren findet den Klartext m einer RSA-verschlüsselten Nachricht, wenn diese an viele Parteien mit unterschiedlichen Modulen aber gleichen öffentlichen Exponenten versendet wird. Dem Angriff liegt die folgende Idee zugrunde:

Seien $C_i \equiv m^e \mod N_i, i = 1, \ldots, k$ verschiedene Chiffretexte C_i der gleichen Nachricht m mit dem gleichen öffentlichen Exponenten e jedoch unterschiedlichen Modulen N_i. Durch Verwendung des Chinesischen Restsatzes ist es möglich, ein $C < \prod_{i=1}^{k} N_i$ zu berechnen, sodass $C \equiv C_i \mod N_i$ für alle $i = 1, \ldots, k$ gilt. Dieses C erfüllt dann aber auch die Gleichung $C \equiv m^e \mod \prod_{i=1}^{k} N_i$. Da die Nachricht m kleiner ist als jedes der N_i, muss m^e kleiner sein als $\prod_{i=1}^{k} N_i$ sobald $k \geq e$. Ist dies der Fall, kann man m durch $m = C^{\frac{1}{e}}$ berechnen.

Es reichen also bereits e verschiedene Paare von C_i und N_i aus, um die Nachricht zu entschlüsseln. Als Schlussfolgerung ergibt sich, dass man niemals die gleiche Nachricht an viele Parteien versenden sollte, die den gleichen öffentlichen Exponenten e haben. Man ist nun geneigt, jede Nachricht vor dem Verschlüsseln zu modifizieren, etwa indem man eine fortlaufende Nummer anhängt, oder eine andere einfache Funktion anwendet. Håstad konnte allerdings das vorgestellte Verfahren mittels Gitterreduktion so verallgemeinern, dass es selbst dann noch funktioniert, wenn die Nachrichten derart modifiziert wurden. Allerdings verwendete er eine andere Technik als die in Kapitel 3.1.4 vorgestellten, weshalb die Originalergebnisse nochmals verbessert werden konnten.

Shimizu [Shi96] und Bleichenbacher [Ble97] zeigen laut [CNS99] erstmal die Verbesserung des Verfahrens nach Håstad durch Verwendung von Coppersmiths Technik zur Lösung univariater modularer Polynome. Hier wird dieses Verfahren skizziert und die Grenzen für k von Håstad und der neuen Variante gezeigt.

Angenommen der Absender der Nachricht wendet vor dem Verschlüsseln das Polynom $f_i(x)$ auf die Nachricht m an und verschlüsselt die so erhaltenen Nachrichten m_i. Dazu berechnet er die Werte $C_i \equiv (m_i)^{e_i} \mod N_i$ für jeden Empfänger $i = 1, \ldots, k$. Kennt ein Angreifer e_i, N_i und $f_i(x)$ und die zugehörigen C_i, so kann er die Nachricht m effizient

berechnen, sobald k groß genug ist. Dazu stellt er die Polynome

$$g_i(x) = f_i^{e_i}(x) - C_i \mod N_i$$

mit

$$g_i(m) \equiv 0 \mod N_i$$

auf. Er erhält also k modulare univariate Polynome für die $g_i(m) \equiv 0 \mod N_i$ gilt. Mittels des Chinesischen Restsatzes berechnet er nun Koeffizienten T_i, für die gilt:$T_i \equiv 0 \mod N_i$ und $T_i \equiv 1 \mod N_j$ für alle $i \neq j$. Nun wird ein neues Polynom $h(x)$ gebildet mit $h(x) = \sum T_i g_i(x)$. Für das so erhaltene $h(x)$ gilt nun $h(m) \equiv 0 \mod \prod N_i$. Der Grad d dieses Polynoms ist $d = \max(e_i) \cdot \max(\deg f_i)$. Mit dem Verfahren für univariate modulare Polynome aus Abschnitt 3.1.4 können wir die Nullstelle m berechnen, wenn $m < (\prod N_i)^{\frac{1}{d}}$ gilt. Da bekannt ist, dass $m < \min(N_i) < (\prod N_i)^{\frac{1}{k}}$ gilt, funktioniert dieses Verfahren sobald $k \geq d$ ist.

Beispiel 3.5. *Es werde eine Nachricht an k Empfänger gesendet. Der Inhalt besteht aus dem Klartext m, der mit der Zahl $i = 1, \ldots, k$ addiert wird. Die öffentlichen Exponenten e seien alle gegeben durch $e = 3$. Dies stellt eine sehr einfache Form des Padding dar und es genügen theoretisch $k = 3$ Nachrichten, um den Klartext zu gewinnen. Allerdings müsste dafür der Parameter h sehr groß werden.*

Das ursprüngliche Ergebnis von Håstad benötigte $k \geq O(d^2)$ statt $k \geq d$ Nachrichten.

In [CNS99] werden praktische Laufzeiten angegeben, die erkennen lassen, dass der Angriff tatsächlich in vertretbarer Zeit durchgeführt werden kann, wenn $e = 3$ ist. Für höhere Werte von e wird das Verfahren deutlich uninteressanter, da die Dimension des Gitters von e abhängt und auch mehr Nachrichten benötigt werden. Daher ist auch der populäre Exponent $e = 2^{16} + 1$ nicht bedroht. Die theoretische Grenze von $k = d$ wird auch meist nicht erreicht, weil dazu h ebenfalls sehr groß werden müsste. Allerdings reicht in vielen Fällen schon $k = d + 1$ aus, um den Klartext zu erhalten.

3.2.2 Stereotype Nachrichten

Coppersmith selbst stellte in [Cop96b] eine Anwendung seines Verfahrens vor, mit der es möglich war, den Klartext einer Nachricht zu entschlüsseln, wenn der größte Teil davon bekannt ist. Um diesen Angriff zu motivieren, sei vorweg ein kurzes Beispiel gegeben.

Beispiel 3.6. *Alice schickt Bob in regelmäßigen Intervallen ein neues Passwort zu. Dazu generiert sie die Nachricht „Das Passwort lautet: xxx“ und verschlüsselt sie mit Bobs*

öffentlichen Schlüssel. Ein Angreifer kann mit dem hier vorgestellten Verfahren den unbekannten Teil herausfinden, falls er den Rest der Nachricht und Bobs öffentlichen Schlüssel kennt.

Allerdings ist dieser Angriff nur anwendbar, falls der unbekannte Teil relativ kurz ist. Zunächst wird das Problem, den unbekannten Teil von m zu finden, umgeformt in eine Nullstellensuche. Dazu betrachtet man die Verschlüsselungsfunktion von RSA näher:

$$m^e = c \ mod \ N \Leftrightarrow m^e - c = 0 \ mod \ N \tag{3.11}$$

Seien B_1 der bekannte Teil, der vor dem unbekannten Teil x steht und B_2 der bekannte Teil, der danach steht. Sei weiter b die Basis in der m repräsentiert wird und l_x, l_{B_2} die Anzahl der Ziffern bzw. Zeichen von x und B_2 in dieser Darstellung. Dann kann man m schreiben als:

$$m = B_1 b^{l_x + l_{B_2}} + x b^{l_{B_2}} + B_2$$

Zusammen mit Gleichung 3.11 erhält man:

$$(B_1 b^{l_x + l_{B_2}} + b^{l_{B_2}} x + B_2)^e - c = 0 \ mod \ N$$

In Abschnitt 3.1.4 wurde gezeigt, dass eine Nullstelle zu dem modularen univariaten Polynom $p(x_0) = 0 \ mod \ N$ gefunden werden kann, wenn x_0 kleiner ist als $2^{-\frac{1}{2}} N^{\frac{1}{k}}$. Dabei bezeichnet k den Grad des Polynoms. In diesem Fall ist $k = e$, man erhält also eine Lösung für x_0 falls $x_0 \leq 2^{-\frac{1}{2}} N^{\frac{1}{e}}$. Praktisch wird die Grenze jedoch darunter liegen, da sich der Parameter h, der hier gegen ∞ strebt sehr stark auf die Laufzeit des Verfahrens auswirkt. In Anhang A wird eine Implementierung des Angriffs vorgestellt und die praktischen Laufzeiten und Grenzen genauer betrachtet.

3.2.3 Short Random Pad

In [Cop97] zeigt Coppersmith einen weiteren Angriff auf RSA mit kleinem öffentlichen Exponenten. Dabei wird vorausgesetzt, dass die Nachricht vor dem Verschlüsseln mit einer zufälligen Folge von Bits aufgefüllt wird. Gelingt es dem Angreifer, in den Besitz von zwei verschiedenen verschlüsselten Texten der gleichen Nachricht M mit dem gleichen Schlüssel (e, N) zu kommen, so kann er die Nachricht entschlüsseln. Anders als bei dem Angriff aus Abschnitt 3.2.1 wird hier die Nachricht also zweimal mit dem selben Schlüssel verschlüsselt.

Hier wird gezeigt, wie das Verfahren funktioniert, wenn der Nachricht M das Padding angehängt wird. Wird das Padding vorangestellt, verläuft es sehr ähnlich. Im Folgenden

wird der Angriff für $e = 3$ vorgestellt. Er lässt sich mit großen Einschränkungen analog auch für größere e verwenden, dazu später mehr.

Sei M der Klartext der Nachricht und R und $R' = R + r$ zufällige Bitfolgen mit der Bitlänge k, die der Nachricht angehängt werden. Man erhält die beiden verschlüsselten Nachrichten c und c' aus:

$$c \equiv m^3 \equiv (2^k M + R)^3 \mod N$$

und

$$c' \equiv (m')^3 \equiv (2^k M + R')^3 \equiv (m + r)^3 \mod N.$$

Durch einfaches Umformen erhält man:

$$m^3 - c \equiv 0 \mod N \text{ und } (m + r)^3 - c' \equiv 0 \mod N.$$

Diese Gleichungen enthalten die beiden Unbekannten m und r. Um eine Gleichung in einer Unbekannten zu erhalten, bildet man die Resultante der beiden Polynome nach m:

$$\mathrm{Res}_m(m^3-c, (m+r)^3-c') = r^9+(3c-3c')r^6+(3c^2+21cc'+3(c')^2)r^3+(c-c')^3 \equiv 0 \mod N$$

Man erhält also wieder ein univariates modulares Polynom in r vom Grad 9. Wenn $|r| \leq N^{\frac{1}{9}}$ ist, kann es mittels des univariaten Verfahrens aus Abschnitt 3.1.4 berechnet werden. Anschließend wendet Coppersmith ein Verfahren von Franklin und Reiter aus [FR95] an, von dem hier nur die folgende Formel ohne Beweis übernommen wird:

$$m = \frac{r(c' + 2c - r^3)}{c' - c + 2r^3} = \frac{r(3m^3 + 3m^2 r + 3mr^2)}{3m^2 r + 3mr^2 + 3r^3} \mod N.$$

Dieses Verfahren lässt sich für jedes e verwenden, solange der zufällige Teil der Nachricht nicht mehr als $N^{\frac{1}{e^2}}$ ist.

Auch hier gibt [CNS99] reale Laufzeiten zu dem vorgestellten Verfahren für $e = 3$ an, die zeigen, dass dieser Angriff in vertretbarer Zeit durchgeführt werden kann. Bemerkenswert ist dabei allerdings auch, dass praktisch sogar längere r gefundenen werden konnten als $N^{\frac{1}{9}}$.

Coppersmith selbst nennt jedoch schon einige einfache Gegenmaßnahmen. So kann der Angriff abgewehrt werden, wenn das Padding nicht zufällig ist, sondern von der Nachricht selbst abhängt, indem man etwa den Hashwert der Nachricht verwendet. Auch ein größeres e macht den Angriff unwirksam, da bei $e = 7$ und $N = 1024$ beispielsweise bereits nur noch 21 Bit Padding der Nachricht toleriert würden und eine vollständige

Suche hier schneller wäre.

In Kapitel 4 werden die verschiedenen anderen Gegenmaßnahmen genauer betrachtet.

3.3 Kurze geheime Exponenten

Um die Entschlüsselung oder den Signiervorgang zu beschleunigen, ist es verlockend kurze geheime Schlüssel d zu verwenden. Das ist besonders interessant, wenn dieser Vorgang in einer Smartcard mit begrenzter Rechenleistung durchgeführt wird. Jedoch bereits 1990 zeigte Wiener [Wie90], dass kurze geheime Exponenten ein Sicherheitsrisiko darstellen. Er stellte ein Verfahren mit der Laufzeit $\log N$ vor, mit dem man d annähern konnte, wenn $d < \frac{1}{3}N^{\frac{1}{4}}$ ist.

1997 zeigten Verheul und van Tilborg [VT97] ein Verfahren, mit dem es möglich war, d zu finden, wenn $d < N^{\frac{1}{2}}$. Das Verfahren hat zwar exponentielle Laufzeit, sobald $d > N^{\frac{1}{4}}$ ist, läuft jedoch schneller, als eine vollständige Suche.

In [BD00] zeigten Boneh und Durfee erstmals Angriffe auf geheime Exponenten $d > N^{\frac{1}{4}}$ mittels Gitterbasenreduktion. Sie konnten die Anwendbarkeit ihres Verfahrens bis zu einer Größe von $N^{0,292}$ nachweisen. Dabei zeigten sie zunächst die Gültigkeit für $d < N^{0,284}$ und verbessern anschließend diese Grenze durch Modifikation der Gitterbasis.

In diesem Abschnitt wird das Verfahren von Blömer und May [BM01, May03] vorgestellt, welches auf dem von Boneh und Durfee basiert. Allerdings wird dabei eine andere Strategie zur Modifikation der Gitterbasis verwendet. Dadurch wird die Grenze für d zwar lediglich auf $N^{0,290}$ erweitert, aber durch die kleinere zu reduzierende Basis können praktisch höhere Werte erreicht werden.

Ausgangspunkt für diesen Angriff stellt die folgende Gleichung dar:

$$ed \equiv 1 \; mod \; \Phi(N)$$

Es existiert also ein $k \in \mathbb{Z}$ für das

$$ed + k\Phi(N) = 1 \tag{3.12}$$

über $\mathbb{Z}$ gilt. Dabei ist $\Phi(N)$ bekanntlich $(p-1)(q-1) = pq - p - q + 1 = N - (p+q) + 1$. Durch Einsetzen erhält man schließlich:

$$ed + k(N + 1 - (p + q)) = 1$$

Setzt man nun $s := -(p + q)$ und $A := N + 1$ und reduziert beide Seiten modulo e, so

ergibt sich:

$$k(A+s) \equiv 1 \ mod \ e$$

Jetzt muss noch die Größe von $|k|$ und $|s|$ abgeschätzt werden. Dazu werden einige Annahmen getroffen. Wir nehmen an, dass p und q zwischen $\sqrt{N}$ und $2\sqrt{N}$ liegen. Außerdem sei e in der gleichen Größenordnung wie N angesiedelt, das heißt wir können e ausdrücken als $e = N^{\alpha}$ mit $\alpha \approx 1$. d wird ebenfalls in Relation zu N angegeben, indem wir sagen $d < N^{\delta}$. Aus Gleichung 3.12 und der Tatsache, dass $\Phi(N) \leq \frac{2}{3}N$ ist, wissen wir, dass

$$|k| < \frac{de}{\Phi(N)} \leq \frac{3de}{2N} < \frac{3}{2}e^{1+\frac{\delta-1}{\alpha}}.$$

Außerdem erhalten wir für s die Ungleichung:

$$|s| < 3N^{\frac{1}{2}} = 3e^{\frac{1}{2\alpha}}$$

Wir setzen im Folgenden vereinfachend $\alpha = 1$ und ignorieren kleine konstante Faktoren. Dann stellen die gesuchten k und s die Nullstelle (x_0, y_0) des bivariaten modularen Polynoms $f(x, y) = x(A + y) - 1$ dar, wobei $A = N + 1$ ist.

$$f(x_0, y_0) = 0 \ mod \ e, \ mit \ |x_0| < e^{\delta} \ und \ |y_0| < e^{\frac{1}{2}}$$

Im Wesentlichen wird nun das in Abschnitt 3.1.5 vorgestellte Verfahren angewendet. Das heißt, es wird eine Gitterbasis aufgestellt und reduziert. Dann werden aus den Vektoren der reduzierten Basis Polynome aus $\mathbb{Z}[x, y]$ aufgestellt, die mittels Resultantenbildung gelöst werden können. Bei der Erzeugung der Basismatrix wird hier der Ansatz von Boneh und Durfee verfolgt. Anschließend wird jedoch der Weg von Blömer und May verwendet, um die Grenze für δ von $\delta \approx 0,284$ auf $\delta \approx 0,290$ zu erhöhen.

Zunächst werden die Polynome

$$g_{i,k}(x, y) := x^i(f(x, y))^k e^{m-k} \text{ und } h_{j,k}(x, y) = y^j(f(x, y))^k e^{m-k}$$

für $i, j, k \geq 0$ und $k < m$ definiert. Diese stellen eine Teilmenge der in Gleichung 3.10 definierten Polynome dar und haben eine Nullstelle in $(x_0, y_0) = (k, s)$ modulo e^m. Anschließend erzeugen Boneh und Durfee eine Gitterbasis aus den Koeffizienten von $g_{i,k}(xX, yY)$ und $h_{j,k}(xX, yY)$. Zur Bildung dieser Basis werden zwei Parameter m und t eingeführt, die später genauer erläutert werden. Die Einträge der Matrix werden definiert als die Koeffizienten von $g_{i,k}(xX, yY)$ für alle $k = 0, \ldots, m$ und $i = 0, \ldots, m - k$ und die Koeffizienten von $h_{j,k}(xX, yY)$ für alle $j = 0, \ldots, t$.

		X^0	X^1		X^2			Y^1			Y^2		
		1	x	xy	x^2	x^2y	x^2y^2	y	xy^2	x^2y^3	y^2	xy^3	x^2y^4
X_0	1	e^2											
	x		e^2X										
X_1	f	-	-	eXY									
	x^2				e^2X^2								
X_2	xf		-		-	eX^2Y							
	f^2	-	-	-	-	-	X^2Y^2						
	y							e^2Y					
Y_1	yf			-				-	eXY^2				
	yf^2			-		-	-	-	-	X^2Y^3			
	y^2										e^2Y^2		
Y_2	y^2f								-		-	eXY^3	
	y^2f^2						-		-	-	-	-	X^2Y^4

Abbildung 3.4: Gitterbasis für $m = 2$ und $t = 2$ nach Boneh/Durfee.

Abbildung 3.4 zeigt die nach dieser Vorschrift entstehende Gitterbasis zu gewählten Parametern $m = 2$ und $t = 2$. Nur die Einträge auf der Diagonalen sind explizit ausgeschrieben. Der Übersichtlichkeit wegen wurden 0-Einträge weggelassen und Einträge ungleich 0 durch einen Strich dargestellt.

Nachfolgend werden die durch $g_{i,k}(xX, yY)$ entstehenden Zeilenvektoren zu $m+1$ Zeilenblöcken zusammengefasst, die mit X_l bezeichnet werden. Der Zeilenblock X_l besteht aus den Vektoren, in denen X^l als höchste Potenz von X auftritt. Die Zeilenvektoren, die aus den Koeffizienten von $h_{j,k}(xX, yY)$ entstehen werden analog dazu zu t Y_j Zeilenblöcken zusammengefasst. Hier bezeichnet der Block Y_j die Vektoren, die durch Multiplikation von f mit y^j entstehen. Die Spaltenvektoren lassen sich ebenso in Blöcke unterteilen. Die Spalten mit der Bezeichnung $x^l y^j, l \geq j$ werden als X^l-Spaltenblöcke bezeichnet, analog werden die mit $x^i y^{i+l}$ überschriebenen Spalten zu den Y^l-Spaltenblöcken zusammengefasst. Die jeweiligen Blocknummern sind ebenfalls in der Abbildung 3.4 eingezeichnet.

Die Berechnung der Parameter m und t und die Schranke für δ ist relativ aufwendig, aber nicht schwierig. Daher wird hier nur der Weg skizziert. Eine ganz analoge Rechnung folgt im weiteren Verlaufe dieses Abschnitts für den Angriff nach Blömer und May.

Zunächst muss die Determinante der Basis bestimmt werden, was sehr einfach ist, da es sich um eine untere Dreiecksmatrix handelt und sie sich daher als Produkt der Diagonaleneinträge errechnet. Dazu ermitteln Boneh und Durfee zunächst das Produkt der Diagonaleneinträge der X-Blöcke und anschließend das der Y-Blöcke. Setzt man $X = e^{\delta}$ und $Y = e^{\frac{1}{2}}$ ein, erhält man eine Formel zur Berechnung der Determinante, abhängig von m, t, δ und e.

Nach der LLL-Reduktion werden die Polynome $H_1(xX, yY)$ und $H_2(xX, yY)$ aus dem

ersten und zweiten Zeilenvektor der reduzierten Basismatrix gebildet. Um sicherzustellen, dass $H_1(x_0, y_0) = 0 \wedge H_2(x_0, y_0) = 0$ gilt, muss nach Satz 3.5 $\|H_1(x_0, y_0)\| < e^m/\sqrt{w} \wedge \|H_2(x_0, y_0)\| < e^m/\sqrt{w}$ gelten. Dazu muss Bedingung 3.1.5 erfüllt sein. Demnach müssen m und t so gewählt werden, dass gilt:

$$\det(L) < 2^{-\frac{w(w-1)}{4}} e^{m(w-1)} w^{-\frac{w-1}{2}}$$

Da $w = (m+1)(m+2)/2 + t(m+1)$ ist, erhalten wir eine Ungleichung abhängig von m, t, δ und e. Dazu kann ein optimales t bestimmt werden und e kann einfach aus der Ungleichung gekürzt werden. Aufwendige Berechnungen zeigen, dass sich für δ die folgende Schranke abhängig von m ergibt:

$$\delta < \frac{7}{6} - \frac{1}{3}\sqrt{7 + \frac{16}{m} + \frac{4}{m^2}} + \frac{5}{6m}$$

Für $m \to \infty$ ergibt sich demnach:

$$\delta < \frac{7}{6} - \frac{\sqrt{7}}{3} \approx 0,2847$$

Das zeigt, dass diese Methode theoretisch bis zu einer Grenze von $d < N^{0,2847}$ funktioniert. Allerdings muss bereits für $\delta = 0.27$ $m \geq 10$ gewählt werden, was zu einem Gitter der Dimension 86 führt.

