CrypTool-Skript*
Mathematik und Kryptographie

Frankfurt am Main

23. April 2003

In diesem *Skript zu dem Programm CrypTool* finden Sie eher mathematisch orientierte Informationen zum Einsatz von kryptographischen Verfahren. Die Hauptkapitel sind von verschiedenen Autoren verfasst und in sich abgeschlossen. Am Ende der meisten Kapitel finden Sie jeweils Literaturangaben und Web-Links.

Sie erhalten Informationen über die Prinzipien der symmetrischen und asymmetrischen **Verschlüsselung**. Ein großer Teil des Skripts ist dem faszinierenden Thema der **Primzahlen** gewidmet. Anhand vieler Beispiele bis hin zum **RSA-Verfahren** wird in die **modulare Arithmetik** und die **elementare Zahlentheorie** eingeführt. Danach erhalten Sie Einblicke in die mathematischen Ideen hinter der **modernen Kryptographie**.

Ein weiteres Kapitel widmet sich kurz den **digitalen Signaturen** — sie sind unverzichtbarer Bestandteil von E-Business-Anwendungen. Das letzte Kapitel stellt **Elliptische Kurven** vor: sie sind eine Alternative zu RSA und für die Implementierung auf Chipkarten besonders gut geeignet.

Während das Programm CrypTool eher den praktischen Umgang vermittelt, dient das Skript dazu, dem an Kryptographie Interessierten ein tieferes Verständnis für die implementierten mathematischen Algorithmen zu vermitteln – und das didaktisch möglichst gut nachvollziehbar.

Die Autoren Bernhard Esslinger, Matthias Büger, Bartol Filipovic, Henrik Koy, Roger Oyono und Jörg-Cornelius Schneider möchten sich an dieser Stelle bedanken bei den Kollegen in der Firma und an den Universitäten Frankfurt, Gießen, Siegen, Karlsruhe und Darmstadt, insbesondere bei Dr. Peer Wichmann vom Forschungszentrum Informatik (FZI) Karlsruhe für die unkomplizierte Unterstützung.

Wie auch bei CrypTool wächst die Qualität des Skripts mit den Anregungen und Verbesserungsvorschlägen von Ihnen als Leser. Wir freuen uns über Ihre Rückmeldung.

Die aktuelle Version von CrypTool finden Sie unter
http://www.CrypTool.de, http://www.CrypTool.com oder http://www.CrypTool.org.

Die Ansprechpartner für dieses kostenlose Open Source-Tool sind in der zum CrypTool-Paket dazugehörenden Readme-Datei genannt.

*Hintergrundmaterial und Zusatzinformationen zum Programm CrypTool

Inhaltsverzeichnis

Vorwort zur 6. Ausgabe des CrypTool-Skripts

Ab dem Jahr 2000 wurde mit dem CrypTool-Paket auch ein Skript ausgeliefert, das die Mathematik einzelner Themen genauer, aber doch möglichst verständlich erläutern sollte.

Um auch hier die Möglichkeit zu schaffen, dass getrennte Entwickler/Autoren mitarbeiten können, wurden die Themen sinnvoll unterteilt und dafür jeweils eigenständig lesbare Kapitel geschrieben. In der anschließenden redaktionellen Arbeit wurden in TeX Querverweise ergänzt und Fußnoten hinzugefügt, die zeigen, an welchen Stellen man die entsprechenden Funktionen in CrypTool aufrufen kann (vgl. den Menübaum im Anhang A). in Anhang A. Natürlich gäbe es viel mehr Themen in Mathematik und Kryptographie, die man vertiefen könnte – deshalb ist diese Auswahl auch nur eine von vielen möglichen.

Die rasante Verbreitung des Internets hat auch zu einer verstärkten Forschung der damit verbundenen Technologien geführt. Gerade im Bereich der Kryptographie gibt es viele neue Erkenntnisse.

Auf den aktuellen Stand gebracht wurden in dieser Ausgabe des Skripts u.a. die Zusammenfassungen zu den Forschungsergebnissen über:

- die Suche nach den größten Primzahlen (verallgemeinerte Mersenne- und Fermatzahlen) (Kap. 2.6),
- Fortschritte in der Zahlentheorie („Primes in P“) (Kap. 3.11.3),
- die Faktorisierung großer Zahlen (TWIRL) (Kap. 3.11.3),
- Fortschritte bei der Kryptoanalyse des AES-Standard (Kap. 1.1.2) und
- Fortschritte bei Brute-Force-Angriffen gegen symmetrische Verfahren (Kap. 1.1.1).