Wie oben erwähnt, stellen Boneh und Durfee anschließend eine Methode vor, mittels derer die theoretische Grenze von δ auf $\delta \approx 0,292$ verbessert werden kann. Die Hauptidee dabei besteht darin, diejenigen Zeilen zu löschen, die einen großen Beitrag zur Determinante haben. Dadurch entsteht allerdings eine Gitterbasis, deren Matrix nicht mehr dreieckig ist, was die Determinantenberechnung erschwert.

Hier wird nun das Verfahren von Blömer und May [BM01] beschrieben. Es basiert auf der oben beschriebenen Gitterbasis und löscht ebenfalls Zeilen heraus, die einen großen Beitrag zu Determinante liefern, allerdings werden anschließend auch die entsprechenden Spalten gelöscht, sodass die Matrix erneut in Dreiecksform ist. Dadurch kann auch eine kleinere Gitterbasis erzielt werden, was der Laufzeit zugute kommt und höhere praktische Werte für δ zulässt. Nach der Reduktion dieser Basis durch LLL müssen die gelöschten Spalten wiederhergestellt werden, um die Polynome daraus zu erhalten. Nachfolgend werden diese Schritte genauer erläutert und die theoretische Anwendbarkeit des Verfahrens bis $d \approx 0,290$ gezeigt.

Nachdem die Basis genau wie oben geschildert aufgestellt ist, werden folgende Einträge gelöscht:

- Alle Zeilenvektoren aus den ersten $m-t-1$ X-Zeilenblöcken, also Zeilenblock X_0 bis X_{m-t-1} einschließlich.
- Aus den Y_i-Zeilenblöcken jeweils die ersten $m+i-t$ Zeilenvektoren.
- Zu den gelöschten Zeilen die entsprechenden Spalten, sodass die Dreiecksform der Matrix wiederhergestellt ist.

		X^0	X^1		X^2			Y^1		
		1	x	xy	x^2	x^2y	x^2y^2	y	xy^2	x^2y^3
X_0	e^2	1								
X_1	xe^2		X							
	fe	-1	AX	XY						
X_2	x^2e^2				X^2					
	xfe		X		AX^2	X^2Y				
	f^2	1	$-2AX$	$-2XY$	A^2Y^2	$2AX^2Y$	X^2Y^2			
Y_1	ye^2							Y		
	yfe			AXY				Y	XY^2	
	yf^2			$-2AXY$		A^2X^2Y	$2AX^2Y^2$	Y	$-2XY^2$	X^2Y^3

Abbildung 3.5: Gitterbasis für $m=2$ und $t=1$ nach Boneh/Durfee.

Abbildung 3.5 zeigt eine Gitterbasis nach Boneh und Durfee bevor die Zeilen und Spalten nach oben genanntem Schema gelöscht wurden. In dieser Darstellung wurden die Potenzen von e nicht eingetragen, um die Übersichtlichkeit nicht zu gefährden.

		X^1		X^2			Y^1
		x	xy	x^2	x^2y	x^2y^2	x^2y^3
X_1	xe^2	X					
	fe	AX	XY				
X_2	x^2e^2			X^2			
	xfe	X		AX^2	X^2Y		
	f^2	$-2AX$	$-2XY$	A^2Y^2	$2AX^2Y$	X^2Y^2	
Y_1	yf^2		$-2AXY$		A^2X^2Y	$2AX^2Y^2$	X^2Y^3

Abbildung 3.6: Gitterbasis für $m=2$ und $t=1$ nach Blömer/May.

Abbildung 3.6 zeigt die Basis, nachdem die Zeilen und Spalten gelöscht wurden. Wie bereits erwähnt, handelt es sich nun nicht mehr um die Koeffizienten der Polynome $g_{i,k}(xX,yY)$ und $h_{j,k}(xX,yY)$. Es wird also eine Vorschrift benötigt, mittels derer die gelöschten Spaltenvektoren der neuen Matrix wiederhergestellt werden können. Blömer und May stellen dazu folgende Lemmata auf:

Lemma 3.1. *Alle gelöschten Spalten in den Spaltenblöcken $X^i, i<m-t$ sind Linearkombinationen von Spalten der verkleinerten Matrix. Die Koeffizienten für die Spaltenvekto-*

ren aus $X^l, l \geq m - t$ in diesen Linearkombinationen sind beschränkt durch $\frac{1}{(XY)^{l-i}} \cdot c$, wobei c nur von m und t abhängt.

Lemma 3.2. *Alle gelöschten Spalten $x^i y^{i+l}, i < m - t + l$ sind Linearkombinationen von Spalten der Y^l Blöcke der verkleinerten Matrix. Der Koeffizient für einen Spaltenvektoren $x^k y^{k+l}$ in diesen Linearkombinationen ist beschränkt durch $\frac{1}{(XY)^{i-k}} \cdot c$, wobei c nur von m und t abhängt.*

Die Beweise für diese Lemmata sind konstruktiv. Für Lemma 3.1 wird der Beweis in [BM01] vollständig geführt und mit Hinweis auf die Analogie dazu wird Lemma 3.1 dort nicht bewiesen. Beide Beweise finden sich jedoch ausführlich in [May03]. Hier werden sie nur skizziert, da die umfangreichen Berechnungen den Rahmen sprengen würden. Zunächst wird die geforderte Vorschrift zur Berechnung der gelöschten Spalten für die X^l Spaltenblöcke angegeben:

Behauptung 3.1. *Wenn $x^i y^j$ eine gelöschte Spalte eines X^l Spaltenblocks ist, dann ist $x^i y^j$ eine Linearkombination der Spalten $x^{i+1}y^{j+1}, x^{i+2}y^{j+2}, \ldots, x^m y^{m-i+j}$. Der Koeffizient der Spalte $x^{i+b}y^{j+b}, b = 1, \ldots, m - i$ in dieser Linearkombination ist gegeben durch:*

$$-\frac{1}{(XY)^b}\binom{j+b}{j}$$

Damit ist es möglich, gelöschte Spaltenvektoren aus anderen Spalten rechts davon wiederherzustellen.

Grundlage des Beweises sind die Ausgangsgleichungen

$$g_{i,k}(x,y) := x^i (f(x,y))^k e^{m-k} \text{ und } h_{j,k}(x,y) = y^j (f(x,y))^k e^{m-k}.$$

Man muss nun eine Möglichkeit finden, die gelöschten Monome dieser Polynome aus den verbliebenen Monomen zu berechnen. Dazu sieht man sich an, wie die Potenzen von $f(x,y)$ mittels des Binomischen Lehrsatzes entwickelt werden können. Erkennt man, dass $f^k(xX, yY) = (xX(A + yY) - 1)^k$ gebildet wird, durch

$$\sum_{p=0}^{k}\sum_{q=0}^{p}(-1)^{k+p}\binom{k}{p}\binom{p}{q}A^{p-q}X^p Y^q x^p y^q,$$

kann man die Behauptung 3.1 jeweils für die Zeilenblöcke X_l und Y_l zeigen.

Ebenso gibt es eine Vorschrift, mittels derer die gelöschten Spalten der Spaltenblöcke Y^l berechnet werden können. Der Beweis verläuft wie der letzte und das Ergebnis ist

dem vorigen sehr ähnlich. Für die Koeffizienten der Spalten der Y^l Spaltenblöcke ergibt sich:

Behauptung 3.2. *Wenn $x^i y^j$ eine gelöschte Spalte des Y^l Spaltenblocks ist, dann ist $x^i y^j$ eine Linearkombination der Spalten $x^{i+1}y^{j+1}, x^{i+2}y^{j+2}, \ldots, x^{m-j+i}y^m$. Der Koeffizient der Spalte $x^{i+b}y^{j+b}, b = 1, \ldots, m-i$ in dieser Linearkombination ist gegeben durch:*

$$-\frac{1}{(XY)^b}\binom{i+b}{i}$$

Sollte ein Vektor, der zur Rekonstruktion benötigt wird ebenfalls gelöscht sein, errechnet er sich nach der gleichen Formel. Daher ist es bei der Rekonstruktion sinnvoll, zunächst die in der Matrix rechts stehenden gelöschten Spalten zu rekonstruieren.

Man beachte, dass die Rekonstruktion erst nach der LLL-Reduktion des Gitters stattfindet. Die Behauptungen gelten dann jedoch immer noch, da die Zeilen der reduzierten Gitterbasenmatrix Linearkombinationen der ursprünglichen Matrix sind.

Anschließend wird gezeigt, dass die Norm der rekonstruierten Zeilenvektoren und somit die Norm der gefundenen Polynome „nicht signifikant größer“ [BM01, S. 9] ist, als die der nicht rekonstruierten Vektoren. Auf den genauen Beweis wird hier zugunsten des Leseflusses nicht weiter eingegangen.

Bleibt nun noch, die Größe der neuen Determinante und das Verhältnis von δ, m und t zu bestimmen. Das soll hier bewusst in aller Ausführlichkeit vorgestellt werden, da das Vorgehen exemplarisch für andere Verfahren ist, die auf dem bivariaten Verfahren nach Coppersmith basieren.

Da die Basismatrix in Dreiecksform ist, ist die Determinante das Produkt der Diagonalenelemente. Wir betrachten diese Einträge getrennt nach den X und Y Spaltenblöcken.

Für die X-Spaltenblöcke ergibt sich das folgende Produkt:

$$\det_x = \prod_{k=m-t}^{m} \prod_{l=0}^{k} e^{m-l} X^k Y^l$$

Entsprechend ergibt sich für die Y-Spaltenblöcke:

$$\det_y = \prod_{k=1}^{t} \prod_{l=m-t+k}^{m} e^{m-l} X^l Y^{k+l}$$

Nun wird $X = e^{\delta}$ und $Y = e^{\frac{1}{2}}$ eingesetzt und man erhält

$$\det_x = \prod_{k=m-t}^{m} \prod_{l=0}^{k} e^{m-l+k\delta+\frac{l}{2}}$$

und

$$\det_y = \prod_{k=1}^{t} \prod_{l=m-t+k}^{m} e^{m-l+l\delta+\frac{k+l}{2}}.$$

Ähnlich wie im Verfahren nach Boneh und Durfee werden nun m, t und δ gesucht, sodass

$$\det(L) < 2^{-\frac{w(w-1)}{4}} e^{m(w-1)} w^{-\frac{w-1}{2}}$$

gilt. Dabei ist $w = (m+1)(t+1)$ die Dimension der Gitterbasis. Wir ignorieren im Folgenden die Terme, die nicht von e abhängen, da ihr Beitrag nicht erheblich ist und erhalten die Ungleichungen:

$$\det_x \cdot \det_y \leq e^{m(m+1)(t+1)} \tag{3.13}$$

$$\Leftrightarrow \prod_{k=m-t}^{m} \prod_{l=0}^{k} e^{m-l+k\delta+\frac{l}{2}} \prod_{k=1}^{t} \prod_{l=m-t+k}^{m} e^{m-l+l\delta+\frac{k+l}{2}} < e^{m(m+1)(t+1)} \tag{3.14}$$

$$\Leftrightarrow \frac{(1+t)(2t^2\delta + t^2 - 6\delta mt + 2t - 2\delta t - 3m^2 + 12\delta m^2 + 12\delta m - 3m)}{12} < 0 \tag{3.15}$$

Für die linke Seite von Ungleichung 3.15 ist t optimal, wenn gilt:

$$t = \frac{12\delta m - 6 + 2\sqrt{-36\delta^2 m^2 - 36\delta m + 3 + 12\delta^2 - 36\delta^2 m - 18\delta m^2 - 6\delta + 9m^2 + 9m}}{2(6\delta + 3)}$$

Setzt man dies in Ungleichung 3.15 ein und formt nach δ um, so erhält man:

$$\delta < \frac{-2 - 6m^2 - 12m + 6\sqrt{12m^2 + 3m + 6m^4 + 15m^3}}{2(15m^2 + 18m - 1)}$$

Für $m \to \infty$ ergibt sich als Schranke für δ demnach

$$\frac{\sqrt{6} - 1}{5} \approx 0,290$$

Abbildung 3.7 zeigt das Verhalten von δ in Abhängigkeit von m.

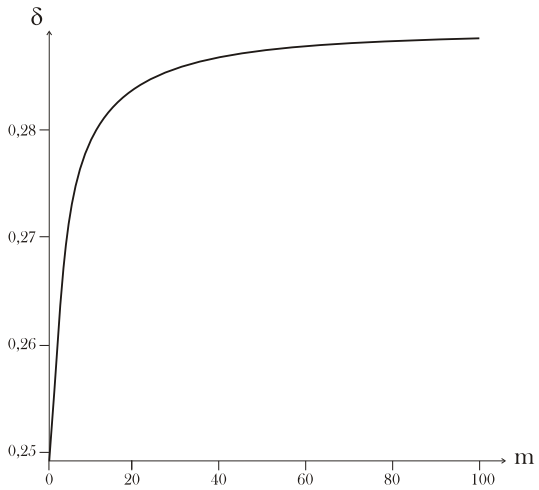

Abbildung 3.7: δ in Abhängigkeit von m

3.4 Faktorisieren mit teilweise bekanntem p oder q

Dieser Abschnitt befasst sich mit der Faktorisierung von N, wenn Teile der Faktoren p oder q bekannt sind. Diese Technik wird auch als „Factoring with a hint“ (deutsch: *Faktorisieren mit einem Hinweis*) bezeichnet und basiert auf dem univariaten Verfahren von Coppersmith. Der erste Unterabschnitt dieses Abschnitts beschäftigt sich mit dem Fall, in dem die oberen Bits von p oder q bekannt sind. Im zweiten Unterabschnitt wird eine Abwandlung dieses Verfahrens auf bekannte untere Bits von p oder q vorgestellt.

3.4.1 Bekannte höchstwertige Bits von p oder q

Dan Boneh, Glenn Durfee und Nick Howgrave-Graham stellten in [BDHG99] ein Verfahren vor, mit dem ein RSA-Modulus $N = p^r q$ dann effizient faktorisiert werden kann, wenn r relativ groß ist. Dabei verwenden sie unter anderem das folgende Lemma, das Ausgangspunkt für unsere weiteren Betrachtungen sein soll.

Lemma 3.3. *Sei $N = p^r q$ mit $q < p^c$ bei gegebenem c. Darüber hinaus sei P eine Zahl, so dass gilt:*

$$|P - p| < p^{1-\frac{c}{r+c}-2\frac{r}{d}}$$

Dann kann p aus N, r, c und P in der Zeit berechnet werden, die benötigt wird, um den LLL-Algorithmus auf einem Gitter mit der Dimension d anzuwenden.

Das heißt, dass für $q < p$ und $N = pq$ für $d \to \infty$ bereits die Hälfte der oberen Bits von p ausreichen, um N zu faktorisieren.

Wir gehen davon aus, dass $N = p^r q$ gegeben ist. Außerdem sei die Zahl P gegeben, die p in den t höchstwertigen Bits entspricht. Wir können auch schreiben $|P - p| < X$ für ein großes X. Die Aufgabe besteht nun darin p bei gegebenen N, P und r zu bestimmen. Wir stellen dazu das Polynom

$$f(x) = (P + x)^r$$

auf und bemerken, dass es im Punkt $x_0 = p - P$ eine Nullstelle modulo p^r besitzt, für die gilt: $|x_0| < X$. Zwar ist p^r unbekannt, aber wir kennen mit N ein Vielfaches davon. Wir suchen also die Nullstellen eines modularen Polynoms. Seien $m > 0$ und $d > 0$ Zahlen, die später genauer bestimmt werden. Wir stellen nun die Polynome

$$\begin{aligned} g_{i,k}(x) &:= N^m - kx^i f^k(x) && \text{für } k = 0, \ldots, m-1 \text{ und } i = 0, \ldots, r-1 \text{ und} \\ g_{j,m}(x) &:= x^j f^m(x) && \text{für } j = 0, \ldots, d - mr - 1 \text{ auf.} \end{aligned}$$

Alle diese Polynome $g_{i,k}(x), g_{j,m}(x)$, sowie auch alle ganzzahligen Linearkombinationen daraus, haben in x_0 eine Nullstelle modulo p^{rm}. Satz 3.4 legt nun nahe, unter diesen Linearkombinationen ein Polynom $h(x)$ mit Grad w zu suchen, das eine gewichtete Norm $\|h(xX)\| < p^{rm}/\sqrt{w}$ besitzt, da dann $h(x_0) = 0$ in ganz $\mathbb{Z}$ gilt. Um dieses $h(x)$ zu finden bedienen wir uns des LLL-Algorithmus. Zunächst wird also eine Gitterbasis M aus den Koeffizientenvektoren der $g_{i,k}(xX)$ und $g_{j,m}(xX)$ gebildet, die dann mittels LLL reduziert wird.

$$\begin{array}{l} g_{0,0}(xX) \\ g_{1,0}(xX) \\ g_{0,1}(xX) \\ g_{1,1}(xX) \\ g_{0,2}(xX) \\ g_{1,2}(xX) \\ g_{0,3}(xX) \\ g_{1,3}(xX) \\ g_{2,3}(xX) \end{array} \begin{bmatrix} N^3 & & & & & & & & \\ & XN^3 & & & & & & & \\ - & - & X^2N^2 & & & & & & \\ & - & - & X^3N^2 & & & & & \\ - & - & - & - & X^4N & & & & \\ & - & - & - & - & X^5N & & & \\ - & - & - & - & - & - & X^6 & & \\ & - & - & - & - & - & - & X^7 & \\ & & - & - & - & - & - & - & X^8 \end{bmatrix}$$

Abbildung 3.8: Beispiel einer Gitterbasis für $m = 2$ und $d = 9$ und $r = 2$.

Da die Basis in Dreiecksform ist, erhält man für ihre Determinante den Ausdruck:

$$\det(M) = \left(\prod_{k=0}^{m-1} \prod_{i=0}^{r-1} N^{m-k} \right) \left(\prod_{j=0}^{d-1} X^j \right) < N^{\frac{rm(m+1)}{2}} X^{\frac{d^2}{2}}$$

Wir bilden anschließend $h(xX)$ aus dem ersten Basisvektor der LLL-reduzierten Basis. Ungleichung 3.8 liefert dann für die Norm dieses Polynoms:

$$\|h(xX)\|^d \leq 2^{\frac{d^2}{2}} \det(L) \leq 2^{\frac{d^2}{2}} N^{\frac{rm(m+1)}{2}} X^{\frac{d^2}{2}}$$

Wir vereinfachen die Bedingung $\|h(xX)\| < p^{rm}/\sqrt{d}$ zu $\|h(xX)\| < p^{rm}$, da der Nenner $\sqrt{d}$ keinen großen Einfluss auf die folgende Berechnung hat, und erhalten durch Umformen die neue Bedingung

$$(2X)^{\frac{d^2}{2}} < p^{rmd} N^{\frac{-rm(m+1)}{2}}.$$

Da $q < p^c$ ist, setzen wir für $N < p^{r+c}$ ein und erhalten

$$(2X)^{\frac{d^2}{2}} < p^{rmd - \frac{r(r+c)m(m+1)}{2}}.$$

Je größer X sein darf desto weniger Bits von p müssen bekannt sein. Deshalb suchen wir nach einem m so, dass X möglichst groß wird. Wir formen dazu zunächst zu

$$X < \frac{1}{2} p^{\frac{2rmd - (r^2+rc)(m^2+m)}{d^2}}$$

um und erkennen, dass der rechte Ausdruck für $m_0 = \frac{d}{r+c} - \frac{1}{2}$ maximal wird. Wird dieser Wert für m eingesetzt, erhält man als neue Schranke nach aufwendigen Berechnungen:

$$X < \frac{1}{2} p^{1 - \frac{c}{r+c} - 2\frac{r}{d}}$$

Wir betrachten hier den Fall, dass $p < q$ und $N = pq$ ist. Daher ergibt sich $c = 1$ und $r = 1$. Mit dieser Methode kann nun N faktorisiert werden sobald

$$|P - p| < \frac{1}{2} p^{\frac{1}{2} - \frac{2}{d}} \tag{3.16}$$

ist. Da $p < q$ ist, muss p kleiner als $\sqrt{N}$ sein. Wenn n die Anzahl der Bits von N bezeichnet, dann ist p also kleiner als $2^{\frac{n}{2}}$. Wir können Ungleichung 3.16 daher umschreiben zu:

$$|P - p| < \frac{1}{2} 2^{\frac{n}{2}(\frac{1}{2} - \frac{2}{d})}$$

Da P eine Zahl ist, die p in den t oberen Bits entspricht, ergibt sich $|P - p| < 2^{\frac{n}{2} - t}$. Wir

erhalten also zusammengefasst die folgende Bedingung:

$$2^{\frac{n}{2}-t} \leq 2^{\frac{n}{2}(\frac{1}{2}-\frac{2}{d})}$$

Diese Bedingung ist erfüllt, sobald $d \geq \frac{n}{t-\frac{n}{4}}$ ist. Daraus folgt, dass $t > \frac{n}{4}$ sein muss, um das Verfahren anzuwenden. Allerdings ergibt sich für den Fall $t = \frac{n}{4} + 1$ als Gitterdimension $d = n$.

Die Laufzeit dieses Algorithmus beträgt $T_{LLL}(w, 2dw)$ und wird im Folgenden mit $T_{MSBFact}(n, t)$ bezeichnet.

```
MSBFact(N, P, t)
 1  n ← log(N)
 2  d ← ceil(n/(t − (n/4))
 3  m ← floor(w/2 − 1/2)
 4  X ← power(2, n/2 − t))
 5  f(x) ← (P + x)
 6  B ← buildLattice(N, X, m, d, P, f)
 7  B′ ← LLL(B)
 8  r(x) ← buildPoly(B′[1], X)
 9  roots[ ] ← findRoots(r(x))
10  for  each x_0 in roots[ ]
11  do
12      if (P + x_0) teilt N
13        then
14              p ← P + x_0
15              q ← N/p
16              return p, q
```

Abbildung 3.9: Algorithmus MSBFact

Abbildung 3.9 zeigt den Algorithmus zu diesem Verfahren in Pseudocode. Die Funktionen LOG, CEIL, FLOOR und POWER berechnen jeweils den Logarithmus, die nächste höhere bzw. niedrigere Ganzzahl sowie die Potenz. Es werden zunächst die Parameter für das Gitter und die Schranke X berechnet. Dann wird in der Funktion BUILDLATTICE die Gitterbasis, wie in diesem Abschnitt beschrieben, aufgestellt und anschließend LLL-reduziert. Aus dem ersten Vektor der gefundenen Gitterbasis B' wird nun in BUILDPOLY das Polynom $f(x)$ gebildet. Dazu müssen lediglich die entsprechenden Potenzen von X herausgerechnet werden. Nun werden die Nullstellen von $f(x)$ mit der Funktion FINDROOTS gesucht. Das kann zum Beispiel durch Näherungsverfahren geschehen. Zum

Schluss werden alle gefundenen Nullstellen geprüft und p und q ausgegeben, wenn die Faktorisierung von N gefunden wird.

Mit diesem Verfahren ist es auch möglich N zu faktorisieren, wenn weniger als $\frac{n}{4}$ Bits zur Verfügung stehen. Dazu werden die fehlenden Bits alle aufgezählt und jeweils MSBFact aufgerufen. Für jedes fehlende Bit verdoppelt sich dabei jedoch die Laufzeit. Auf diese Weise ist es auch möglich N ganz ohne Kenntnis von Bits von p zu faktorisieren. Allerdings kommt zu der Laufzeit von MSBFact dann ein großer multiplikativer Faktor hinzu, sodass die Laufzeit sehr schlecht wird.

3.4.2 Bekannte niederwertigste Bits von p oder q

In [Dur02] beschreibt Durfee eine Variante des Verfahrens aus dem letzten Abschnitt. Damit ist es möglich N zu faktorisieren, wenn die t niederwertigsten Bits des kleineren Faktors p bekannt sind. Dabei ist das Vorgehen dem zuvor vorgestellten sehr ähnlich. Daher werden hier nur die Unterschiede dargestellt.

Ausgangspunkt ist ein RSA Modulus N von n Bit Länge. Außerdem sei ein P gegeben, das dem Faktor p in den niederwertigsten $t > 0$ Bits entspricht. Dann lässt sich auch schreiben:

$$P \equiv p \mod R, \qquad \text{mit } R \geq 2^t$$

Hier suchen wir also die verbleibenden höchstwertigen Bits x_0 von p. Dieses Problem kann ausgedrückt werden durch die Formel $p = x_0 R + P \Leftrightarrow x_0 = \frac{p-P}{R}$. Daher bilden wir das gegenüber dem letzten Verfahren leicht abgewandelte Polynom:

$$f(x) = \frac{P}{R} + x$$

Man kann davon ausgehen, dass N und R teilerfremd sind, da sonst die Faktorisierung von N sofort bekannt wäre. Dieser Umstand erlaubt es nun das multiplikative Inverse $a = R^{-1} \mod N$ zu berechnen. Wir setzen dieses a in das Polynom ein und erhalten:

$$f(x) = aP + x$$

Man erkennt wie zuvor, dass für den gesuchten Wert x_0 das Polynom eine Nullstelle modulo p besitzt. Nun wird das gleiche Verfahren, wie das zu Lemma 3.3 verwendet, um x_0 zu finden. Die Entwicklung der Parameter verläuft ebenfalls genau wie zuvor. So erhält man die Lösung x_0 sobald $t > \frac{n}{4}$ ist bei einer Wahl von $d \geq \frac{n}{t-\frac{n}{4}}$. Die Laufzeit dieses Algorithmus beträgt $T_{LLL}(w, 6dw)$ und wird im Folgenden mit $T_{LSBFact}(n, t)$ bezeichnet.

Abbildung 3.10 zeigt das Verfahren in Pseudocode. Es entspricht in weiten Teilen

```
LSBFact(N, P, t)
 1  n ← log(N)
 2  d ← ceil(n/(t − (n/4))
 3  m ← floor(w/2 − 1/2)
 4  R ← power(2, t))
 5  X ← power(2, n/2 − t))
 6  a ← inv(R, N)
 7  f(x) ← (aP + x)
 8  B ← buildLattice(N, X, m, d, P, f)
 9  B′ ← LLL(B)
10  r(x) ← buildPoly(B′[1], X)
11  roots[ ] ← findRoots(r(x))
12  for each x_0 in roots[ ]
13  do
14      if (P + x_0 ∗ R) teilt N
15        then
16              p ← P + x_0 ∗ R
17              q ← N/p
18              return p, q
```

Abbildung 3.10: Algorithmus LSBFact

MSBFact. Es wird zusätzlich das multiplikative Inverse von $R^{-1} \mod N$ berechnet. Dazu kann in der Funktion inv zum Beispiel der erweiterte euklidische Algorithmus verwendet werden. Außerdem wird das geänderte Polynom $f(x)$ verwendet, um die Gitterbasis zu erstellen. Bei der Auswertung der gefundenen Nullstellen wird ebenfalls dem geänderten Polynom Rechnung getragen.

3.5 Teilweise bekannter Schlüssel

3.5.1 Übersicht

In diesem Kapitel werden Angriffe vorgestellt, die das RSA-Kryptosystem brechen, wenn Teile des geheimen Exponenten d bekannt sind. Diese werden ihrer Art nach in zwei Kategorien unterschieden. Die einen funktionieren bei bekannten höchstwertigen Bits von d, die anderen bei bekannten niederwertigsten Bits. Hier werden nachfolgend die englischen Abkürzungen MSB für „most significant bits“ und LSB für „least significant bits“ verwendet.

Viele der so genannten Seitenkanalangriffe liefern dem Angreifer nach und nach ei-

ne bestimmte Anzahl zusammenhängender Bits von d. Das liegt an der Funktionsweise dieser Angriffe, die in Abschnitt 4.2 genauer untersucht werden. Die hier vorgestellten Angriffe erlauben es in solchen Fällen den Seitenkanalangriff früher abzubrechen. Je nach Verfahren und abhängig vom öffentlichen Exponenten e reichen bereits $\frac{1}{4}$ der Bits von d, um das RSA-System zu brechen.

Erste Angriffe dieser Art zeigten Boneh, Durfee und Frankel auf der Asiacrypt '98 [BDF98]. Sie stellten ein Verfahren für bekannte LSB von d vor, das für kleine e funktioniert und drei Verfahren für bekannte MSB, die alle bis maximal $e < N^{\frac{1}{2}}$ anwendbar sind. Sie warfen damals die Frage auf, ob es ähnliche Angriffe auch für $e > N^{\frac{1}{2}}$ geben kann. Auf der Crypto '03 [JB03] stellten Blömer und May zwei weitere Angriffe für bekannte LSB vor, die bis zu der Grenze $e < N^{\frac{1}{2}}$ bzw. $e < N^{\frac{7}{8}}$ funktionieren. Auch für bekannte MSB konnten sie einen Angriff präsentieren, der für $e > N^{\frac{1}{2}}$ anwendbar ist.

Zunächst werden nun die verschiedenen Verfahren in ihren Ergebnissen beschrieben, um eine Übersicht zu geben. Anschließend werden sie genauer vorgestellt.

Bekannte LSB von d

Beginnen wir mit dem Ergebnis von Boneh, Durfee und Frankel aus [BDF98].

Satz 3.6 (BDF1). *Sei $N = pq$ ein n-bit RSA-Modulus mit $N \equiv 3 \mod 4$ und sei $e < \frac{1}{8}N^{\frac{1}{4}}$. Dann kann N in polynomieller Zeit in $\log(N)$ und e faktorisiert werden, wenn die*

$$\frac{n}{4} \text{ LSB von } d$$

bekannt sind.

Das bedeutet, dass bei kleinem e bereits $\frac{n}{4}$ der LSB von d genügen, um N zu faktorisieren. Allerdings gilt hier die starke Einschränkung, dass $N = 3 \mod 4$ sein muss, was bei zufällig verteilten p und q nur genau für jedes zweite N zutrifft.

Das folgende Ergebnis von Blömer und May aus [JB03] benötigt zwar mehr LSB von d, ist dafür allerdings für fast alle $e < N^{\frac{1}{2}}$ gültig.

Satz 3.7 (BM1). *Sei N ein RSA-Modulus und seien $0 < \alpha, \epsilon < \frac{1}{2}$. Für alle außer einem $O\left(\frac{1}{N^\epsilon}\right)$-Teil der öffentlichen Exponenten e aus dem Interval $[3, N^\alpha]$ gilt: Sei d der geheime Schlüssel. Angenommen man erhält d_0 und M mit $d = d_0 \mod M$ mit*

$$N^{\alpha+\frac{1}{2}+\epsilon} \leq M \leq 2N^{\alpha+\frac{1}{2}+\epsilon},$$

dann kann N in polynomieller Zeit faktorisiert werden.

Zusätzlich zu diesem beweisbaren Verfahren zeigen Blömer und May in der gleichen Veröffentlichung ein heuristisches Verfahren für alle $e < N^{\frac{7}{8}}$, welches erneut auf der Annahme basiert, dass das bivariate Verfahren zwei algebraisch unabhängige Polynome hervorbringt.

Satz 3.8 (BM2). *Unter der Annahme, dass das bivariate Verfahren zwei algebraisch unabhängige Polynome erzeugt, gilt:*

Für jedes $\epsilon > 0$ existiert ein N_0, sodass für jedes $N \geq N_0$ gilt: Sei (N, e) ein öffentlicher RSA-Schlüssel mit $\alpha := \log_N(e) \leq \frac{7}{8}$ und sei d der geheime Schlüssel. Angenommen man erhält d_0 und M so, dass $d = d_0 \mod M$ mit

$$M \geq N^{\frac{1}{6}+\frac{1}{3}\sqrt{1+6\alpha}+\epsilon}$$

gilt, dann kann N in polynomieller Zeit faktorisiert werden.

Bekannte MSB von d

Für bekannte MSB von d zeigen Boneh, Durfee und Frankel drei Verfahren, die sich zum Teil darin unterscheiden, ob die Faktorisierung von e bekannt ist. Auch hier werden zunächst nur die Ergebnisse vorgestellt.

Satz 3.9 (BDF2). *Sei $N = pq$ ein n-bit RSA-Modulus. Wenn e eine t-bit Primzahl aus dem Intervall $\left[N^{\frac{1}{4}}, N^{\frac{1}{2}}\right]$ ist, so kann N in polynomieller Zeit abhängig von $\log N$ faktorisiert werden, wenn die*

t MSBs von d

bekannt sind.

Das heißt, wenn e eine Primzahl ist und in der Größenordnung $N^{\frac{1}{4}}$ liegt, reichen bereits wenig mehr als ein Viertel der MSB von d aus, um N zu faktorisieren. Wenn e keine Primzahl ist, kann eine modifizierte Version des vorherigen Verfahrens angewendet werden und man kommt zu den gleichen Grenzen bei schlechterer Laufzeit.

Satz 3.10 (BDF3). *Sei $N = pq$ ein n-bit RSA-Modulus. Sei e ein t-bit Produkt von r verschiedenen Primzahlen und e sei aus dem Intervall $\left[N^{\frac{1}{4}}, N^{\frac{1}{2}}\right]$. Wenn die Faktorisierung von e bekannt ist, so kann N in polynomieller Zeit abhängig von $\log N$ und 2^r faktorisiert werden, wenn die*

t MSBs von d

bekannt sind.

Sollte die Faktorisierung von e unbekannt sein, kann das folgende Verfahren verwendet werden.

Satz 3.11 (BDF4). *Sei $N = pq$ ein n-bit RSA-Modulus. Sei $e \leq N^{\frac{1}{2}}$ eine t-bit Zahl von unbekannter Faktorisierung. Sei weiter $d > \epsilon N$ für ein gegebenes ϵ, so kann N in polynomieller Zeit abhängig von $\log N$ und $\frac{1}{\epsilon}$ faktorisiert werden, wenn die*

$$n - t \text{ MSBs von } d$$

bekannt sind.

Der folgende Satz ist nicht als Algorithmus in [BDF98] enthalten, er geht allerdings laut [JB03] aus einem anderen dort aufgestellten Satz hervor. Er wird hier daher auch nur als Satz vorgestellt.

Satz 3.12 (BDF5). *Sei $N = pq$ ein n-bit RSA-Modulus. Sei $e \leq N^{\frac{1}{2}}$ eine t-bit Zahl von unbekannter Faktorisierung. Sei weiter $d > \epsilon N$ und $|p - q| > \epsilon\sqrt{N}$ für ein beliebiges ϵ. Dann kann N in polynomieller Zeit in $\log N$ und $\frac{1}{\epsilon}$ faktorisiert werden, wenn die*

$$\frac{3}{4}n \text{ MSBs von } d$$

bekannt sind.

Blömer und May zeigen einen Angriff bei bekannten MSB, der für $e \in [N^{\frac{1}{2}}, N^{\frac{\sqrt{6}-1}{2}}]$ anwendbar ist. Dabei kommt eine trivariate Variante des Verfahrens nach Coppersmith zum Einsatz, also modulare Polynome in drei Unbekannten. Auch dieser Angriff stützt sich auf die Annahme, dass die Resultanten der gefundenen Polynome nicht das Nullpolynom darstellen, die Polynome somit nicht algebraisch abhängig sind. In den durchgeführten Experimenten hat sich diese Annahme auch bestätigt. Blömer und May haben kein Gegenbeispiel gefunden.

Satz 3.13 (BM3). *Unter der Annahme, dass das trivariate Verfahren nach Coppersmith drei algebraisch unabhängige Polynome erzeugt gilt:*

Zu gegebenem $\epsilon > 0$ existiert ein N_0 so, dass für jedes $N > N_0$ das Folgende gilt: Sei (N, e) ein öffentlicher RSA-Schlüssel mit $\alpha := \log_N(e) \in \left[N^{\frac{1}{2}}, N^{\frac{\sqrt{6}-1}{2}}\right]$. Angenommen man erhält eine Näherung $\tilde{d}$ von d mit

$$|d - \tilde{d}| \leq N^{\frac{1}{8}\left(5 - 2\alpha - \sqrt{36\alpha^2 + 12\alpha - 15}\right) - \epsilon},$$

dann kann N in polynomieller Zeit in $\log N$ faktorisiert werden.

Übersicht über die Verfahren

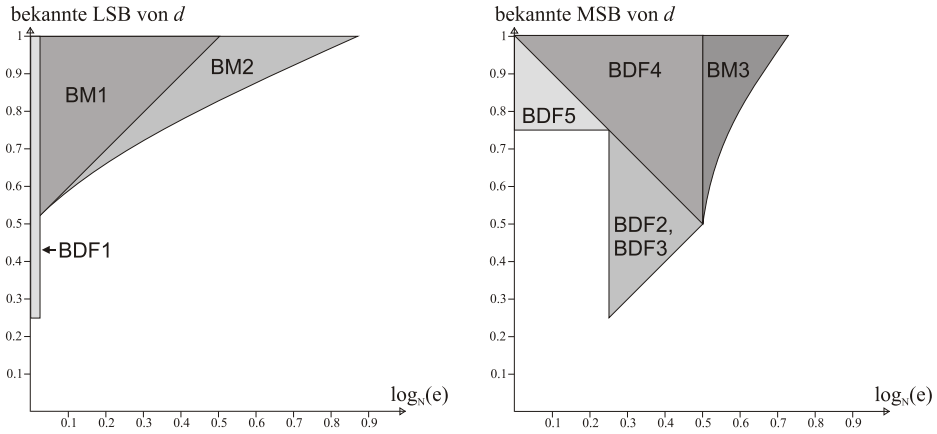

Abbildung 3.11: Die verschiedenen Angriffe auf teilweise bekannte d. Abb. nach [JB03, S. 3]

Verfahren	Art	$\alpha := \log_N(e)$	benötigte Bits von d	Bedingung
BDF1	LSB	$O(\log N \log N)$	$\frac{1}{4}$	$N = 3 \mod 4$
BM1	LSB	$[0, \frac{1}{2}]$	$\frac{1}{2} + \alpha$	alle außer $O(n^{\alpha-\epsilon})$ e's
BM2	LSB	$[0, \frac{7}{8}]$	$\frac{1}{6} + \frac{1}{3}\sqrt{1+6\alpha}$	heuristisch
BDF2/3	MSB	$[\frac{1}{4}, \frac{1}{2}]$	α	e prim od. fakt. bek.
BDF4	MSB	$[0, \frac{1}{2}]$	$1 - \alpha$	$\frac{d}{\Phi(N)} = \Omega(1)$
BDF5	MSB	$[0, \frac{1}{2}]$	$\frac{3}{4}$	$\frac{d}{\Phi(N)}, \frac{\lvert p-q\rvert}{\sqrt{N}} = \Omega(1)$
BM3	MSB	$[\frac{1}{2}, \frac{\sqrt{6}-1}{2}]$	$1 - \frac{5-2\alpha-\sqrt{36\alpha^2+12\alpha-15}}{8}$	heuristisch

Abbildung 3.12: Übersicht der Angriffe auf teilweise bekannte d, Abb. nach [JB03, S. 5]

Abbildung 3.11 zeigt die Verfahren in einer graphischen Darstellung. Auf der Abszissenachse ist die Größe von e im Verhältnis zu N abgetragen, um zu verdeutlichen, welches Verfahren sich für welchen Bereich von e eignet. Auf der Ordinatenachse ist abgetragen, wie viel von d bekannt sein muss, damit der Angriff funktioniert. Das heißt, je weiter eine Fläche nach unten reicht desto weniger muss von d bekannt sein, um den Angriff durchzuführen. Hier erkennt man bereits, dass die theoretisch stärksten Angriffe BDF1,

BDF2 und BDF3 im günstigsten Fall bereits mit einem Viertel der Bits auskommen.

Abbildung 3.12 fasst die Ergebnisse noch einmal in tabellarischer Form zusammen. Dabei wird jeweils der Bereich für e angegeben, der für den jeweiligen Angriff anfällig ist. Es fällt auf, dass nicht nur abhängig von e ist, ob ein Angriff überhaupt durchführbar ist, sondern oft ist von e auch abhängig, wie viel von d bekannt sein muss.

Im Folgenden werden die einzelnen Methoden in ihren Möglichkeiten, Einschränkungen und Vorgehensweisen genau beschrieben.

3.5.2 Angriffe bei bekannten LSB von d

BDF1

Satz 3.14 (BDF1). *Sei $N = pq$ ein n-bit RSA-Modulus mit $N \equiv 3 \mod 4$ und sei $e < \frac{1}{8}N^{\frac{1}{4}}$. Dann kann N in polynomieller Zeit in $\log(N)$ und e faktorisiert werden, wenn die*

$$\frac{n}{4} \text{ LSB von } d$$

bekannt sind.

Ausgangspunkt diese Angriffs ist die folgende elementare Gleichung des RSA-Kryptosystems:

$$ed \equiv 1 \mod \Phi(N)$$

Diese kann leicht zu

$$ed - k\Phi(N) = ed - k(N - p - q + 1) = 1 \tag{3.17}$$

umgeformt werden. Da $\Phi(N) > d$ ist folgt daraus, dass $k < e$ sein muss. In diesem Abschnitt gehen wir von einem sehr kleinen e aus. Klein heißt in diesem Fall, dass alle möglichen Werte für k ausprobiert werden können. Wenn d_0 den letzten m Bits von d entspricht, dann kann man auch schreiben $d_0 \equiv d \mod 2^m$. Wenn man die Gleichung 3.17 $\mod 2^m$ reduziert, gilt demnach auch:

$$ed_0 \equiv 1 + k(N - p - q + 1) \mod 2^m$$

Setzt man $q = N/p$, dann ist $p \mod 2^m$ eine Lösung für x in der Gleichung

$$ed_0 \equiv 1 + k(N - x - N/x + 1) \mod 2^m.$$

Daraus erhält man durch Umformung die quadratische modulare Gleichung

$$kx^2 + (ed_0 - k(N+1) - 1)x + kN \equiv 0 \mod 2^m. \tag{3.18}$$

Wenn k' eine Vermutung für k bezeichnet, ergibt sich daraus

$$k'x^2 + (ed_0 - k'(N+1) - 1)x + k'N \equiv 0 \mod 2^m. \tag{3.19}$$

Der Angreifer löst nun diese quadratische modulare Gleichung für alle $k' < e$. Das ist nicht so einfach, wie in $\mathbb{Z}$ und man erhält möglicherweise auch mehr als zwei Lösungen für jedes k. Wenn $k' = k$ ist, dann ist unter den Lösungen ein x_i, das p in den m niederwertigsten Bits entspricht. In Abschnitt 3.4 wird gezeigt, wie man N faktorisieren kann, wenn mehr als die Hälfte von einem der Faktoren p oder q bekannt ist. Demnach muss für jedes gefundene x_i der Algorithmus LSBFact aufgerufen werden und wenn $k' = k$ ist, dann ist unter den Lösungen x_i eine, die zur Faktorisierung von N führt. Die Frage, wie viele solcher x_i es gibt und wie man sie berechnet, beantwortet das folgende Lemma aus [Dur02]:

Lemma 3.4. *Sei $N = pq$ ein RSA-Modulus mit $N \equiv 3 \mod 4$ und $4 < \sqrt{N}/2 < q < p < 2\sqrt{N}$. Sei weiter $e \leq 2^{(n/4)-3}$ und k' und m seien Ganzzahlen. Dann gibt es einen Algorithmus, der zu gegebenen N, e, k' und m eine Liste von Lösungen $\{x_1, \ldots, x_l\}$ zu der Gleichung 3.19 berechnet und es gilt:*

1. *Wenn $k' = k$ ist, dann sind alle Lösungen zu Gleichung 3.19 in der Liste.*
2. *Sei $t_{k'}$ die größte Ganze Zahl so, dass $2^{t_{k'}}$ Teiler von k' ist, dann ist die Länge der Liste $l \leq 2^{t_{k'}}$ und der Algorithmus läuft in der Zeit $O(n^3 2^{t_{k'}})$.*

Angenommen $k' = k$. Man beachte, dass k alle Koeffizienten der Gleichung 3.18 teilt. Für den quadratischen und den konstanten Term ist diese Aussage trivial, für den linearen Term lässt sich mit Gleichung 3.17 leicht zeigen, dass $ed_0 - k(N+1) - 1 = -k(p+q)$ ist. Wenn 2^{t_k} die größte Zweierpotenz ist, die k teilt, lässt sich k zerlegen in $k = 2^{t_k} w$, wobei w ungerade ist. Dann ist jede Lösung x für Gleichung 3.18 auch eine Lösung für die Gleichung

$$wx^2 - w(p+q)x + wN \equiv 0 \mod 2^{m-t_k}.$$

Da w ungerade ist, ist $w^{-1} \mod 2^{m-t_k}$ wohl definiert und man kann die Gleichung vereinfachen zu

$$x^2 - (p+q)x + N \equiv 0 \mod 2^{m-t_k}.$$

Durch quadratische Ergänzung erhält man schließlich

$$(x - (p+q)/2)^2 \equiv ((p-q)/2)^2 \mod 2^{m-t_k} \tag{3.20}$$

Diese Gleichung zu lösen ist der schwierige Teil des Angriffs. Von Boneh, Durfee und Frankel wird ein von Don Redmond [Red96] beschriebenes Verfahren verwendet, das modulare quadratische Gleichungen der Form $y^2 \equiv c \mod 2^u$ mit $c \equiv 1 \mod 8$ und $u \geq 3$ löst. Wir beschränken uns hier darauf zu zeigen, dass Gleichung 3.20 in der gewünschten Form ist. Das genaue Verfahren findet sich zum Beispiel im Anhang von [BDF98].