Erweitert wurde im Skript auch das Kapitel 6: hier wird nun genauer erklärt, wie Elliptische Kurven über verschiedenen Zahlkörpern definiert werden.

Seit das Skript dem CrypTool-Paket in Version 1.2.01 das 1. Mal beigelegt wurde, ist es mit jeder neuen Version von CrypTool (1.2.02, 1.3.00, 1.3.02, 1.3.03 und nun 1.3.04) ebenfalls erweitert und aktualisiert worden.

Ich würde mich sehr freuen, wenn sich dies auch innerhalb der ab nun folgenden Open Source-Versionen von CrypTool so weiter fortsetzt (diese werden von der Universität Darmstadt betreut).

Herzlichen Dank nochmal an alle, die mit Ihrem großem Einsatz zum Erfolg und zur weiten Verbreitung dieses Projekts beigetragen haben.

Ich hoffe, dass viele Leser mit diesem Skript mehr Interesse an und Verständnis für dieses moderne und zugleich uralte Thema finden.

Bernhard Esslinger

Frankfurt, März 2003

Einführung – Zusammenspiel von Skript und CrypTool

Das Skript

Dieses Skript wird zusammen mit dem Programm CrypTool ausgeliefert.

Da die Artikel dieses Skripts weitgehend in sich abgeschlossen sind, kann es auch unabhängig von CrypTool gelesen werden.

Während für das Verständnis der meisten Kapitel Abiturwissen ausreicht, dürften Kapitel 4 und 6 tiefere mathematische Kenntnisse erfordern.

Die Autoren haben sich bemüht, Kryptographie für eine möglichst breite Leserschaft darzustellen – ohne mathematisch unkorrekt zu werden. Dieser didaktische Anspruch ist am besten geeignet, die Awareness für die IT-Sicherheit und die Bereitschaft für den Einsatz standardisierter, moderner Kryptographie zu fördern.

Das Programm CrypTool

CrypTool ist ein Programm mit sehr umfangreicher Online-Hilfe, mit dessen Hilfe Sie unter einer einheitlichen Oberfläche kryptographische Verfahren anwenden und analysieren können.

CrypTool wurde im Zuge des End-User Awareness-Programms entwickelt, um die Sensibilität der Mitarbeiter für IT-Sicherheit zu erhöhen und um ein tieferes Verständnis für den Begriff Sicherheit zu ermöglichen.

Ein weiteres Anliegen war die Nachvollziehbarkeit der in der Deutschen Bank eingesetzten kryptographischen Verfahren. So ist es mit CrypTool als verlässlicher Referenzimplementierung der verschiedenen Verschlüsselungsverfahren (aufgrund der Nutzung der Industrie-bewährten Secude-Bibliothek) auch möglich, die in anderen Programmen eingesetzte Verschlüsselung zu testen.

Inzwischen wird CrypTool vielerorts in Ausbildung und Lehre eingesetzt und von mehreren Hochschulen mit weiterentwickelt.

Dank

An dieser Stelle möchte ich explizit drei Personen danken, die bisher in ganz besonderer Weise zu CrypTool beigetragen haben. Ohne ihre besonderen Fähigkeiten und ihr großes Engagement wäre CrypTool nicht, was es heute ist:

- Hr. Henrik Koy
- Hr. Jörg-Cornelius Schneider und
- Dr. Peer Wichmann.

Auch allen hier nicht namentlich Genannten ganz herzlichen Dank für das (meist in der Freizeit) geleistete Engagement.

Bernhard Esslinger

1 Verschlüsselungsverfahren

(Bernhard Esslinger, besslinger@web.de, Mai 1999, Updates: Dez. 2001, Feb. 2003)

Dieses Kapitel soll einen eher beschreibenden Einstieg bieten und die Grundlagen ohne Verwendung von allzuviel Mathematik vermitteln.

Sinn der Verschlüsselung ist es, Daten so zu verändern, dass nur ein autorisierter Empfänger in der Lage ist, den Klartext zu rekonstruieren. Das hat den Vorteil, dass verschlüsselte Daten offen übertragen werden können und trotzdem keine Gefahr besteht, dass ein Angreifer die Daten unberechtigterweise lesen kann. Der autorisierte Empfänger ist im Besitz einer geheimen Information, des sogenannten Schlüssels, die es ihm erlaubt, die Daten zu entschlüsseln, während sie jedem anderen verborgen bleiben.