Da $N \equiv 3 \mod 4$ ist, folgt, dass $p \not\equiv q \mod 4$ ist und daher ist $((p-q)/2)^2 \equiv 1 \mod 8$. Aus $k < e$ folgt $t_k \leq \log_2 e$, außerdem gilt $e \leq 2^{(n/4)-3}$ und $m \geq n/4$, daher ist $m - t_k \geq 3$. Also ist Gleichung 3.20 in der Form, die nach dem Verfahren von Redmond gelöst werden kann. Ferner zeigt Redmond, dass es genau vier Lösungen zu dieser Gleichung gibt und wie diese berechnet werden können. Da für jede Zerlegung diese vier Lösungen gefunden werden, berechnen sich die Lösungen für ein k folgendermaßen: $\{x_1, x_2, x_3, x_4\}$ sind die vier Lösungen für Gleichung 3.20, dann sind die Lösungen für Gleichung 3.18 $x_i + j \cdot 2^{(n/4)-t_k}$ für $i \in 1, 2, 3, 4$ und $j \in 0, \ldots, 2^{t_k} - 1$.

Für den Fall, dass $k' = k$ ist, ergeben sich somit 2^{2+t_k} mögliche Lösungen, die durch das oben vorgestellte Verfahren alle gefunden werden. Wenn mehr als 2^{2+t_k} Lösungen gefunden werden, kann k' nicht k sein. Es kann also immer abgebrochen werden, nachdem die ersten 2^{2+t_k} Lösungen ausgegeben wurden.

Das Verfahren funktioniert unter Umständen auch, wenn $N \equiv 1 \mod 4$ ist, also p und q in den letzten beiden Binärstellen übereinstimmen. Ron Steinfeld und Yuliang Zheng [SZ01] untersuchen die Anwendbarkeit des Verfahrens für zufällige p und q und zeigen, dass eine Modifikation dieses Verfahrens eine Laufzeit von $O(T_{LSBFact}(n, m) \cdot n \cdot e \log_2 e)$ besitzt. Wenn p und q in vielen LSB übereinstimmt, kann der Angriff jedoch nicht mehr effizient angewendet werden.

Der Algorithmus des Angriffs ist in Abbildung 3.13 dargestellt. Die Schleife in Zeile 1 läuft über alle möglichen k'. Für jeden Kandidat k' wird in Zeile 2-4 die größte Zweierpotenz gesucht, die k' teilt. In Zeile 5 wird Gleichung 3.19 aufgestellt und anschließend werden die vier Lösungen mit der Gleichung von Redmond berechnet. In Zeile 7-9 werden die verbleibenden Lösungen für dieses k' berechnet. Anschließend wird für jede Lösung x_i, die für dieses k' gefunden wurde der Faktorisierungsalgorithmus aus Abschnitt 3.4 aufgerufen und bei Erfolg werden die Faktoren p und q ausgegeben.

Aus Satz 3.4 wissen wir, dass wenn k' ungerade ist, höchstens 4 Lösungen für Gleichung

```
BDF1(e, N, d_0)
 1  for k' ← 1 to e
 2  do t_k ← 0
 3     while 2^{t_k} teilt k'
 4     do t_k ← t_k + 1
 5     f ← Gleichung 3.19
 6     x[1..4] ← REDMOND(f)
 7     for j ← 1 to 2^{t_k − 1}
 8     do for i ← 1 to 4
 9        do x[j + i] ← x[i] + j · 2^{(n/4) − t_k}
10     for i ← 1 to 2^{2+t_k}
11     do p, q = LSBFACT(N, x[i])
12        if N = pq
13          then return p, q
```

Abbildung 3.13: Algorithmus für den Angriff BDF1

3.19 geprüft werden müssen, wenn $k' \equiv 2 \mod 4$ ist, dann sind es höchstens 8 und so weiter. Da k' im Intervall $[0, \ldots, e]$ vollständig gesucht wird, müssen höchstens $4e \lceil \log_2 e \rceil$ Lösungen zu Gleichung 3.19 geprüft werden, bevor N faktorisiert werden kann. Daher ergibt sich eine Laufzeit für den Algorithmus von $O(T_{LSBFact}(n, m) \cdot e \log_2 e)$.

BM1

Satz 3.15 (BM1). *Sei N ein RSA-Modulus und seien $0 < \alpha, \epsilon < \frac{1}{2}$. Für alle außer einem $O\left(\frac{1}{N^\epsilon}\right)$-Teil der öffentlichen Exponenten e aus dem Interval $[3, N^\alpha]$ gilt: Sei d der geheime Schlüssel. Angenommen man erhält d_0, M mit $d = d_0 \mod M$ mit*

$$N^{\alpha+\frac{1}{2}+\epsilon} \leq M \leq 2N^{\alpha+\frac{1}{2}+\epsilon},$$

dann kann N in polynomieller Zeit faktorisiert werden.

Der in diesem Abschnitt vorgestellte Angriff von Blömer und May findet die Faktorisierung von N, wenn $e < N^{\frac{1}{2}}$ ist und dem Angreifer ausreichend viele der niederwertigsten Bits von d vorliegen. Wir betrachten erneut die Gleichung

$$ed - k\Phi(N) = 1.$$

Wenn d_0 der bekannte Teil von d ist, kann man auch schreiben $d_0 \equiv d \mod M$. Man

zerlegt d in den bekannten Teil d_0 und den unbekannten Teil d_1 und erhält $d = d_1M + d_0$. Durch Einsetzen und Umformen erhält man daraus

$$ed_1M + k(p + q - 1) - 1 + ed_0 = kN.$$

Daraus bildet man das bivariate Polynom

$$f_N(x, y) = eMx + y + ed_0,$$

mit einer Nullstelle $(x_0, y_0) = (d_1, k(p + q - 1) - 1)$ modulo N.

Nun wird gezeigt, dass mit dem bivariaten Verfahren nach Coppersmith für alle außer einer vernachlässigbar kleinen Anzahl von öffentlichen Exponenten e zwei *lineare* Polynome $f_1(x, y)$ und $f_2(x, y)$ in $\mathbb{Z}$ gefunden werden können, die die Nullstelle (x_0, y_0) haben. Darüber hinaus sind die Koeffizientenvektoren dieser Polynome linear unabhängig. Die Nullstelle (x_0, y_0) kann mit dem einfachen Eliminationsverfahren nach Gauß aus $f_1(x, y)$ und $f_2(x, y)$ berechnet werden. Das besondere daran ist also, dass in diesem Fall das bivariate Verfahren nicht heuristisch ist, da garantiert wird, dass (x_0, y_0) gefunden wird.

Zunächst werden die Schranken X und Y für x_0 bzw. y_0 gesucht. Um Y zu bestimmen betrachtet man die Ungleichung

$$k = \frac{ed - 1}{\Phi(N)} < e\frac{d}{\Phi(N)} < e \leq N^{\alpha}.$$

Da $d_1 \leq \frac{N}{M}$ ist, erhalten wir die Schranken $X := N^{\frac{1}{2}-\alpha-\epsilon}$ und $Y := 3N^{\frac{1}{2}+\alpha}$. Dazu beachte man, dass $p, q \leq 2\sqrt{N}$ und mindestens einer der Faktoren kleiner als $\sqrt{N}$ ist. Somit ist $p + q \leq 3\sqrt{N}$.

Nun soll $f_N(x, y)$ in ein Polynom $f(x, y)$ aus $\mathbb{Z}$ umgeformt werden, für das ebenfalls (x_0, y_0) eine Nullstelle ist. Hierfür bedienen wir uns des Satzes 3.4 nach Howgrave Graham. Dazu stellen wir zunächst die Hilfspolynome N und Nx auf, die modulo N beide das Nullpolynom darstellen. Daher hat auch jede Linearkombination $f = a_0N + a_1Nx + a_2f_N(x, y)$ die Nullstelle(x_0, y_0) modulo N. Die erste Bedingung von Satz 3.4 ist also erfüllt. Wir stellen aus den Polynomen die dreidimensionale Gitterasis

$$B = \begin{bmatrix} N & & \\ & NX & \\ ed_0 & eMX & Y \end{bmatrix}$$

auf. Um die zweite Bedingung zu erfüllen, suchen wir nach Linearkombinationen f mit

$\|f(xX, yY)\| \leq \frac{N}{\sqrt{3}}$. Wir nehmen nun an, dass das LLL-Verfahren zwei Vektoren mit kleinerer Norm als $\frac{N}{\sqrt{3}}$ findet. Da diese durch Linearkombination der Basisvektoren entstehen, lassen sie sich auch schreiben als $(a_0, a_1, a_2)B$ und $(b_0, b_1, b_2)B$ und es gilt

$$a_0N + a_1Nx_0 + a_2f_N(x_0, y_0) = 0$$

$$b_0N + b_1Nx_0 + b_2f_N(x_0, y_0) = 0.$$

Wir wissen, dass $f_N(x_0, y_0) = kN$ ist, und erhalten daher das vereinfachte lineare Gleichungssystem:

$$a_1x_0 + a_2k = -a_0$$

$$b_1x_0 + b_2k = -b_0$$

Jetzt müssen noch zwei Dinge gezeigt werden. Zum einen, dass (a_1, a_2) und (b_1, b_2) linear unabhängig sind und zum anderen, dass das LLL-Verfahren tatsächlich zwei Vektoren mit ausreichend kleiner Norm findet.

Ersteres kann leicht durch einen indirekten Beweis gezeigt werden: Es ist bekannt, dass $(a_0, a_1, a_2), (b_0b_{,1}, b_2) \in \mathbb{Z}$ linear unabhängig sind. Gibt es ein $c \in \mathbb{Q}$ so, dass $c \cdot (a_0, a_1) = (b_0, b_1)$ gilt, dann gilt auch die Folgerung

$$c \cdot (a_1x_0, a_2k) = (b_1x_0, b_2k) \Rightarrow c \cdot (a_1x_0 + a_2k) = b_1x_0 + b_2k.$$

Daraus folgt, dass $c \cdot a_0 = b_0$ und damit $c \cdot (a_0, a_1, a_2) = (b_0, b_1, b_2)$. Das steht jedoch im Widerspruch zur linearen Unabhängigkeit von (a_0, a_1, a_2) und (b_0, b_1, b_2). Daher müssen (a_1, a_2) und (b_1, b_2) linear unabhängig sein.

Man erhält daher x_0, k als einzige Lösung des Gleichungssystems durch das Gaußsche Eliminationsverfahren. Damit lässt sich auch $y_0 = kN - eMx_0 - ed_0$ berechnen. Man berechnet dann $\frac{y_0+1}{k} = p + q - 1$ und erhält genug Informationen, um N zu faktorisieren.

Bleibt noch zu zeigen, dass das Gitter zwei Vektoren enthält, deren Norm kleiner als $\frac{N}{\sqrt{3}}$ ist. Dazu stellen Blömer und May das folgende Lemma auf:

Lemma 3.5. *Seien N, α, ϵ und M gegeben, wie in Satz 3.15. Dann gilt für alle außer $O(N^{\alpha-\epsilon})$ e's aus dem Intervall $[3, N^{\alpha}]$: Sei $X := N^{\frac{1}{2}-\alpha-\epsilon}$ und $Y := 3N^{\frac{1}{2}+\alpha}$, dann enthält das oben definierte Gitter zwei linear unabhängige Vektoren mit kleinerer Norm als $\frac{N}{\sqrt{3}}$.*

Der genaue Beweis ist für die Anwendung des Verfahrens uninteressant und kann in [May03, S.113] oder [JB03, S.14] nachgelesen werden.

Wir erhalten für den Angriff den Algorithmus aus Abbildung 3.14. In Zeile 2 des Algorithmus wird die LLL-Funktion aufgerufen. Dabei wird eine erweiterte Variante ver-

BM1(e, N, d_0, M)

1 $B = \begin{bmatrix} N & & \\ & NX & \\ ed_0 & eMX & Y \end{bmatrix}$
2 $B', U \leftarrow \mathrm{LLL}(B)$
3 $(a_0, a_1, a_3) \leftarrow U[1]$
4 $(b_0, b_1, b_3) \leftarrow U[2]$
5 **if** $(\|B'[1]\| > \frac{N}{\sqrt{3}}$ or $\|B'[2]\| > \frac{N}{\sqrt{3}})$
6 **then**
7 **error** "keine ausreichend kurzen Vektoren gefunden"
8 **else**
9 $f_1(x_0, k) \leftarrow a_1 x_0 + a_2 k + a_0$
10 $f_2(x_0, k) \leftarrow b_1 x_0 + b_2 k + b_0$
11 $x_0, k \leftarrow \textsc{Gaussian}(f_1, f_2)$
12 $y_0 \leftarrow kN - eMx_0 - ed_0$
13 $p, q \leftarrow \textsc{solve}(x^2 - (\frac{y_0+1}{k})x - N)$
14 **return** p, q

Abbildung 3.14: Algorithmus für den Angriff BM1

wendet, die neben der reduzierten Matrix B' auch die Transformationsmatrix U liefert mit $U \cdot B = B'$. Die Zeilen von U stellen dann genau die gesuchten Vektoren (a_0, a_1, a_3) und (b_0, b_1, b_3) dar. Nun wird in Zeile 5 geprüft, ob hinreichend kurze Vektoren gefunden wurden und wenn dies der Fall ist, werden durch das Gaußsche Eliminationsverfahren x_0 und k als einzige Lösung von (f_1, f_2) gefunden. Anschließend wird y_0 berechnet und durch das Lösen einer quadratischen Gleichung p und q bestimmt und zurückgegeben. Da jeder dieser Schritte in polynomieller Zeit durchgeführt werden kann, ergibt sich insgesamt auch eine polynomielle Laufzeit.

BM2

Satz 3.16 (BM2). *Unter der Annahme, dass das bivariate Verfahren zwei algebraisch unabhängige Polynome erzeugt gilt:*

Für jedes $\epsilon > 0$ existiert ein N_0, so, dass für jedes $N \geq N_0$ gilt: Sei (N, e) ein öffentlicher RSA-Schlüssel mit $\alpha := \log_N(e) \leq \frac{7}{8}$. Sei d der geheime Schlüssel. Angenommen man erhält d_0 und M so, dass $d = d_0 \mod M$ mit

$$M \geq N^{\frac{1}{6} + \frac{1}{3}\sqrt{1+6\alpha} + \epsilon}.$$

Dann kann N in polynomieller Zeit faktorisiert werden.

Wir betrachten die Gleichung $ed - 1 = k\Phi(N)$. d_0 und M sind wie im Satz 3.16 gegeben, dann können wir d schreiben als $d = d_1M + d_0$. Eingesetzt ergibt sich

$$k(N - (p + q - 1)) - ed_0 + 1 = eMd_1.$$

Wir reduzieren modulo eM und erhalten das folgende Polynom in zwei Unbekannten:

$$f_{eM}(y, z) = y(N - z) - ed_0 + 1$$

Es hat die Nullstelle $(y_0, z_0) = (k, p+q-1)$ modulo eM. Da $k < e \leq N^\alpha$ und $p+q \leq 3N^{\frac{1}{2}}$ gilt, ergeben sich für y_0 und z_0 die Schranken $Y := N^\alpha$ und $Z := 3N^{\frac{1}{2}}$. Es werden wieder zwei Parameter t und m, deren Berechnung später erläutert wird, festgelegt und dann die Polynome

$$g_{i,j} := y^j(eM)^i f_{eM}^{m-i} \text{ für } i = 0, \ldots, m; j = 0, \ldots, i; \tag{3.21}$$

$$h_{i,j} := z^j(eM)^i f_{eM}^{m-i} \text{ für } i = 0, \ldots, m; j = 1, \ldots, t;. \tag{3.22}$$

definiert. Da alle den Term $(eM)^i f_{eM}^{m-i}$ beinhalten, haben alle ganzzahligen Linearkombinationen dieser Polynome die Nullstelle (y_0, z_0) modulo $(eM)^m$ und erfüllen daher die erste Bedingung des Satzes 3.5 für das bivariate Verfahren nach Coppersmith. Um die zweite Bedingung zu erfüllen, suchen wir nun nach Linearkombinationen der Polynome $g_{i,j}(yY, zZ)$ und $h_{i,j}(yY, zZ)$, deren Norm kleiner ist als $\frac{(eM)^m}{\sqrt{\dim(L(m))}}$. Wir zeigen nun, dass mittels des LLL-Algorithmus in dem Gitter, das von den Koeffizientenvektoren von $g_{i,j}(yY, zZ)$ und $h_{i,j}(yY, zZ)$ aufgespannt wird, immer zwei ausreichend kurze Vektoren gefunden werden können. Sei $B(m)$ die von den Koeffizientenvektoren gebildete Basismatrix mit der Dimension w. Seien weiter v_1 und v_2 die ersten beiden Basisvektoren der LLL-reduzierten Basis $B'(m)$. Aus Satz 3.2 wissen wir, dass

$$\|v_1\| \leq \|v2\| \leq 2^{\frac{n}{4}} \det(B(m))^{\frac{1}{n-1}}.$$

Daher müssen wir die Bedingung

$$2^{\frac{n}{4}} \det(B(m))^{\frac{1}{n-1}} < \frac{(eM)^m}{\sqrt{w}}$$

erfüllen. Vernachlässigen wir alle Terme, die nicht von N abhängen, vereinfacht sich diese

Bedingung zu $\det(B(m)) < (eM)^{m(n-1)}$. Wir setzen $t = \tau m$ und berechnen

$$\det(B(m)) = \left((eMX)^{3\tau+1} Z^{3\tau^2+3\tau+1}\right)^{\frac{1}{6}m^3(1+o(1))}.$$

Nun setzen wir die Schranken $Y = N^{\alpha}$ und $Z = 3N^{\frac{1}{2}}$ ein und erhalten die Bedingung

$$N^{\frac{1}{12}m^3(3\tau^2+3\tau(2\alpha+1)+4\alpha+1)(1+o(1))} \leq (eM)^{m(n-1)(3\tau+2)(\frac{1}{6}m^3(1+o(1)))}.$$

Wir berechnen $w = (\tau + \frac{1}{2})m^2$ und setzen $eM \geq N^{\alpha+\frac{1}{6}+\frac{1}{3}\sqrt{1+6\alpha}+\epsilon}$. Wir vernachlässigen erneut Terme kleinerer Ordnung und erhalten als neue Bedingung

$$9\tau + 6(\alpha + \tau) - 2\sqrt{1+6\alpha}(1+3\tau) + 2 \leq 0.$$

Die linke Seite ist minimal für $\tau = \frac{1}{3}(\sqrt{1+6\alpha} - 1)$ und für diese Wahl von τ ist die Bedingung erfüllt.

Die genaue Bestimmung der Parameter m und t verläuft ganz analog wie in Abschnitt 3.3 gezeigt. Man stellt aus der Formel für die Determinante und der Bedingung aus Satz 3.5 eine Relation in e, ϵ, m und t auf. Dann wird e aus der Gleichung eliminiert und t optimal in Abhängigkeit von ϵ und m bestimmt. Setzt man diesen Term für t ein, erhält man die gesuchte Relation von ϵ und m.