Es gibt ein beweisbar sicheres Verschlüsselungsverfahren, das *One-Time-Pad*. Dieses weist allerdings einige praktische Nachteile auf (der verwendete Schlüssel muss zufällig gewählt werden und mindestens so lang wie die zu schützende Nachricht sein), so dass es außer in geschlossenen Umgebungen, zum Beispiel beim heißen Draht zwischen Moskau und Washington, kaum eine Rolle spielt.

Für alle anderen Verfahren gibt es (theoretische) Möglichkeiten, sie zu brechen. Bei guten Verfahren sind diese jedoch so aufwändig, dass sie praktisch nicht durchführbar sind und diese Verfahren als (praktisch) sicher angesehen werden können.

Einen sehr guten Überblick zu den Verfahren bietet das Buch von Bruce Schneier [Schneier1996]. Grundsätzlich unterscheidet man zwischen symmetrischen und asymmetrischen Verfahren zur Verschlüsselung.

1.1 Symmetrische Verschlüsselung[1]

Bei der *symmetrischen* Verschlüsselung müssen Sender und Empfänger über einen gemeinsamen (geheimen) Schlüssel verfügen, den sie vor Beginn der eigentlichen Kommunikation ausgetauscht haben. Der Sender benutzt diesen Schlüssel, um die Nachricht zu verschlüsseln und der Empfänger, um diese zu entschlüsseln.

Alle klassischen Verfahren sind von diesem Typ. Beispiele dazu finden Sie in CrypTool oder in [Nichols1996]. Im folgenden wollen wir jedoch nur die moderneren Verfahren betrachten.

Vorteile von symmetrischen Algorithmen sind die hohe Geschwindigkeit, mit denen Daten ver- und entschlüsselt werden. Ein Nachteil ist das Schlüsselmanagement. Um miteinander vertraulich kommunizieren zu können, müssen Sender und Empfänger vor Beginn der eigentlichen Kommunikation über einen sicheren Kanal einen Schlüssel ausgetauscht haben. Spontane Kommunikation zwischen Personen, die sich vorher noch nie begegnet sind, scheint so nahezu unmöglich. Soll in

[1] Mit CrypTool v1.3 können Sie über das Menü **Ver-/Entschlüsseln \ Symmetrisch** folgende modernen symmetrischen Verschlüsselungsverfahren ausführen:
IDEA, RC2, RC4, DES (ECB), DES(CBC), Triple-DES(ECB), Triple-DES(CBC), MARS (AES-Kandidat), RC6 (AES-Kandidat), Serpent (AES-Kandidat), Twofish (AES-Kandidat), Rijndael (offizielles AES-Verfahren).

einem Netz mit n Teilnehmern jeder mit jedem zu jeder Zeit spontan kommunizieren können, so muss jeder Teilnehmer vorher mit jedem anderen der $n-1$ Teilnehmer einen Schlüssel ausgetauscht haben. Insgesamt müssen also $n(n-1)/2$ Schlüssel ausgetauscht werden.

Das bekannteste symmetrische Verschlüsselungsverfahren ist der DES-Algorithmus. Der DES-Algorithmus ist eine Entwicklung von IBM in Zusammenarbeit mit der National Security Agency (NSA). Er wurde 1975 als Standard veröffentlicht. Trotz seines relativ hohen Alters ist jedoch bis heute kein „effektiver“ Angriff auf ihn gefunden worden. Der effektivste Angriff besteht aus dem Durchprobieren (fast) aller möglichen Schlüssel, bis der richtige gefunden wird (*brute-force-attack*). Aufgrund der relativ kurzen Schlüssellänge von effektiv 56 Bits (64 Bits, die allerdings 8 Paritätsbits enthalten), sind in der Vergangenheit schon mehrfach mit dem DES verschlüsselte Nachrichten gebrochen worden, so dass er heute als nur noch bedingt sicher anzusehen ist. Symmetrische Alternativen zum DES sind zum Beispiel die Algorithmen IDEA oder Triple-DES.

Hohe Aktualität besitzen die symmetrischen AES-Verfahren. Das dazu gehörende Rijndael-Verfahren wurde am 2. Oktober 2000 zum Gewinner des AES-Ausschreibens erklärt und ist damit Nachfolger des DES-Verfahrens.

Näheres zu den AES-Algorithmen finden Sie in der CrypTool-Online-Hilfe[2].