Abbildung 3.15 zeigt den Angriff in Pseudocode. Als Eingabe wird der öffentliche Schlüssel (e, N) und die LSB von d benötigt. Dazu wird zusätzlich zu d_0 ein M angegeben, sodass $d_0 \equiv d \mod M$ ist. Im Fall, dass die j LSB von d bekannt sind, ergibt sich für $M = 2^j$. Der Algorithmus berechnet zunächst α und ϵ nach Satz 3.16. Je größer ϵ ist desto mehr ist von d, über die Mindestmenge hinaus, bekannt. Deshalb ist auch einleuchtend, dass ϵ umgekehrt proportional auf die Laufzeit einwirkt. Anschließend werden die Schranken Y und Z berechnet und das Polynom $f_{eM}(y, z)$ aufgestellt. In BUILDLATTICE werden die Polynome $g_{i,j}(yY, zZ)$ und $h_{i,j}(yY, zZ)$ aufgestellt und ihre Koeffizienten in die Matrix eingetragen. Nach der LLL-Reduktion werden aus den ersten beiden Vektoren die entsprechenden Polynome aufgestellt. Dazu wird der Koeffizient zu y^i, z^j aus dem entsprechenden Vektorelement berechnet, indem es durch $Y^i Z^i$ geteilt wird. Danach wird die Resultante der beiden gefundenen Polynome berechnet und deren Nullstellen gesucht. Für jede Nullstelle werden p' und q' als Lösungen einer quadratischen Gleichung bestimmt. Wenn $z_0 = p+q-1$ ist, dann ist $p' = p$ und $q' = q$ und sie werden ausgegeben.

```
BM2(e, N, d_0, M)
 1  α ← LOG_N(e)
 2  needed ← N^{1/6 + 1/3 √(1+6α)}
 3  ε ← LOG_N(M/needed)
 4  m ← FINDM(ε)
 5  t ← FINDT(m)
 6  Y ← N^α
 7  Z ← 3N^{1/2}
 8  f_eM(y, z) ← y(N − z) − ed_0 + 1
 9  B ← BUILDLATTICE(f_eM, M, e, d_0, m, t, Y, Z)
10  B' ← LLL(B)
11  f_1(y, z) ← BUILDPOLY(B'[1], Y, Z)
12  f_2(y, z) ← BUILDPOLY(B'[2], Y, Z)
13  r_12(z) ← RESULTANT(f_1, f_2, y)
14  roots[ ] ← FINDROOTS(r_12(z))
15  for each z_0 in roots[ ]
16  do
17      p', q' ← SOLVE(x^2 − (z_0 + 1)x − N)
18      if N = p · q
19        then
20              return p', q'
```

Abbildung 3.15: Algorithmus für den Angriff BM2

3.5.3 Angriffe bei bekannten MSB von d

BDF2

Satz 3.17 (BDF2). *Sei $N = pq$ ein n-bit RSA-Modulus. Wenn e eine t-bit Primzahl aus dem Intervall $\left[N^{\frac{1}{4}}, N^{\frac{1}{2}}\right]$ ist, so kann N in polynomieller Zeit abhängig von $\log N$ faktorisiert werden, wenn die*

$$t \text{ MSBs von } d$$

bekannt sind.

Wir betrachten die Gleichung $ed+k\Phi(N) = 1 \mod N$. Grundlage für diesen Angriff ist es, k zu bestimmen. Da k jedoch beliebig im Intervall $[1, \ldots, e]$ liegt, kann es nicht durch bloßes Ausprobieren gefunden werden. Wir werden deshalb ein Verfahren vorstellen, mit dem ein konstantes kleines Intervall bestimmt werden kann, in dem k liegt. Voraussetzung dafür ist, dass eine ausreichend gute Näherung d_0 für d vorhanden ist. Dazu verwenden wird das folgende Lemma von Boneh, Durfee und Frankel:

Lemma 3.6. *Sei d_0 gegeben, sodass die folgenden Bedingungen gelten:*

1. $|e(d - d_0)| < c_1 N$ *und*
2. $ed_0 < c_2 N^{\frac{2}{3}}$.

Dann ist das eindeutig bestimmte k mit $ed + k\Phi(N) = 1 \mod N$ eine Ganzzahl im Intervall $[\tilde{k} - \Delta, \tilde{k} + \Delta]$, wobei $\tilde{k} = (ed_0 - 1)/N$ und $\Delta = 8c_2 + 2c_1$ ist.

Beweis 3.2. *Sei $\tilde{k} = (ed_0 - 1)/N$. dann gilt:*

$$|\tilde{k} - k| = \left|(ed_0 - 1)\left(\frac{1}{\Phi(N)} - \frac{1}{N}\right) + \frac{e(d - d_0)}{\Phi(N)}\right| < c_2 N^{3/2}\left(\frac{N - \Phi(N)}{\Phi(N)N}\right) + c_1\frac{N}{\Phi(N)}$$

Da $N - \Phi(N) < 4\sqrt{N}$ und $\Phi(N) > N/2$ folgt daraus

$$|\tilde{k} - k| < 8c_2 + 2_c 1.$$

Daher liegt k wie gefordert im Intervall $[\tilde{k} - \Delta, \ldots, \tilde{k} + \Delta]$.

□

Mit den t höchstwertigen Bits von d kann man ein d_0 konstruieren, für das $|d - d_0| < 2^{n-t}$ gilt. Nun kann mit Lemma 3.6 k bestimmt werden. Durch die Beschränkungen für e aus Satz 3.17 ist Bedingung 1 efüllt mit $c_1 = 2$. Da $d_0 < N$ ist, gilt Bedingung 2 mit $c_2 = 2$. Es ergibt sich $\Delta = 8c_2 + 2c_1 = 20$ und somit findet man k in einem bekannten Intervall der Größe 40. Nun wird die Gleichung $ed + k(N + 1 - p + q) = 1$ betrachtet. Wir setzen $s = p + q$ ein und erhalten $ed + k(N + 1 - s) = 1$. Wenn diese Gleichung modulo e reduziert wird, erhält man

$$s \equiv k^{-1} - N - 1 \mod e.$$

Da k und e teilerfremd sind, ist k^{-1} wohldefiniert. Für jeden Kandidaten k' für k aus dem berechneten Intervall führt man nun die folgenden 3 Schritte aus:

1. Man berechnet einen Kandidaten s' mit

$$s' \equiv k'^{-1} - N - 1.$$

2. Man berechnet ein $p' \mod e$ als Nullstelle für x in der quadratischen Gleichung

$$x^2 - s'x + N \equiv 0 \mod e.$$

Das kann effizient berechnet werden, da e prim ist und wenn $s' \equiv s \equiv p+q \mod e$ ist, dann ist $p' \equiv p \mod e$

3. Mit der Funktion LSBFact aus Abschnitt 3.4 wird nun p aus $p \mod e$ berechnet. Das ist möglich, da $t \geq 2^{n/4}$.

```
BDF2(e, N, d_0)
1  k̃ ← (ed_0 − 1)/N
2  for k' ← (k̃ − 20) to (k̃ + 20)
3  do
4       s' ← (INV(k', e) − N − 1) mod e
5       p_e ← SOLVE(x^2 − s'x + N = 0 mod e)
6       p, q ← LSBFact(N, p_e)
7       if N = pq
8          then return p, q
```

Abbildung 3.16: Algorithmus für den Angriff BDF2

Der Algorithmus in Pseudocode ist in Abbildung 3.16 dargestellt. Die meisten Schritte sind elementar. Zunächst wird eine Abschätzung $\tilde{k}$ für k berechnet und ein Intervall gebildet, in dem k zu finden ist. Dieses Intervall wird nun durchlaufen und für jeden Kandidaten k' wird ein s' berechnet. Der eigentlich schwierige Teil des Algorithmus ist die Funktion Solve, in der die Nullstellen einer modularen quadratischen Gleichung gesucht werden. Da e prim ist, hat die Gleichung genau zwei Lösungen und diese lassen sich mit polynomiellem Aufwand finden. Die Lösungen entsprechen p und q in den t niederwertigsten Bits, da $p' \equiv p \mod e$ und $p' \equiv p \mod e$. Nun wird die LSBFact Funktion aufgerufen und bei erfolgreicher Faktorisierung wird das Ergebnis zurückgegeben. Die Schleife läuft über ein konstantes Intervall und als Laufzeit ergibt sich $O(T_{LSBFact}(n,t))$.

BDF3

Satz 3.18 (BDF3). *Sei $N = pq$ ein n-bit RSA-Modulus. Sei e ein t-bit Produkt von r verschiedenen Primzahlen und e sei aus dem Intervall $\left[N^{\frac{1}{4}}, N^{\frac{1}{2}}\right]$. Wenn die Faktorisierung von e bekannt ist, so kann N in polynomieller Zeit abhängig von $\log N$ und 2^r faktorisiert werden, wenn die*

t MSBs von d

bekannt sind.

```
BDF3(e, N, d_0)
 1  k̃ ← (ed_0 − 1)/N
 2  for k' ← (k̃ − 20) to (k̃ + 20)
 3  do
 4      if GCD(e, k') = 1
 5        then
 6              s' ← (INV(k', e) − N − 1)  mod  e
 7              p_e[ ] ← SOLVE(x^2 − s'x + N = 0  mod  e)
 8              for  each i in p_e[ ]
 9              do
10                  p, q ← LSBFact(N, i)
11                  if N = pq
12                    then return p, q
```

Abbildung 3.17: Algorithmus für den Angriff BDF3

Dieser Angriff funktioniert im Wesentlichen so, wie der letzte. Der einzige Unterschied ist, dass e keine Primzahl ist. Das hat zur Folge, dass die quadratische Gleichung, die in jedem Schritt gelöst werden muss, nicht mehr nur 2 Lösungen hat. Tatsächlich ist es so, dass wenn e eine Zahl mit r verschiedenen Primfaktoren ist, die quadratische Gleichung bis zu 2^r Lösungen besitzt. Für jede dieser Lösungen muss die Funktion LSBFact aufgerufen werden, wodurch sich als neue Laufzeit $O(T_{LSBFact}(n,t) \cdot 2^r)$ ergibt. Darüber hinaus ist nicht mehr gewährleistet, dass k' invertierbar ist. Da jedoch e und k teilerfremd sein müssen, müssen nur diejenigen k' geprüft werden, die teilerfremd zu e sind.

BDF4

Satz 3.19 (BDF4). *Sei $N = pq$ ein n-bit RSA-Modulus. Sei $e \leq N^{\frac{1}{2}}$ eine t-bit Zahl von unbekannter Faktorisierung. Sei weiter $d > \epsilon N$ für ein gegebenes ϵ, so kann N in polynomieller Zeit abhängig von $\log N$ und $\frac{1}{\epsilon}$ faktorisiert werden, wenn die*

$$n - t \text{ MSBs von } d$$

bekannt sind.

Dieser Angriff beschäftigt sich mit dem Fall, wenn die Faktorisierung von e unbekannt ist. Er funktioniert, wenn e eine t Bit Zahl kleiner als $N^{\frac{1}{2}}$ ist und die $n-t$ höchstwertigen Bits von d bekannt sind. Als weitere Einschränkung gilt, dass k nicht signifikant kleiner ist als e, wir also schreiben können $k > \epsilon \cdot e$ mit einem kleinen $\epsilon > 0$.

Mit den $n-t$ MSB von d können wir wieder ein d_0 konstruieren, sodass $0 \leq d-d_0 < 2^t$ gilt. Da $e < 2^{n/2}$ ist, können wir Lemma 3.6 verwenden, um mit d_0 k auf ein Intervall konstanter Größe zu beschränken. Nun werden für jeden Kandidaten k' die Folgenden Schritte durchgeführt:

1. Berechne $d_1 \equiv e^{-1} \mod k'$. Das ist möglich, da e und k teilerfremd sind. Deshalb müssen auch nur solche k' betrachtet werden, die teilerfremd zu e sind. Da $ed - k\Phi(N) = 1$ ist, wissen wir, dass $d_1 \equiv d \mod k'$ ist, wenn $k' = k$ ist.

2. Laut Annahme gilt $k' > \epsilon 2^t$. Zu diesem Zeitpunkt kennen wir $d \mod k'$ und die $n-t$ MSB von d. Die restlichen Bits von d werden nun durch erschöpfende Suche gesucht. Wir stellen die Gleichung $d = k'd_2 + d_1$ auf, dann ist $d_2 = d_0/k' + (d - d_0)/k' - d_1/k'$. Der einzige unbekannte Term dieser Summe ist $v = (d-d_0)/k'$. Da $k' > \epsilon 2^t$ ist, wissen wir, dass $v = (d-d_0)/k' < 1/\epsilon$ sein muss. Um v zu finden, probieren wir nun alle möglichen Kandidaten v' im Intervall $\{0, \ldots, 1/\epsilon\}$ aus. Zu jedem Kandidatenpaar (k', v') berechnen wir den zugehörigen Wert d und testen ihn.

3. Wenn die korrekten Werte k' und v' gefunden wurden, ist auch d gefunden. Jedes Paar (k', v') zu testen dauert $O(n^3)$ Zeiteinheiten und es gibt $O(1/\epsilon)$ Kandidatenpaare zu testen.

```
BDF4(e, N, d_0, ε)
 1  k̃ ← (e d_0 − 1)/N
 2  for k' ← (k̃ − 20) to (k̃ + 20)
 3  do
 4      if GCD(e, k') = 1
 5        then
 6              d_1 ← INV(e, k')
 7              for v = 0 to (1/ε)
 8              do
 9                  d_2 ← d_0/k' + v − d_1/k'
10                  d ← k' d_2 + d_1
11                  if TEST(N, e, d)
12                    then
13                          return d
```

Abbildung 3.18: Algorithmus für den Angriff BDF4

Abbildung 3.18 zeigt den Algorithmus zu diesem Angriff. Das Intervall, in dem k gesucht wird, ist konstant. Also hängt die Anzahl der Schleifendurchläufe nur von $1/\epsilon$ ab. Die Funktion TEST läuft in $O(n^3)$, sodass sich als Gesamtlaufzeit $O(n^3/\epsilon)$ ergibt.

BM3

Satz 3.20 (BM3). *Unter der Annahme, dass das trivariate Verfahren nach Coppersmith drei algebraisch unabhängige Polynome erzeugt gilt:*

Zu gegebenem $\epsilon > 0$ existiert ein N_0 so, dass für jedes $N > N_0$ das Folgende gilt: Sei (N, e) ein öffentlicher RSA-Schlüssel mit $\alpha := \log_N(e) \in \left[N^{\frac{1}{2}}, N^{\frac{\sqrt{6}-1}{2}}\right]$. Angenommen man erhält eine Näherung $\tilde{d}$ von d mit

$$|d - \tilde{d}| \leq N^{\frac{1}{8}\left(5-2\alpha-\sqrt{36\alpha^2+12\alpha-15}\right)-\epsilon},$$

dann kann N in polynomieller Zeit in $\log N$ faktorisiert werden.

Während die zuvor genannten Angriffe für bekannte MSB von d alle lediglich bis $e < N^{\frac{1}{2}}$ funktionieren, präsentieren Blömer und May mit diesem Angriff erstmals einen Angriff auf RSA mit $e \in [N^{\frac{1}{2}}, N^{\frac{\sqrt{6}-1}{2}}]$. Dabei wird das Verfahren nach Coppersmith im trivariaten Fall verwendet. Deswegen muss dieses Verfahren auch als Heuristik angesehen werden. Die Annahme, dass die gefundenen Polynome algebraisch unabhängig sind, wurde jedoch in allen Experimenten von Blömer und May bestätigt, es wurde kein Gegenbeispiel gefunden.

Das Verfahren geht erneut von der Gleichung

$$ed - 1 = k\Phi(N) \tag{3.23}$$

aus.

Im Unterschied zu den von Boneh, Durfee und Frankel vorgestellten Verfahren, wird hier nicht das exakte k benötigt, sondern vielmehr mittels der Näherung $\tilde{d}$ von d eine Näherung $\tilde{k}$ von k berechnet. Wir gehen davon aus, dass $|d - \tilde{d}| < N^{\delta}$ mit $\delta = \frac{1}{8}\left(5 - 2\alpha - \sqrt{36\alpha^2 + 12\alpha - 15}\right) - \epsilon$. Daraus berechnet man die Abschätzung $\tilde{k}$ nun mit $\tilde{k} := \frac{e\tilde{d}-1}{N+1}$ und erhält

$$|k - \tilde{k}| = \left|\frac{ed-1}{\Phi(N)} - \frac{e\tilde{d}-1}{N+1}\right|.$$

Wir bringen die rechte Seite auf einen gemeinsam Nenner und erhalten

$$\left| \frac{(ed-1)(N+1) - (e\tilde{d}-1)(N+1-(p+q))}{\Phi(N)(N+1)} \right|.$$

Da $(p+q) < 3N^{\frac{1}{2}}$ ist, erhält man als Abschätzung:

$$|k - \tilde{k}| \leq \left| \frac{e(d-\tilde{d})}{\Phi(N)} \right| + \left| \frac{(p+q)(e\tilde{d}-1)}{\Phi(N)(N+1)} \right| \leq \frac{e}{\Phi(N)}(N^{\delta} + 3N^{-\frac{1}{2}}\tilde{d})$$

Der schwierigere Fall ist hier der, wenn der Term $N^{-\frac{1}{2}}\tilde{d}$ dominierend gegenüber N^{δ} ist. Gehen wir zunächst vom Gegenteil aus, also $N^{\delta} > N^{-\frac{1}{2}}\tilde{d}$. Vernachlässigt man Terme kleinerer Ordnung, ergibt sich für $|k - \tilde{k}|$ als obere Schranke $N^{\alpha+\delta-1}$. Also kann k genau bestimmt werden, wenn $\alpha + \delta - 1 \leq 0$ ist. Diese Bedingung folgt jedoch aus den Voraussetzungen von Satz 3.20. Wenn k bekannt ist, kann man $p + q = N + 1 + k^{-1} \mod e$ berechnen und da $e \geq N^{\frac{1}{2}}$ ist, erhält man $p + q$ damit in $\mathbb{Z}$ und nicht nur modulo e. Das führt direkt zur Faktorisierung von N.

Daher gehen wir im Folgenden von dem Fall aus, dass $N^{-\frac{1}{2}}\tilde{d} \geq N^{\delta}$ ist und erhalten $4N^{\alpha-\frac{1}{2}}$ als Schranke für $|k - \tilde{k}|$. Wir definieren nun d_0 und k_0 mit $d_0 = d - \tilde{d}$ und $k_0 = k - \tilde{k}$. Dann wird Gleichung 3.23 zu:

$$e(\tilde{d} + d_0) - 1 = (\tilde{k} + k_0)\Phi(N).$$

Das kann umgeschrieben werden zu

$$ed_0 + (\tilde{k} + k_0)(p + q - 1) + e\tilde{d} - 1 = (\tilde{k} + k_0)N.$$

Wir reduzieren modulo N und erhalten das folgende Polynom in drei Unbekannten:

$$f_N(x, y, z) = ex + (\tilde{k} + y)z + e\tilde{d} - 1$$

Man erkennt, dass dieses Polynom die Nullstelle $(x_0, y_0, z_0) = (d_0, k_0, p + q - 1)$ modulo N hat. Das weitere Vorgehen ist genau wie im univariaten und bivariaten Verfahren. Zunächst beschränken wir die Nullstelle, dann stellen wir weitere Polynome $\mod N$ auf, die die gleiche Nullstelle besitzen. Die Koeffizientenvektoren dieser Polynome bilden dann die erneut die Basis eines Gitters. In diesem Gitter wird mit dem LLL-Algorithmus nach drei kurzen Vektoren gesucht, aus denen drei Polynome aufgestellt werden. Mit dem Satz 3.4, der sich leicht auf den trivariaten Fall erweitern lässt, zeigt man dann, dass die

gefundenen Polynome auch in $\mathbb{Z}$ die Nullstelle (x_0, y_0, z_0) besitzen. Als Schranken für x_0, y_0 und z_0 erhalten wir $X := N^\delta$, $Y := 4N^{\alpha-\frac{1}{2}}$ und $Z := 3N^{\frac{1}{2}}$.

Jetzt bestimmen wir erneut die Parameter m und t und definieren die Polynome

$$g_{i,j,k} := x^{j-k} z^k N^i f_N^{m-i} \text{ für } i = 0, \ldots, m; j = 0, \ldots, i; k = 0, \ldots, j \tag{3.24}$$

$$h_{i,j,k} := x^j y^k N^i f_N^{m-i} \text{ für } i = 0, \ldots, m; j = 0, \ldots, i; k = 1, \ldots, t. \tag{3.25}$$

Wie leicht einsichtig ist, haben alle diese Polynome eine Nullstelle bei (x_0, y_0, z_0) modulo N. Damit erfüllen sie die erste Bedingung der trivariaten Version von Satz 3.4. Dies gilt ebenso für alle Linearkombinationen dieser Polynome. Wir erstellen nun aus den Koeffizienten von $g_{i,j,k}(xX, yY, zZ)$ und $h_{i,j,k}(xX, yY, zZ)$ eine Gitterbasis $B(m)$ und verwenden den LLL-Algorithmus, um in dem aufgespannten Gitter $L(m)$ nach kurzen Vektoren zu suchen.

Beispiel 3.7. *Wir geben hier als Beispiel die Basis $B(1)$ an. Es gilt also $m = 1$ und es ergibt sich t optimal zu $t = 1$. Wir erhalten aus den Koeffizientenvektoren $g_{1,0,0}, g_{1,1,0}, g_{1,1,1}$ und $g_{0,0,0}$ sowie $h_{1,0,1}, h_{1,1,1}$ und $h_{0,0,1}$ die Zeilenvektoren der Basis $B(1)$ als:*

$$B(1) = \left[\begin{array}{cccccccc} N & & & & & & & \\ & NX & & & & & & \\ & & NZ & & & & & \\ e\tilde{d}-1 & eX & \tilde{k}Z & YZ & & & & \\ \hline & & & & NY & & & \\ & & & & & NXY & & \\ & & & \tilde{k}YZ & (e\tilde{d}-1)Y & eXY & Y^2Z & \end{array}\right]$$

Es bleibt noch zu zeigen, dass tatsächlich drei Vektoren gefunden werden, die kleiner als $N^m/\sqrt{w}$ sind, w bezeichnet dabei die Dimension des Gitters. Aus Satz 3.2 wissen wir, dass für die ersten drei Vektoren v_1, v_2, v_3 der LLL-reduzierten Basis gilt:

$$\|v_1\| \leq \|v_2\| \leq \|v_3\| \leq 2^{\frac{w(w-1)}{4(w-2)}} \det(L(m))^{\frac{1}{w-2}}.$$

Damit $\|v_3\| \leq N^m/\sqrt{w}$ gilt, muss die Bedingung

$$\det(L) < cN^{m(w-2)}$$

erfüllt sein. Dabei hängt $c = 2^{-\frac{w(w-1)}{4}} w^{-\frac{w-2}{2}}$ nicht von N ab und trägt daher zu dem

Fehlerterm ϵ bei. Sei $t := \tau m$, dann ergibt sich die Determinante von $L(m)$ zu

$$\det(L(m)) = \left(N^{8\tau+3} X^{4\tau+1} Y^{6\tau^2+4\tau+1} Z^{4\tau+2}\right)^{\frac{1}{24}m^4(1+o(1))}.$$

Wir setzen die Schranken $X := N^{\delta}$, $Y := 4N^{\alpha-\frac{1}{2}}$ und $Z := 3N^{\frac{1}{2}}$ ein und erhalten

$$\det(L(m)) = N^{\frac{1}{24}m^4(3\tau^2(2\alpha-1)+4\tau(\delta+\alpha+2)+\delta+\alpha+\frac{7}{2})(1+o(1))}.$$

Nun wird $w = \frac{1}{24}m^3(12\tau+4)(1+o(1))$ berechnet und unter Nichtberücksichtigung von Termen kleinerer Ordnung kann die Bedingung zu

$$3\tau^2(2\alpha-1) + 4\tau(\delta+\alpha-1) + \delta + \alpha - \frac{1}{2} < 0$$

vereinfacht werden. Wir suchen für die linke Seite ein minimales τ und erhalten $\tau = \frac{2}{3}\frac{1-\delta-\alpha}{2\alpha-1}$. Wenn wir diesen Wert einsetzen und nach δ auflösen, erhalten wir die gesuchte Bedingung

$$\delta \leq \frac{1}{8}\left(5 - 2\alpha - \sqrt{36\alpha^2 + 12\alpha - 15}\right).$$

Die Bestimmung von m und t verläuft analog zu dem Vorgehen aus Abschnitt 3.3, man erhält erneut eine Formel für m in Abhängigkeit von ϵ und kann t optimal in Abhängigkeit von ϵ und m bestimmen.

Abbildung 3.19 zeigt den Algorithmus in Pseudocode. Der Anwender übergibt der Funktion die Parameter e, N, d_0 und ϵ. Das ϵ bestimmt, wie viel mehr als die Mindestmenge an Bits zur Verfügung steht und daher ist dieser Parameter sehr entscheidend für die Laufzeit der Funktion. Zunächst wird die Näherung $\tilde{k}$ bestimmt. Dann wird abhängig von ϵ der Parameter m berechnet. Zu dem so gefundenen m wird nun t optimal bestimmt. Nachdem die Schranken X, Y, Z berechnet wurden, wird die Gitterbasis aufgestellt und reduziert. Um aus der erhaltenen reduzierten Basis die Polynome $f_1(x,y,z), f_2(x,y,z)$ und $f_3(x,y,z)$ zu erzeugen, müssen die entsprechenden Potenzen von X, Y und Z wieder herausdividiert werden. Anschließend werden aus den drei Polynomen in drei Unbekannten durch Resultantenbildung zunächst zwei Polynome in zwei Unbekannten erzeugt. Von diesen zwei Polynomen wird dann erneut die Resultante gebildet, die nur noch eine Unbekannte enthält. Sollte diese Resultante gefunden werden, was bekanntlich nicht garantiert ist, so hat sie eine Nullstelle in $z_0 = p + q - 1$. Die zwei Lösungen der Gleichung $x^2 - (z_0+1)x - N$ liefern demnach p und q. Wenn die richtige Nullstelle identifiziert ist, werden p und q ausgegeben.

BM3$(e, N, \tilde{d}, \epsilon)$

1 $\tilde{k} \leftarrow \frac{e\tilde{d}-1}{N+1}$
2 $m \leftarrow \text{FINDM}(\epsilon)$
3 $t \leftarrow \text{FINDT}(\epsilon, m)$
4 $\alpha \leftarrow \text{LOG}(e, N)$
5 $\delta \leftarrow \frac{1}{8}\left(5 - 2\alpha - \sqrt{36\alpha^2 + 12\alpha - 15}\right) - \epsilon$
6 $X \leftarrow N^{\delta}$
7 $Y \leftarrow 4N^{\alpha-\frac{1}{2}}$
8 $Z \leftarrow 3N^{\frac{1}{2}}$
9 $B \leftarrow \text{BUILDLATTICE}(N, e, \tilde{d}, \tilde{k}, \epsilon, m, t, X, Y, Z)$
10 $B' \leftarrow \text{LLL}(B)$
11 $f_1(x, y, z) \leftarrow \text{BUILDPOLY}(B'[1], X, Y, Z)$
12 $f_2(x, y, z) \leftarrow \text{BUILDPOLY}(B'[2], X, Y, Z)$
13 $f_3(x, y, z) \leftarrow \text{BUILDPOLY}(B'[3], X, Y, Z)$
14 $r_{12}(x, z) \leftarrow \text{RESULTANT}(f_1, f_2, y)$
15 $r_{23}(x, z) \leftarrow \text{RESULTANT}(f_2, f_3, y)$
16 $r_{123}(z) \leftarrow \text{RESULTANT}(r_{12}, r_{23}, x)$
17 $roots[\,] \leftarrow \text{FINDROOTS}(r_{123}(z))$
18 **for each** z_0 **in** $roots[\,]$
19 **do**
20 $p, q \leftarrow \text{SOLVE}(x^2 - (z_0 + 1)x - N)$
21 **if** $N = p \cdot q$
22 **then**
23 **return** p, q

Abbildung 3.19: Algorithmus für den Angriff BM3

4 Anforderungen an eine sichere Implementierung von RSA

Nachdem nun die bekannten Angriffe auf RSA eingehend untersucht wurden, sollen nun Abwehrmaßnahmen diskutiert werden. Zunächst werden die durch die Angriffe entstehenden Einschränkungen auf die Parameter e und d vorgestellt. Neben den gezeigten Angriffen auf die RSA Funktion selbst, sind noch weitere Angriffe möglich, die sich durch eine unsichere Implementierung oder Ausnutzung eines Seitenkanals ergeben. Abschnitt 4.2 erläutert kurz die bekannten Seitenkanal-Angriffe und geeignete Gegenmaßnahmen. In Abschnitt 4.3 wird der Chosen-Ciphertext Angriff von Bleichenbacher und die sich daraus ergebenden Anforderungen an das Padding der Nachricht vor dem Verschlüsseln vorgestellt.

4.1 Vermeiden falscher Parameter für die Exponenten e und d

Wenden wir uns der Frage zu, welche Einschränkungen sich für die Parameter e und d durch die in Kapitel 3 beschriebenen Angriffe ergeben. Genauer soll der Frage nachgegangen werden, ob generell von der Verwendung kleiner Exponenten e und d Abstand genommen werden sollte.

Kleine öffentliche Exponenten

Die Angriffe aus den Abschnitten 3.2.1, 3.2.2 und 3.2.3 zeigen, dass kleine öffentliche Exponenten ein Sicherheitsrisiko darstellen, dennoch werden sie heute noch verwendet. Das liegt daran, dass digitale Signaturen in der Regel wesentlich häufiger geprüft als erzeugt werden und ein kleines e die Signaturprüfung und Verschlüsselung stark beschleunigt. Der häufig für e verwendete Wert $2^{16}+1$ ist durch Coppersmiths Angriffe auf Stereotype Nachrichten oder die Random Pad Attacke bei üblichen Werten für N nicht bedroht. Für den Angriff auf Broadcasted Messages von Håstad sind mit dieser Wahl von e mindestens

65537 Nachrichten nötig. Mit $e = 2^{16} + 1$ werden zur Prüfung einer Signatur 17 Multiplikationen benötigt, bei $e = 3$ sind es zwar nur 3 aber verglichen mit ca. $\log_2(\Phi(N))$ bei einer zufälligen Wahl von e ein guter Kompromiss. Daher wird häufig empfohlen $e = 2^{16} + 1$ zu verwenden [Bon99, Sch99].

Allerdings kann weiterhin $e = 3$ verwendet werden, wenn bestimmte Vorsichtsmaßnahmen getroffen werden. Die Angriffe funktionieren nur bei teilweise bekannten oder linear abhängigen Nachrichten oder bei Nachrichten, bei denen nur ein kurzer zufälliger Füllstring verwendet wurde. Werden die Nachrichten vor dem Verschlüsseln durch eine geeignete Funktion und unter Verwendung von zufälligen Daten kodiert, können demnach die Angriffe zuverlässig unterbunden werden. Wird lediglich ein zufälliger Sitzungsschlüssel geeignet verschlüsselt stellen kleine e ebenfalls keine Gefahr dar. So kommen Ferguson und Schneier in [FS03] auch zu der Empfehlung $e = 3$ bei Signaturen und $e = 5$ beim Verschlüsseln zu verwenden.

Der von den RSA Labs veröffentlichte Public-Key Cryptography Standard (PKCS) [RSA01] beinhaltet generell keine expliziten Empfehlungen für die Wahl von e und d. Um den Angriff von Håstad abzuwenden, kommt hier eine Funktion (EME-PKCS1) zum Einsatz, die die Nachricht vor dem Verschlüsseln kodiert, wobei zufällige Daten eingebracht werden. Um auch die Angriffe nach Coppersmith wirksam zu verhindern, wird seit Version 2.0 eine andere Kodierungsfunktion (EME-OAEP) empfohlen, wobei explizit auf die Ergebnisse von Coppersmith hingewiesen wird. Das zeigt, dass die gitterbasierten Angriffe auch von Standard-gebenden Organisationen als Bedrohung erkannt wurden, dass sie jedoch mit geeigneten Gegenmaßnahmen effektiv abgewendet werden können.

Details zu diesen Funktionen werden in Abschnitt 4.3 diskutiert.

Kleine geheime Exponenten

Auch die Verwendung von kleinen d ist aus Gründen verbesserter Geschwindigkeit wünschenswert. Davon kann allerdings nur abgeraten werden. Zwar können die Angriffe von Wiener, Boneh und Durfee sowie Blömer und May abgewendet werden wenn $e > N^{1,875}$ gewählt wird. Das würde jedoch die Verschlüsselung bzw. die Signaturprüfung stark verlangsamen. Außerdem ist es ein Unsicherheitsfaktor, welche Verbesserungen für diese Angriffe in Zukunft zu erwarten sind. Boneh hält die Schranke von $d < N^{0,292}$ nicht für korrekt sondern $d < N^{0.5}$ [Bon99, S. 206]. Alexander May hält für die von Boneh und Durfee verwendeten Polynome $N^{0.292}$ für die korrekte Schranke [May04]. Auch der von Verheul und van Tilborg vorgestellte Angriff [VT97] rechtfertigt die Wahl von $d > N^{0,5}$. Wählt man zunächst ein kleines e wie 3, 5, 17 oder $2^{16} + 1$ und bestimmt dazu das multiplikative Inverse zu $\Phi(N)$, so ist d in der Größenordnung von N und somit sicher

gegen die genannten Angriffe.

4.2 Vermeidung von Seitenkanalangriffen

Bisher wurden nur Angriffe auf RSA betrachtet, die versuchen die mathematischen Funktionen des Verfahrens auszunutzen. Die Sicherheit von RSA in realen Anwendungen hängt darüber hinaus jedoch noch von anderen Faktoren ab. Die verwendete Hardware und reale Implementierungen geben dem Angreifer zusätzliche Informationen, die in das mathematischen Modell des Kryptosystems nicht eingehen. Dazu zählen Informationen zur Laufzeit, dem Stromverbrauch, elektromagnetischer Strahlung, Geräuschentwicklung und das Verhalten bei Hardwarefehlern. Diese so genannten Seitenkanalangriffe machen sich diese physikalischen Eigenschaften zu nutze, um das mathematische Problem, auf dem die Sicherheit des Kryptosystems beruht, zu vereinfachen oder ganz zu umgehen. Im Folgenden werden die oben genannten Kanäle und die dazu veröffentlichten Angriffe kurz beschrieben und Gegenmaßnahmen vorgestellt.

Kocher beschreibt in [Koc96] einen Angriff, bei dem die Zeit gemessen wird, die zum Erstellen einer Signatur oder zum Entschlüsseln einer Nachricht benötigt wird. Der Zeitaufwendigste Schritt bei diesen Vorgängen ist die Berechnung der modularen Potenz $c^d \mod N$. Um diesen Schritt zu beschleunigen kann das Verfahren des wiederholten Quadrierens und Multiplizierens verwendet werden. Für jedes Bit d_i von d wird dabei ein modulares Quadrat berechnet und wenn $d_i = 1$ wird zusätzlich noch eine Multiplikation durchgeführt. Daher wird abhängig von d_i unterschiedlich viel Zeit für den jeweiligen Schleifendurchlauf benötigt. Der Angreifer erstellt vor dem Angriff eine Menge von Chiffretexten c_i und misst die Zeit T_i, die mit der Hardware (meistens eine Smartcard) benötigt wird, um $c_i^2 \cdot c_i \mod N$ zu berechnen. Kocher machte die Beobachtung, dass wenn $d_1 = 1$ ist, die Zeit, die die Hardware benötigt, um $c_i^d \mod N$ zu berechnen, mit diesen T_i korreliert ist. Wenn $d_1 = 0$ ist, dann verhalten sich die Zeiten völlig unabhängig. Nach einer Reihe von Entschlüsselungsoperationen kann der Angreifer den Wert von d_1 bestimmen. Mit der gleichen Technik fährt er nun für die nächsten Bits fort. Unter kontrollierten Bedingungen ist es möglich diesen Angriff erfolgreich durchzuführen, allerdings sind dafür Hunderte oder Tausende von Messungen erforderlich. Wird die so genannte Montgomery Reduktion verwendet, um die modulare Multiplikation zu beschleunigen, sind die Zeitunterschiede zwischen den Berechnungen für $d_i = 1$ und $d_i = 0$ geringer, was den Angriff erschwert. Wird der Chinesische Restsatz verwendet, um die Berechnung zu beschleunigen, kann der Angriff so nicht durchgeführt werden. Dabei werden $d_p = d \mod (p-1)$ und $d_q = d \mod (q-1)$ verwendet, um $m_p = c^{d_p} \mod p$ und $m_q = c^{d_q}$

mod q zu berechnen. Anschließend wird aus m_p und m_q mit dem Chinesischen Restsatz die Signatur oder der Klartext berechnet. Da c zunächst modulo p oder modulo q reduziert wird, hat der Angreifer keine direkte Kontrolle über die Eingabe der Funktion und kann den Angriff nicht durchführen. Da diese beiden Verfahren in den meisten Implementierungen von RSA eingesetzt werden, galt lange die Auffassung, diese Art von Angriffen sei in der Praxis nicht einsetzbar.

Im Jahr 2003 veröffentlichten Brumley und Boneh in [BB03] einen Timing-Angriff, den sie erfolgreich gegen OpenSSL, der verbreitetsten Open-Source SSL-Bibliothek, einsetzten. OpenSSL setzt die Montgomery Reduktion ein, um die Berechnung zu beschleunigen. Bei diesem Verfahren steigt die Laufzeit mit zunehmender Größe des Chiffretextes c an, solange diese kleiner ist als p oder q. Ist c genau so groß oder nur wenig größer als p oder q, fällt die Laufzeit stark ab. Dieser Zeitunterschied ist ein Indiz für die Größe der Faktoren p und q. Eine weitere Veränderung in der Laufzeit ergibt sich aus der Verwendung des verschiedener Multiplikationsalgorithmen abhängig von der Eingabe. Ähnlich wie beim oben geschilderten Angriff wird nun mit einer Reihe von Zeitmessungen für die Entschlüsselung von entsprechend gewählten Chiffretexten Bit für Bit eines der Faktoren p oder q herausgefunden.

Bei Experimenten konnten Brumley und Boneh einen geheimen Schlüssel eines Webservers herausfinden, mit dem sie über mehrere Netzwerkknoten verbunden waren. Der Versuchsaufbau ist zwar deutlich günstiger für den Angriff, als es ein Angriff über das Internet wäre, aber es wird deutlich, dass Timing-Angriffe eine reale Bedrohung darstellen. Daher wurden nach dieser Veröffentlichung Patches für OpenSSL, Apaches mod_SSL und viele andere Produkte herausgebracht und ihre Aufspielung nahegelegt. Als Gegenmaßnahme wird von Brumley und Boneh das so genannte Blinding (deutsch: *Blenden*) der Nachricht vor dem Entschlüsseln empfohlen. Dabei wird zunächst eine zufällige Zahl r erzeugt und $x = r^e \cdot c \mod N$ berechnet. Damit wird die Entschlüsselung $m' = x^d \mod N$ durchgeführt. Anschließend wird mit $m = m'/r \mod N$ der richtigen Klartext berechnet. Da die eigentliche Entschlüsselung mit zufälligen Daten durchgeführt wird, kann der Angreifer aus der Laufzeit keine Information gewinnen.

Ein anderer Seitenkanal, über den ein Angreifer Informationen erhalten kann, ist die Änderung des Stromverbrauches über die Zeit, während einer Operation mit dem geheimen Schlüssel. Dabei wird der Effekt ausgenutzt, dass verschiedene Instruktionen unterschiedlich viel Strom verbrauchen. Kocher, Jaffe und Jun zeigen in [KJJ99] die Anfälligkeit von Smartcards für derartige Angriffe. Da die Multiplikation einen messbar erhöhten Stromverbrauch hat, kann überprüft werden, ob eine oder zwei Multiplikationen während eines modularen Potenzierungschritts gemacht werden. Sind die Signalunter-

schiede zu gering, um sie direkt zu messen, oder werden sie durch zu starkes Rauschen überdeckt, kann eine differentielle Stromanalyse verwendet werden, die sich statistischer Methoden bedient und direkt auf den verwendeten Algorithmus zugeschnitten ist. Kocher, Jaffe und Jun betonen, dass vergleichbare Techniken auch für elektromagnetische Strahlung eingesetzt werden können. In [AARR03] beschreiben Agrawal, Archambeault, Rao und Rohatgi, wie elektromagnetische Strahlung als Seitenkanal verwendet werden kann.

Als Gegenmaßnahmen gegen die Strom- und Strahlungsanalyse kann durch ein sorgfältiges Design und Abschirmungen die Signalstärke reduziert werden, damit eine Messung erschwert wird. Außerdem kann der Informationsgehalt der Messungen durch Verwendung von zufälligen Daten und häufigem Wechsel der Schlüssel vermindert werden. Kocher, Jaffe und Jun halten eine enge Zusammenarbeit zwischen den Designern der Algorithmen, Protokolle, Software und Hardware für notwendig, um diese Art von Angriffen zu verhindern [KJJ99, S. 10].

Auf [ST04] zeigen Shamir und Tromer erste Ergebnisse zu Experimenten mit Schallwellen, die von einem PC ausgestrahlt werden. Sie werden von Kondensatoren ausgestrahlt, die abhängig von der jeweiligen Prozessorinstruktion charakteristische Schwingungen erzeugen. Shamir und Tromer konnten zeigen, dass unterschiedliche Schlüssel beim Signiervorgang auch unterschiedliche Klangspektren erzeugen. Dabei kam normale PC-Hardware zum Einsatz und auch die verwendeten Messinstrumente sind im unteren Preissegment angesiedelt. Gegen Messungen der elektromagnetischen Strahlung oder des Stromverbrauchs sind viele Systeme bereits abgeschirmt. Schallwellen könnten diese Abschirmungen laut Shamir und Tromer durchdringen. Als Gegenmaßnahmen empfehlen sie daher entweder den Schall zu dämmen oder durch Rauschen zu überlagern. Alternativ kann durch gutes Design der Hardware und hochwertige Komponenten die Abstrahlung vermindert werden. Wie im letzten Absatz erläutert, kann auch ein darauf ausgelegtes Design der Algorithmen dazu beitragen, dass die Abstrahlung keine Information über die verwendete Eingabe der Funktion preisgibt.

Auch Hardwarefehler können dem Angreifer dienen, einen geheimen Schlüssel herauszufinden. Boneh, DeMillo und Lipton [Bon99] zeigen, dass ein einziges übergesprungenes Bit bei einem Signiervorgang im Extremfall ausreichen kann, um den geheimen Schlüssel herauszufinden. Hardwarefehler sind zwar in der Regel extrem selten, ein Angreifer könnte sie jedoch durch elektromagnetische Störungen, Spannungsschwankungen oder andere Manipulationen künstlich herbeiführen. Als Gegenmaßnahme empfehlen Boneh, DeMillo und Lipton, die Ausgabe der Funktion zu überprüfen. Wird als öffentlicher Exponent $e = 3$ verwendet, kann diese Prüfung relativ schnell durchgeführt werden.

Viele der hier präsentierten Angriffe finden den geheimen Schlüssel Bit für Bit. Das unterstreicht die praktische Relevanz der in Kapitel 3 vorgestellten Angriffe auf teilweise bekannte d, p oder q. Ein Angreifer kann unter Umständen den Angriff schon früher beenden und die verbleibenden Bits mit den gezeigten Gitterangriffen bestimmen.

In diesem Abschnitt wurde gezeigt, dass bei der Analyse der Sicherheit einer RSA-Implementation auch die Rahmenfaktoren betrachtet werden müssen. Neben der Analyse der Algorithmen und Protokolle müssen auch Software und Hardware einbezogen werden.

4.3 Korrektes Padding

Bevor Daten mit RSA verschlüsselt werden, müssen sie mit einem so genannten Padding (deutsch: *Füllung*) versehen werden. Das hat mehrere Gründe, unter anderem, um die gezeigten Angriffe bei kleinem e zu verhindern, aber auch, um kurze Nachrichten, wie etwa einen symmetrischen Schlüssel, überhaupt verschlüsseln zu können, wie das folgende Beispiel veranschaulicht.

Beispiel 4.1. *Sei N ein RSA-Modulus mit 1024 Bit Länge und $e = 3$ der öffentliche Exponent. Sei weiter m ein symmetrischer Schlüssel mit 128 Bit Länge. Ohne geeignete Kodierung würde sich als Chiffretext c der Wert $m^e \mod N$ ergeben. Die Länge von m^3 beträgt maximal $3 \cdot 128 = 384$ Bit und somit ist m^e kleiner als N. Daraus folgt, dass $c = m^3$ ist, da keine Reduktion modulo N stattfindet. Daher kann einfach die dritte Wurzel von c berechnet werden, um m zu erhalten.*

Das bislang gebräuchlichste Füllschema für RSA-Verschlüsselung wird im Public Key Cryptography Standard #1 (PKCS#1) definiert [RSA01]. Es wird mit EME-PKCS1-v1_5 (Encoding Method for Encryption PKCS#1 v1.5) oder auch Block-02-Padding bezeichnet. Dabei wird ein Puffer vorbereitet, der so viele Bytes fassen kann, wie der Modulus lang ist. An das Ende dieses Puffers wird die zu verschlüsselnde Nachricht geschrieben. Das erste Byte wird mit 00_{16} und das zweite mit 02_{16} gefüllt. Außer dem letzten Byte vor der Nachricht werden nun alle freien Bytes mit zufälligen Werten ungleich 00_{16} gefüllt. Das letzte Byte vor der Nachricht erhält den Wert 00_{16} und markiert damit das Ende der Füllzeichen. Abbildung 4.1 zeigt das Format einer derart aufgefüllten Nachricht.

Der Sender verschlüsselt den so entstandenen Block nun mit dem RSA-Verfahren und verhindert so zum Beispiel den Angriff von Håstad. Nachdem der Empfänger den so entstandenen Chiffretext entschlüsselt hat, prüft er, ob an den erwarteten Stellen 00_{16} und 02_{16} vorkommen, und weiß so auch, wo das Padding beendet ist und wo die Nachricht

Abbildung 4.1: Mit Block-02-Padding aufgefüllte Nachricht

beginnt. Diese Art des Auffüllens verhindert jedoch nicht die Angriffe von Coppersmith. Ist die zu verschlüsselnde Nachricht zu lang, wird der zufällige Anteil möglicherweise zu klein und die Nachricht wird für den Short-Random-Pad-Angriff anfällig. Es ist auch möglich, den Angriff auf stereotype Nachrichten auf den bivariaten Fall zu erweitern. Dann kann das zufällige Padding als die eine und der unbekannte Teil der Nachricht als die andere Unbekannte angesehen werden. Auf diese Bedrohungen weist [RSA01] seit Version 2.0 ausdrücklich hin. Eine Gegenmaßnahme ist in diesem Fall die Nachricht auf mehrere Blöcke aufzuteilen, um den zufälligen Anteil pro Block zu erhöhen.

Die Struktur des Block-02-Padding machte sich 1998 Daniel Bleichenbacher zu nutze, um einen Angriff zu entwickeln [Ble98]. Die Idee dabei ist eine Fehlermeldung des Empfängers auszunutzen, um Informationen zu erhalten. Mit Hilfe eines solchen Orakels ist es möglich eine geheime Nachricht zu entschlüsseln. Der Angriff wird hier nur kurz skizziert:

Aus einer abgefangenen Nachricht wird durch eine spezielle Funktion eine verwandte Nachricht erzeugt. Diese wird dann an den Empfänger gesendet. Wenn der Empfänger die Nachricht entschlüsselt hat, prüft er, ob am Anfang $00\ 02_{16}$ steht und ob in der Nachricht die 00_{16} vorkommt. In den meisten Fällen wird dies nicht so sein und es wird eine Fehlermeldung an den Sender geschickt. Dieser erzeugt darauf hin eine neue verwandte Nachricht und lässt sie wieder vom Empfänger prüfen. Nach einer bestimmten Anzahl von Versuchen, wird die Nachricht im korrekten Format sein und der Empfänger meldet keinen Fehler. Die damit gefundene Nachricht enthält zwar keinen sinnvollen Inhalt, der Angreifer erhält jedoch mit jeder gültigen Nachricht mehr Informationen zum Klartext der ursprünglichen Nachricht. Da jeder gesendete Chiffretext auf dem ursprünglichen Chiffretext sowie den Antworten des Empfängers basiert, spricht man hier von einem adaptiven Angriff mit frei wählbarem Geheimtext (engl. *adaptive chosen-chiphertext attack*). Mit einer bestimmten Anzahl von akzeptierten Nachrichten kann der Angreifer die ursprüngliche Nachricht rekonstruieren. Nach Experimenten von Bleichenbacher muss der Algorithmus dazu zwischen 300 Tausend und 2 Millionen Nachrichten prüfen lassen. Daher ist der Angriff zum Beispiel per E-Mail nicht durchführbar, da eine so große Zahl von E-Mail niemals von einem Empfänger beantwortet werden würde. Für SSL-Server stellte er jedoch tatsächlich eine Bedrohung dar. Es gibt verschiedene Präventivmaßnah-

men gegen diesen Angriff und laut Burnett und Paine [BP01, S. 451] ist es in der Praxis wahrscheinlich weiterhin sicher PKCS#1 Block-02-Padding zu verwenden.

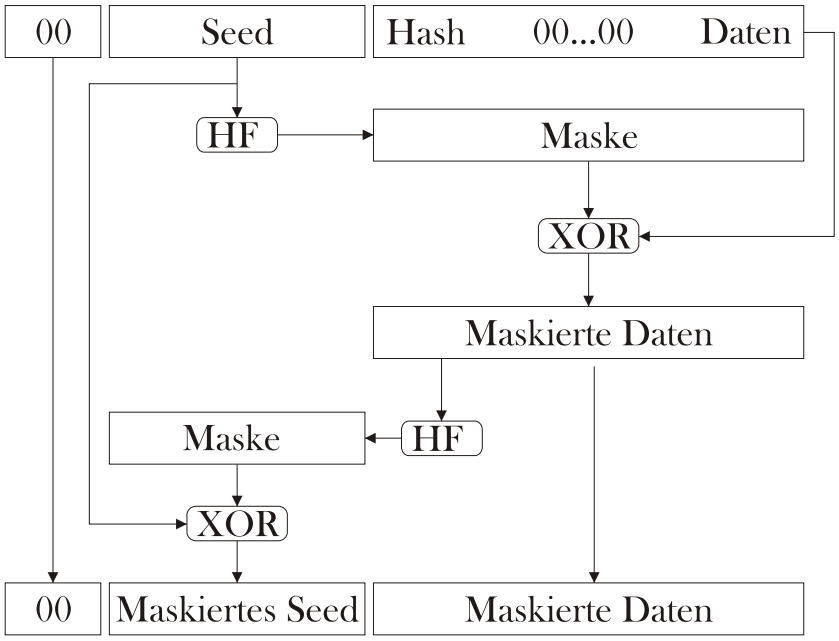

Abbildung 4.2: Diagramm der OAEP Füllfunktion

Um jedoch diesen Angriff und auch die zuvor genannten Angriffe auf kleine Exponenten wirksam zu verhindern, empfiehlt der PKCS-#1-Standard seit der Version 2.0 eine andere Padding Funktion zu verwenden. Beim so genannten *Optimal Asymmetric Encryption Padding* (OAEP) von Bellare und Rogaway [BR95] werden wiederholt Hashfunktionen verwendet, um die Struktur des verschlüsselten Textes möglichst zu zerstören. Man beginnt ähnlich wie beim Block-02-Padding mit einem Puffer, an dessen Ende man die zu verschlüsselnden Daten schreibt. An den Anfang des Puffers wird 00_{16} gefolgt von 20 Byte zufälligen Daten, dem so genannten Seed (deutsch: *Saat*) geschrieben. Nach dem Seed werden weitere 20 Byte Hashwert irgendwelcher, beiden Parteien bekannter Daten geschrieben. Der Rest des Puffers wird mit 00_{16} gefüllt. Wie Abbildung 4.2 verdeutlicht, besteht die Nachricht nun aus drei Teilen: Am Anfang steht 00_{16}, dann kommt das Seed und am Ende steht die Nachricht, die mit einem Hashwert und Nullen aufgefüllt ist. Nun wird aus dem Seed mit einer Hashfunktion eine Maske erstellt, die so lang ist, wie der

hintere Teil der Nachricht. Diese Maske wird mit dem hinteren Teil XOR-verknüpft und man erhält einen maskierten Datenblock. Daraus wird ebenfalls eine Maske gebildet, die mittels XOR über das Seed gelegt wird. Die Nachricht, die mit RSA verschlüsselt wird, setzt sich aus der 00_{16}, dem maskierten Seed und den maskierten Daten zusammen. Der Empfänger entschlüsselt die Nachricht, bildet die Maske für das Seed aus den maskierten Daten und verknüpft diese mit dem Seed. So erhält er wieder das ursprüngliche Seed. Daraus erstellt er die Maske für die Daten und erhält durch XOR-Verknüpfung wieder die Originaldaten. Um die Nachricht zu prüfen, erstellt er aus den beiden Parteien bekannten Daten einen Hashwert und vergleicht ihn mit dem in der Nachricht übertragenen.

Die Wahrscheinlichkeit, dass jemand einen Chiffretext erzeugen kann, dessen Klartext diesem Schema entspricht, ohne den zugehörigen Klartext zu kennen, ist verschwindend gering. Der Angreifer müsste dazu einen Chiffretext erzeugen, der nach dem Entschlüsseln und OAEP-Dekodieren an der richtigen Stelle den 20-Byte Hashwert der bekannten Daten und eine noch größere Anzahl von Nullen besitzt. Da die verwendeten Hashfunktionen nicht umkehrbar sind, kann solch eine Nachricht nicht effizient hergestellt werden. Daher ist diese Art des Paddings auch ein wirksame Schutz vor dem Bleichenbacher Angriff, da der Empfänger die gefälschte Nachricht wahrscheinlich niemals entschlüsseln kann und keine Fehlermeldung zurück sendet.

Neben den beiden vorgestellten Padding-Funktionen gibt es noch eine ganze Reihe weitere, die zum Beispiel in [Nac99] beschrieben werden. Für die Sicherheit von RSA sind derartige Füllfunktionen absolut notwendig, wichtig ist jedoch auf gut untersuchte Funktionen zurückzugreifen und möglichst keine eigenen Entwicklungen zu verwenden.

5 Ökonomische Konsequenzen

In Kapitel 1 wurde gezeigt, dass RSA der defacto-Standard für Verschlüsselungen und digitale Signaturen ist. Zertifizierung und Verschlüsselung in betrieblichen Anwendungen sind im Informationszeitalter Schlüsseltechnologien. Moderne Geschäftsprozesse benötigen verlässliche Sicherheitslösungen. Ausschlaggebend für die Investitionssicherheit ist die Zukunftssicherheit und Standardisierung der eingesetzten Verfahren, die mit RSA gegeben ist. Die große Verbreitung von RSA stellt jedoch auch ein Risiko dar, da es im Moment keine etablierten Alternativen gibt.

Daher ist leicht abzuschätzen, dass ein totaler Bruch von RSA – als tragender Säule der gesamten IT-Sicherheitslandschaft – verheerende ökonomische Konsequenzen für Anbieter und Anwender von Sicherheitslösungen haben würde. Selbst wenn sich rasch eine Ersatztechnologie durchsetzen würde, bleibt fraglich, ob diese schnell in bestehende Lösungen eingepflegt werden kann. Denn, wie in Kapitel 4 gezeigt wurde, reicht es für ein sicheres Gesamtsystem nicht aus nur die mathematischen Grundlagen zu berücksichtigen. Vom Protokoll bis zur Hardware muss das gesamte System integriert betrachtet und entwickelt werden.

Auch wenn die hier gezeigten Angriffe weit entfernt von einem totalen Bruch von RSA sind, ergeben sich aus ihnen geänderte Anforderungen an gegenwärtige und zukünftige Implementierungen. Sie unterstreichen die Wichtigkeit diese Entwicklungen im Auge zu behalten, um schnell auf neue Angriffe reagieren zu können. Neben den real entstehenden Kosten sind im Falle eines Sicherheitsbruches weitere Effekte zu erwarten. Campbell et al. zeigen in [KCZ03], dass veröffentlichte Einbrüche in IT-Sicherheitssysteme, bei denen auf vertrauliche Informationen zugegriffen wurde, spürbar negative Auswirkungen auf den Aktienkurs der betroffenen Firma hatten. Nicht nur das Vertrauen der Anleger kann durch derartige Veröffentlichungen beeinträchtigt werden, sondern auch das der Kunden. Das Vertrauen des Kunden ist aber gerade für den Erfolg von Online-Shopping und Online-Banking eine Grundvoraussetzung und nicht selten Alleinstellungsmerkmal gegenüber der Konkurrenz.

Neben den Kosten, die ein tatsächlicher Angriff verursacht, sind auch die Kosten zu berücksichtigen, die für Präventivmaßnahmen aufzubringen sind. Die Fortschritte

bei der Faktorisierung großer Zahlen machen immer längere Schlüssel notwendig, um das gewünschte Sicherheitslevel aufrecht zu erhalten. Schon heute beanspruchen RSA-Operationen viel Rechenzeit. Gerade in Internetanwendungen, bei denen Server eine große Menge dieser Operationen durchführen müssen, wird beim Einsatz längerer Schlüssel mehr Rechenkapazität benötigt, wodurch zusätzliche Kosten entstehen.

Auch die Hardware wird im Zuge verbesserter Seitenkanalangriffe teurer werden, denn z.B. Abschirmung von elektromagnetischen Wellen oder Sicherung gegen Manipulation erhöhen die Stückkosten von Smartcards und ähnlicher Hardware.

Der Angriff von Bleichenbacher und der Timing-Angriff von Brumley und Boneh machten Anpassungen der Standards nötig. Als Reaktion auf den Timing-Angriff wurden sogar für die verbreitetsten SSL-Implementierungen Patches und Updates herausgegeben, deren Aufspielung mit Kosten verbunden war.

6 Fazit

Die Angriffe, die auf Gitterreduktion basieren, zeigen, dass bei der Verwendung von kleinen öffentlichen Exponenten besondere Sorgfalt nötig ist. Völlig auf kleine öffentliche Exponenten zu verzichten scheint jedoch nicht indiziert. Geeignete Paddingfunktionen stellen bereits eine einfache und wirksame Gegenmaßnahme dar. Der häufig verwendete Exponent $2^{16} + 1$ ist von diesen Angriffen nicht bedroht.

Kleine geheime Exponenten sollten spätestens seit den Ergebnissen von Wiener nicht mehr benutzt werden. Die neuen Erkenntnisse von Boneh, Durfee, Blömer und May haben daher keine direkte praktische Relevanz. Sie zeigen jedoch die Möglichkeit der Entwicklung auf diesem Gebiet. Generell muss hier die Empfehlung lauten, einen geheimen Exponenten d größer als $N^{\frac{1}{2}}$ zu wählen. Denn es ist laut Boneh [Bon99, S. 206] und May [May04] unwahrscheinlich, dass der Angriff auf $d > N^{\frac{1}{2}}$ auszuweiten ist.

Die Angriffe auf teilweise bekannte Schlüssel haben gerade im Zusammenspiel mit Seitenkanalangriffen eine praktische Relevanz, da sie diese beschleunigen können. Der Angriff von Bleichenbacher und der Timing-Angriff von Brumley und Boneh sorgten für Anpassungen der Standards und machten in vielen verbreiteten Produkten ein Update notwendig.

Die in dieser Arbeit vorgestellten Entwicklungen zeigen, dass auch bei einem so gut untersuchten und lange bekannten Verfahren wie RSA bekannte Angriffe verbessert und neue Angriffe entwickelt werden können. Sie zeigen aber auch, dass RSA nach wie vor weit von einem totalen Bruch entfernt ist.

Kurzfristig wird RSA daher weiterhin das Verfahren der Wahl sein. Mittelfristig muss das Ziel sein Lösungen zu schaffen, die es erlauben, den kryptographischen Unterbau flexibel auszutauschen, um auf neuartige Angriffe reagieren zu können. Langfristig werden Alternativen zu RSA benötigt, denn die Entwicklungen im Bereich der Faktorisierung, gerade im Zusammenhang mit Spezialhardware, machen RSA in Zukunft zunehmend angreifbar. Auch wenn es theoretisch möglich ist, mit immer längeren Schlüsseln die Sicherheit für einen längeren Zeitraum aufrecht zu erhalten, ist es nicht ökonomisch sinnvoll. Gerade serverseitig ist die Rechenkapazität, die für RSA-Berechnungen aufgebracht werden muss, ein limitierender Faktor. Mögliche Nachfolger können Verfahren sein, die

auf elliptischen Kurven basieren. Sie haben bessere Sicherheitseigenschaften, kleinere Schlüssel und versprechen höhere Geschwindigkeiten. Jedoch ist die Standardisierung noch nicht vollständig abgeschlossen und Verfahren, die die Performance des Verfahrens ausmachen, sind patentrechtlich geschützt. Egal welche Verfahren als RSA-Nachfolger in Betracht kommen, die Einstiegsinvestitionen werden hoch sein. Die Investitionssicherheit kann jedoch nur durch Standardisierung und Interoperabilität gewährleistet werden.

A Implementierungen

Mit Hilfe der NTL-Bibliothek [Sho] wurden drei gitterbasierte Angriffe, die in dieser Arbeit vorgestellt wurden, implementiert. Ziel dieser Implementierungen ist es, die Anwendbarkeit dieser Angriffe zu demonstrieren und ihre praktischen Laufzeiten zu ermitteln. Durch das Einbringen der bislang wenig bekannten Angriffe in das freie eLearning-Programm CrypTool sollen sie mehr in den Fokus der Aufmerksamkeit der Anwender und Entwickler von RSA-basierten Lösungen gerückt werden. So kann am praktischen Beispiel erprobt werden, wie die öffentlichen und geheimen RSA-Schlüsselparameter e und d in Abhängigkeit von N für eine sichere RSA-Implementierung gewählt werden müssen.

In diesem Anhang werden zentrale Codefragmente präsentiert, Entwurfsschritte nachvollzogen und Laufzeiten vorgestellt. Die Benutzeroberflächen sind als Dialoge aufgebaut und in die CrypTool-Software integriert. Ihre Verwendung wird in der zugehörigen Kontexthilfe dokumentiert.

Es wurde der Angriff auf stereotype Nachrichten, der Faktorisierungs-Angriff für bekannte MSB und LSB sowie der Angriff auf kleine d nach Blömer und May implementiert. Ihr Ablauf lässt sich grob in drei Schritte unterteilen:

1. Aufstellen der Polynome
2. Aufstellen des Gitters
3. Finden der Lösung

Zwischen den Schritten 2 und 3 findet die LLL-Reduktion statt. Hier werden die Implementierungen dieser Schritte getrennt vorgestellt. Dabei wird in jedem Schritt auf die Besonderheiten der einzelnen Angriffe eingegangen und anschließend ihre Laufzeiten vorgestellt.

Aufstellen der Polynome

Die Angriffe sind je in einer eigenen Fachkonzeptklasse implementiert. Die Angriffsparameter werden als Attribute dieser Klassen verwaltet. Die Benutzeroberfläche setzt diese

Attribute abhängig von den Eingaben des Benutzers. Dazu mussten Methoden implementiert werden, die als Eingabe des Benutzers einen String akzeptieren und diesen z.B. als Hexadezimalzahl interpretieren. Da auch Eingaben der Form `2^16+1` möglich sein sollten, wurde zudem noch ein einfacher Parser für arithmetische Ausdrücke implementiert. Für den Angriff auf stereotype Nachrichten wurden außerdem Funktionen benötigt, die die Umwandlung von Strings in Ganzzahlen und umgekehrt ermöglichen. Sind alle Parameter gesetzt, können daraus die Polynome aufgestellt werden. NTL stellt dafür die Klasse `ZZX` zur Verfügung. Das folgende Codefragment zeigt am Beispiel des Faktorisierungsangriffs bei bekannten MSB von p, wie die Polynome gebildet werden. Dabei werden neben dem ursprünglichen Polynom (in diesem Fall $f(xX) = (P + xX)^r$) direkt die später davon benötigten Potenzen $f(xX)^k$ mit Hilfe der Binomialkoeffizienten gebildet und in einem Array abgespeichert.

```
ZZ X=to_ZZ(1);   // Variable für das Gewicht X.
X<<=(bitsOfP-b); // Berechnung der Schranke der gesuchten Nullstelle.
ZZ tmpKoeff;     // Temporäre Variablen für den Koeffizienten,
ZZ sign;         // das Vorzeichen und den Binomialkoeffizienten
ZZ bino;
polyPowers = new ZZX[m+1]; // Array für Potenzen der Polynome.
for (int k=0; k<=m; k++){ // Die äußere Schleife läuft über alle
                          // benötigten Polynome.
  for (int i=0; i<=k; i++) { // Die innere Schleife läuft über die
                             // jeweiligen Monome.
    bino=binom(k,i); // Berechnet den Binomialkoeffizienten.
    tmpKoeff=power(P,k-i)*power(X,i); // Berechnet die Potenz des
                                      // Koeffizienten des ursprünglichen
                                      // Polynoms und des Gewichtes.
    SetCoeff(polyPowers[k],i,bino*tmpKoeff);// Setzt den Koeffizienten
    }                                       // des entsprechenden Monoms
}                                           // auf den berechneten Wert.
```

Für die Berechnung der Polynome der anderen Angriffe wird ähnlich vorgegangen. Ein Problem ergibt sich allerdings beim Angriff auf kleine d. Zum Einen ist die Berechnung der Koeffizienten für Polynome in mehreren Unbekannten mittels Binomialkoeffizienten schwieriger. Zum Anderen stellt NTL keine Unterstützung für diese Polynome bereit. Daher musste zunächst eine eigene Klasse zur Verwaltung von Polynomen in zwei Unbekannten implementiert werden. Dazu wurde die Klasse `ZZXY` implementiert, die die

Multiplikation, Addition, Subtraktion und Pseudodivision unterstützt. Dabei wurde zur Datenhaltung ein Vektor von ZZX-Objekten verwendet, um die darin enthaltenen Optimierungen für diese Operationen teilweise ausnutzen zu können. Für jede Potenz von y wurde ein eigenes ZZX-Polynom angelegt. Das Polynom $xy^2 + 3y^2 - xy + 2y + x - 1$ würde dann also zum Beispiel zerlegt in $y^2(x + 3) + y(-x + 2) + (x - 1)$. Die Funktion zur Pseudodivision mit Rest wurde nach dem in [Coh96] beschriebenen Algorithmus implementiert. Sie wird für den später erläuterten Subresultanten-Algorithmus benötigt.

Aufstellen des Gitters

Aus den berechneten Potenzen der Ausgangsfunktionen wird nun das Gitter aufgestellt. Für Matritzen stellt NTL die Klasse mat_ZZ zu Verfügung. Die Schwierigkeit beim Befüllen der Basismatrix besteht darin, die Indizes für die entsprechenden Potenzen und zugehörigen Koeffizienten so zu berechnen, dass die Basis in der gewünschten Form ist. So ist sicherzustellen, dass die Koeffizienten geordnet nach den Potenzen von x und y eingetragen werden. Hier wird die Funktion BUILDLATTICE des Faktorisierungsangriffes dargestellt, da sie am anschaulichsten ist.

```
void FactorHint::buildLattice(){
 Lattice.kill();       // Variable zur Speicherung des Gitters.
 Lattice.SetDims(d, d); // Setzt die Dimension der Gitterbasis.
 ZZX tmpPoly; // Temporäre Variable für das Polynom.

 for(int k=0; k<m; k++){// Füllt die ersten m Zeilen des Gitters.
   tmpPoly = polyPowers[k];  // Wählt die entsprechende Potenz von f(x).
   tmpPoly = tmpPoly*power(N, m-k); // Multiplikation mit N^(m-k)
   for(int i=0; i<=k; i++) // Füllt die Zeile bis zur Diagonalen mit
     Lattice[k][i]=coeff(tmpPoly, i); // dem Koeffizienten von tmpPoly.
   }
 tmpPoly = polyPowers[m];
 for(k=0; k<d-m; k++){   // Füllt die restlichen d-m Zeilen des Gitters.
   for(i=0; i<=k+m; i++) // Füllt die Zeile bis zur Diagonalen
     Lattice[k+m][i]     // mit den Koeffizienten von F(x)^m * X^k.
       = coeff(tmpPoly, i-k)*power(X, k);
 }
}
```

Wenn die Basismatrix vollständig gefüllt ist, kann sie als Parameter der LLL-Funktion übergeben werden. Beim Angriff auf kleine d ist noch zusätzliche Arbeit zu leisten. Hier müssen vor der LLL-Reduktion noch die Zeilen und Spalten gelöscht werden, wie in Kapitel 3.3 beschrieben wird.

Finden der Lösung

Nachdem das Gitter LLL-reduziert ist, wird aus dem ersten Vektor erneut ein Polynom gebildet. Nun werden die Potenzen von X aus diesem Polynom entfernt und die Nullstellen bestimmt. Allerdings ist in der NTL-Bibliothek kein Verfahren zur Nullstellensuche enthalten. Nun wäre es möglich gewesen, ein Näherungsverfahren zur Nullstellensuche zu implementieren. Da wir jedoch lediglich an den ganzzahligen Nullstellen des Polynoms interessiert sind, gibt es einen einfacheren Weg, diese zu finden. Jedes Polynom mit Koeffizienten aus $\mathbb{Z}$, das eine Nullstelle $x_0 \in \mathbb{Z}$ hat, muss sich ohne Rest durch $(x_0 - x)$ teilen lassen. Die Funktion FACTOR, die in NTL enthalten ist, faktorisiert ein gegebenes Polynom vollständig. Für jede ganzzahlige Nullstelle x_0 wird also ein Faktor in der Form $(x_0 - x)$ gefunden.

Für den Angriff auf kleine d ist erneut mehr Aufwand nötig, um die gesuchte Lösung zu finden. Zunächst müssen die gelöschten Spalten wiederhergestellt werden. Anschließend werden die bivariaten Polynome aus dem wiederhergestellten Gitter aufgestellt. Nun müssen die Resultanten dieser Polynome bestimmt werden. Da in NTL keine Unterstützung für Polynome in mehreren Unbekannten enthalten ist, existiert auch keine Funktion zur Bestimmung von Resultanten. Daher wurde der so genannte Subresultanten-Algorithmus implementiert, der ebenfalls in [Coh96] beschrieben ist und auch in kommerziellen Computeralgebra-Anwendungen verwendet wird. Die Polynome werden nach ihrer gewichteten Norm geordnet, da sich diese bei der Wiederherstellung ändert. Beginnend mit den Polynomen mit der kleinsten gewichteten Norm wird nun versucht die Resultante zu bilden. Wird eine Resultante gefunden, werden die Nullstellen bestimmt und ausgegeben.

Laufzeiten

Dieser Abschnitt zeigt charakteristische experimentelle Laufzeiten, die die verschiedenen Angriffe aufweisen. Alle Tests wurden durchgeführt auf einem Laptop mit einem AMD Athlon XP 2500+ mobile Prozessor mit 1860 MHz und 480 MB Hauptspeicher. Sie dienen lediglich zur Veranschaulichung der Möglichkeiten dieser Angriffe. Die gemessenen Zeiten werden einheitlich in Sekunden angegeben.

L_{unbek}	h	dim	T_{LLL}
34	4	12	2
36	5	15	8
37	6	18	28
38	7	21	83
39	9	27	412
40	12	36	2361

L_N	h	dim	L_{gesamt}	L_{unbek}	T_{LLL}
128	6	18	15	4	1
256	6	18	32	9	2
512	6	18	64	18	5
1024	6	18	128	37	28
2048	6	18	256	75	167
4096	6	18	512	150	1006

Tabelle A.1: Charakteristische Laufzeiten des Angriffs auf stereotype Nachrichten

Tabelle A.1 zeigt die Laufzeiten verschiedener Angriffe auf stereotype Nachrichten. L_{unbek} bezeichnet die Länge des unbekannten Teils der Nachricht in Zeichen. Dabei wird ein Zeichen mit acht Bit Kodiert. T_{LLL} ist die Zeit in Sekunden, die bei diesem Angriff für die LLL-Reduktion benötigt wird. Im linken Teil wird die Änderung der Laufzeit in Abhängigkeit von der Länge des unbekannten Teils L_{unbek} deutlich. Mit zunehmender Länge muss der Parameter h immer größer gewählt werden, wodurch die Dimension dim des Gitters steigt. Die Bitänge L_N des Modulus N wurde dabei konstant bei 1024 Bit gehalten.

Im rechten Teil der Tabelle sieht man, wie sich die Laufzeit mit zunehmendem L_N ändert. Hier wurde auch die Länge L_{gesamt} der gesamten Nachricht erhöht. Wie zu erwarten, wirkt sich eine Verdoppelung der Bitlänge von N weit weniger negativ aus als eine Verdoppelung der Gitterdimension.

Tabelle A.2 zeigt die Ergebnisse des Angriffs nach Blömer und May. Im linken Teil wurde nach zunehmend großen $d = N^\delta$ gesucht, wobei die Bitlänge L_N des Modulus konstant bei 1024 Bit blieb. Neben der Zeit für die Gitterreduktion T_{LLL} wurde auch die Zeit der Resultantenberechnung T_{RES} gemessen. Es fällt auf, dass die Berechnung der Resultante zunächst länger dauert als die Gitterreduktion, bei größerer Gitterdimension jedoch schneller läuft.

Der rechte Teil zeigt die Veränderung der Laufzeit, wenn die Gitterdimension konstant bleibt und sich lediglich die Bitlänge L_N des Modulus ändert. Auch hier ist bei klei-

δ	m	t	dim	T_{LLL}	T_{RES}
0,265	4	2	15	26	14
0,269	5	2	18	135	78
0,27	6	2	21	506	289
0,272	7	3	32	3110	1388
0,274	8	3	36	7552	3503

L_N	δ	m	t	dim	T_{LLL}	T_{RES}
128	0,265	6	2	21	1	7
256	0,265	6	2	21	12	39
512	0,265	6	2	21	77	83
1024	0,265	6	2	21	491	289
2048	0,265	6	2	21	2845	1053

Tabelle A.2: Charakteristische Laufzeiten des Angriffs nach Blömer und May

L_N	L_p	L_P	dim	T_{MSB}	T_{LSB}
1024	512	330	13	5	33
1024	512	320	16	13	86
1024	512	310	18	37	333
1024	512	300	23	239	1073
1024	512	290	30	1109	4545

L_N	L_p	L_P	dim	T_{MSB}	T_{LSB}
256	128	80	15	1	2
512	256	161	15	2	14
768	384	241	15	6	39
1024	512	321	15	13	80
2048	1024	641	15	81	463

Tabelle A.3: Charakteristische Laufzeiten der Faktorisierung bei bekannten Bits von p

neren N die Resultantenberechnung langsamer, bei größeren N jedoch schneller als die Gitterreduktion.

Tabelle A.3 zeigt die Laufzeiten von Experimenten mit den Faktorisierungen von N mit teilweise bekanntem p. T_{MSB} und T_{LSB} geben die benötigte Zeit bei bekannten höchstwertigen bzw. niederwertigsten Bits wieder. L_p, L_P und L_N bezeichnen jeweils die Bitlängen des Faktors p, des bekannten Teils P und des Modulus N. Im linken Teil wurden erneut die Bitlängen von N und p konstant gehalten, um die Auswirkung der Änderung der Dimension genauer betrachten zu können. Es fällt auf, dass die Berechnung bei bekannten LSB von p deutlich länger dauert als bei bekannten MSB. Beim Angriff auf bekannte LSB wird das Polynom mit dem multiplikativen Inversen von $R \mod N$ multipliziert. Die Koeffizienten der Polynome werden dadurch erhöht und die Laufzeit steigt.

Der rechte Teil der Tabelle zeigt die Veränderungen der Laufzeit, wenn die Gitterdimension konstant bleibt und sich lediglich die Länge von N, p und P ändert. Hier wird deutlich, dass ein Angriff auf die LSB von p etwa die gleiche Laufzeit hat, wie ein Angriff auf die MSB von p mit der halben Bitlänge von N, p und P.

Literaturverzeichnis

[AARR03] Agrawal, Dakshi ; Archambeault, Bruce ; Rao, Josyula R. ; Rohatgi, Pankaj: The EM Side-Channel(s). In: *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, Springer-Verlag, 2003. – ISBN 3–540–00409–2, S. 29–45

[AS03] Adi Shamir, Eran T.: Factoring Large Numbers with the TWIRL Device. In: *Crypto 2003, Lecture Notes in Computer Science* Bd. 2729, Springer-Verlag, 2003, S. 1–26

[BB03] Boneh, D. ; Brumley, D.: Remote Timing Attacks are Practical. In: *Proceedings of the 12th USENIX Security Symposium, August 2003.*, 2003

[BD00] Boneh, Dan ; Durfee, Glenn: Cryptanalysis of RSA with private key d less than $N^{0.292}$. In: *IEEE Transactions on Information Theory* Bd. 46, 2000. – ISSN 0018–9448, S. 1339–1349

[BDF98] Boneh, Dan ; Durfee, Glenn ; Frankel, Yair: An attack on RSA given a small fraction of the private key bits. In: *Advances in Cryptology - Asiacrypt'98, Lecture Notes in Computer Science* Bd. 1514, Springer Verlag, 1998, S. 25–34

[BDHG99] Boneh, Dan ; Durfee, Glenn ; Howgrave-Graham, Nick: Factoring $N = p^r q$ for Large r. In: *Advances in Cryptology - Crypto'99, Lecture Notes in Computer Science* Bd. 1666, Springer Verlag, 1999, S. 326–337

[Ber01] Bernstein, D. J. *Circuits for integer factorization: a proposal.* http://cr.yp.to/papers.html Abruf 2004-07-31. 2001

[BFT02] Bourseau, Frank ; Fox, Dirk ; Thiel, Christoph: Vorzüge und Grenzen des RSA-Verfahrens. In: *Datenschutz und Datensicherheit* 26 (2002)

[Ble97] Bleichenbacher, Daniel: On the Security of the KMOV Public Key cryptosystem. In: *Advances in Cryptology - Crypto'97, Lecture Notes in Computer Science* Bd. 1294, Springer-Verlag, 1997, S. 235–248

[Ble98] Bleichenbacher, Daniel: Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In: *Lecture Notes in Computer Science* 1462 (1998), S. 1–12

[BM01] Blömer, Johannes ; May, Alexander: Low Secret Exponent RSA Revisited. In: *Cryptography and Lattice Conference (CaLC 2001), Lecture Notes in Computer Science* Bd. 2146, Springer Verlag, 2001, S. 4–19

[Bon99] Boneh, Dan: Twenty Years of Attacks on the RSA Cryptosystem. In: *Notices of the American Mathematical Society* 46 (1999), Nr. 2, S. 203–213. – ISSN 0002–9920; 1088–9477

[BP01] Burnett, Steve ; Paine, Steven: *Kryptographie - RSA Security's Official Guide.* 1. Auflage. RSA Press, 2001

[BR95] Bellare, M. ; Rogaway, P.: Optimal asymmetric encryption – How to encrypt with RSA. In: Santis, A. D. (Hrsg.): *Advances in Cryptology - Eurocrypt 94, Lecture Notes in Computer Science* Bd. 950, Springer-Verlag, 1995

[BT04] Buchmann, Johannes ; Takagi, Tsuyoshi. *Kryptographie - Chancen und Risiken.* Fachbereich Informatik, Technische Universität Darmstadt, http://www.informatik.tu-darmstadt.de/ftp/pub/TI/TR/TI-03-06.thema_Forschung.pdf Abruf 2004-07-31. 2004

[BV98] Boneh, Dan ; Venkatesan, Ramarathnam: Breaking RSA may not be equivalent to factoring. In: *Eurocrypt '98, Lecture Notes in Computer Science* Bd. 1233, Springer Verlag, 1998, S. 59–71

[Cas71] Cassels, J. W. S. ; Eckmann, B. (Hrsg.): *An Introduction to the Geometry of Numbers.* zweite, korrigierte Auflage. Springer, 1971

[CNS99] Coupè, Christophe ; Nguyen, Phong ; Stern, Jaques: The Effectiveness of Lattice Attacks Against Low-Exponent RSA. In: *Public Key Cryptography '99, Lecture Notes in Computer Science* Bd. 1560, Springer Verlag, 1999, S. 204–218

[Coh96] Cohen, Henri ; Ewing, J.H. (Hrsg.): *A Course in Computational Algebraic Number Theory.* dritte korrigierte Auflage. Springer Verlag, 1996

[Cop96a] Coppersmith, Don: Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known. In: *Advances in Cryptology – EUROCRYPT '96, Lecture Notes in Computer Science* 1070 (1996), S. 178–189

[Cop96b] Coppersmith, Don: Finding a Small Root of a Univariate Modular Equation. In: *Advances in Cryptology – EUROCRYPT '96, Lecture Notes in Computer Science* 1070 (1996), S. 155–165

[Cop97] Coppersmith, Don: Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. In: *Journal of Cryptology: the journal of the International Association for Cryptologic Research* 10 (1997), Nr. 4, S. 233–260

[Cop01] Coppersmith, Don: Finding Small Solutions to Small Degree Polynomials. In: Silverman, Joseph H. (Hrsg.): *Cryptography and Lattice Conference (CaLC 2001), Lecture Notes in Computer Science* Bd. 2146, Springer Verlag, 2001

[DA99] Dierks, T. ; Allen, C.: The TLS Protocol, Version 1.0. In: *Request for Comments (RFC) 2246, Standards Track* (1999)

[Dur02] Durfee, Glenn: *Public Key Cryptanalysis Using Algebraic and Lattice Methods*, Stanford University, Diss., 2002

[FR95] Franklin, Matthew ; Reiter, Michael: A linear protocol failure for RSA with exponent three. In: *Rump Session, Crypto'95* (1995)

[FS03] Ferguson, Niels ; Schneier, Bruce ; Long, Carol A. (Hrsg.): *Practical Cryptography*. Wiley, 2003

[GCL92] Geddes, K. O. ; Czapor, S. R. ; Labahn, G.: *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992

[HG97] Howgrave-Graham, Nick: Finding Small Roots of Univariate Modular Equations Revisited. In: *Cryptography and coding, Lecture Notes in Computer Science* 1355 (1997), S. 131–142

[Hås88] Håstad, Johan: Solving simultaneous modular equations of low degree. In: *SIAM Journal of Computing* 17 (1988), S. 336–341

[JB03] Johannes Blömer, Alexander M.: New Partial Key Exposure Attacks on RSA. In: *Advances in Cryptology (Crypto 2003), Lecture Notes in Computer Science* 2729 (2003), S. 27–43

[KCZ03] Katherine Campbell, Martin P. L. ; Zhou, Lei: The economic cost of publicly announced information security breaches: empirical evidence from the stock market. In: *Journal of Computer Security* 11 (2003), S. 431–448

[KJJ99] Kocher, Paul ; Jaffe, Joshua ; Jun, Benjamin: Differential Power Analysis. In: *Lecture Notes in Computer Science* 1666 (1999), S. 388–397

[Koc96] Kocher, Paul C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: *Lecture Notes in Computer Science* 1109 (1996), S. 104–113

[LLL82] Lenstra, Arjen K. ; Lenstra, Hendrik W. ; Lovász, Lásló: Factoring Polynomials with Rational Coefficients. In: *Mathematische Annalen* 261 (1982), S. 513–534

[LV01] Lenstra, Arjen K. ; Verheul, Eric R.: Selecting Cryptographic Key Sizes. In: *Journal of Cryptology* 14 (2001), S. 255–293

[May03] May, Alexander: *New RSA Vulnerabilities Using Lattice Reduction Methods*, Universität Paderborn, Diss., 2003

[May04] May, Alexander. *Private Kommuniaktion.* 2004

[Nac99] Naccache, David: Padding attacks on RSA. In: *Information Security Technical Report* 4 (1999), Nr. 4, S. 28–33

[NS01] Nguyen, Phong Q. ; Stern, Jacques: The Two Faces of Lattices in Cryptology. In: Silverman, Joseph H. (Hrsg.): *Cryptography and Lattice Conference (CaLC 2001), Lecture Notes in Computer Science*, Springer Verlag, 2001 (Lecture Notes in Computer Science 2146), S. 146–180

[Red96] Redmond, Don: *Monographs and Textbooks in Pure and Applied Mathematics.* Bd. 201: *Number Theory: An Introduction.* Marcel Dekker, New York, 1996

[Reg98] Regulierungsbehörde für Telekommunikation und Post. *Geeignete Kryptoalgorithmen gemäß § 17 Abs. 2 SigV.* Veröffentlich im Bundesanzeiger Nr. 31, S. 1787–1788. 14. Februar 1998

[RSA01] RSA Laboratories. *PKCS #1 v2.1: RSA Cryptography Standard.* ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf, Abruf 2004-07-31. Januar 2001

[Sch99] Schunk, Reinhold: *RSA-Verschlüsselung mit kleinen Exponenten*, TU-Darmstadt, Diplomarbeit, 1999

[SE94] Schnorr, C.P. ; Euchner, M.: Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. In: *Mathematical Programming* 66 (1994), S. 181–191

[Sha99] Shamir, Adi. *TWINKLE*. http://www.jya.com/twinkle.eps Abruf 2004-07-31. 1999

[Shi96] Shimizu, H.: On the improvement of the Hastad bound. In: *IEICE Fall Conference* Bd. A-162, 1996. – in Japanisch

[Sho] Shoup, Victor. *NTL: A Library for doing Number Theory*. http://shoup.net/ntl/ Abruf 2004-07-31

[Sho94] Shor, Peter W.: Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: *IEEE Symposium on Foundations of Computer Science*, 1994, S. 124–134

[ST04] Shamir, Adi ; Tromer, Eran. *Acoustic cryptanalysis – On nosy people and noisy machines*. http://www.wisdom.weizmann.ac.il/~tromer/acoustic/ Abruf 2004-07-31. 2004

[SZ01] Steinfeld, Ron ; Zheng, Yuliang: An Advantage of Low-Exponent RSA with Modulus Primes Sharing Least Significant Bits. In: Naccache, D. (Hrsg.): *Progress in Cryptology - CT-RSA 2001, Lecture Notes in Computer Science* Bd. 2020, Springer-Verlag, 2001, S. 52–62

[VT97] Verheul, Eric R. ; van Tilborg, Henk C. A.: Cryptanalysis of 'Less Short' RSA Secret Exponents. In: *Applicable Algebra in Engineering, Communication and Computing* Bd. 8, Springer-Verlag, 1997, S. 425 – 435

[Wie90] Wiener, Michael J.: Cryptanalysis of short RSA secret exponents. In: *IEEE Transactions on Information Theory* 36 (1990), Nr. 3, S. 553–558. – ISSN 0018–9448

[WLB02] Weis, Ruediger ; Lucks, Stefan ; Bogk, Andreas: Sicherheit von 1024 bit RSA-Schlüsseln gefährdet. In: *Datenschutz und Datensicherheit* 26 (2002)

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Maryborough, Australien, 3. Oktober 2004