1.1.1 Neue Ergebnisse zur Kryptoanalyse von AES

Im Anschluss finden Sie einige Informationen, die das AES-Verfahren in letzter Zeit in Zweifel zogen – unserer Ansicht nach aber (noch) unbegründet. Die folgenden Informationen beruhen auf den unten angegebenen Originalarbeiten und auf [Wobst-iX2002] und [Lucks-DuD2002].

Der AES bietet mit einer Mindestschlüssellänge von 128 Bit gegen Brute-force-Angriffe auch auf längere Sicht genügend Sicherheit – es sei denn, es stünden entsprechend leistungsfähige Quantencomputer zur Verfügung. Der AES war immun gegen alle bis dahin bekannten Kryptoanalyse-Verfahren, die vor allem auf statistischen Überlegungen beruhen und auf DES angewandt wurden: man konstruiert aus Geheim- und Klartextpaaren Ausdrücke, die sich nicht rein zufällig verhalten, sondern Rückschlüsse auf den verwendeten Schlüssel zulassen. Dazu waren aber unrealistische Mengen an abgehörten Daten nötig.

Bei Kryptoverfahren bezeichnet man solche Kryptoanalyseverfahren als "akademischen Erfolg" oder als "kryptoanalytischen Angriff", da sie theoretisch schneller sind als das Durchprobieren aller Schlüssel, das beim Brute-force-Angriff verwendet wird. Im Fall des AES mit maximaler Schlüssellänge (256 Bit) braucht die erschöpfende Schlüsselsuche im Durchschnitt 2^{255} Verschlüsselungsoperationen. Ein kryptoanalytischer Angriff muss diesen Wert unterbieten. Als aktuell gerade noch praktikabel (z.B. für einen Geheimdienst) gilt ein Aufwand von 2^{75} bis 2^{90} Verschlüsselungsoperationen.

Eine neue Vorgehensweise wurde in der Arbeit von Ferguson, Schroeppel und Whiting im Jahre 2001 [Ferguson2001] beschrieben: sie stellten AES als geschlossene Formel (in der Form einer Art Kettenbruch) dar, was aufgrund seiner "relativ" übersichtlichen Struktur gelang. Da diese Formel

[2] CrypTool-Online-Hilfe: das Stichwort **AES** im Index führt auf die 3 Hilfeseiten: **AES-Kandidaten**, **Der AES-Gewinner Rijndael** und **Der AES-Algorithmus Rijndael**.

aus rund 1 Billiarde Summanden besteht, taugt sie zunächst nicht für die Kryptoanalyse. Dennoch war die wissenschaftliche Neugier geweckt. Bereits bekannt war, dass sich der 128-Bit-AES als ein überbestimmtes System von rund 8000 quadratischen Gleichungen (über algebraischen Zahlkörpern) mit rund 1600 Variablen (einige Unbekannte sind die Schlüsselbits) darstellen lässt – solch große Gleichungssysteme sind praktisch nicht lösbar. Dieses Gleichungssystem ist relativ dünn besetzt ("sparse"), d.h. von den insgesamt etwa 1.280.000 möglichen quadratischen Termen tauchen nur relativ wenige überhaupt im Gleichungssystem auf.

Die Mathematiker Courois und Pieprzyk [Courtois2002] veröffentlichten 2002 eine Arbeit, die in der Krypto-Szene stark diskutiert wird: Sie entwickelten das auf der Eurocrypt 2000 von Shamir et al. vorgestellte XL-Verfahren (eXtended Linearization) weiter zum XSL-Verfahren (eXtended Sparse Linearization). Das XL-Verfahren ist eine heuristische Technik, mit der es manchmal gelingt, große nicht-lineare Gleichungssysteme zu lösen und die bei der Analyse eines asymmetrischen Algorithmus (HFE) angewandt wurde. Die Innovation von Courois und Pieprzyk war, das XL-Verfahren auf symmetrische Verfahren anzuwenden: das XSL-Verfahren kann auf spezielle Gleichungssysteme angewandt werden. Damit könnte ein 256-Bit-AES-Verfahren in rund 2^{230} Schritten geknackt werden. Dies ist zwar immer noch ein rein akademischer Angriff, aber er ist richtungsweisend für eine ganze Klasse von Blockchiffren. Das generelle Problem mit diesem Angriff besteht darin, dass man bisher nicht angeben kann, unter welchen Umständen er zum Erfolg führt: die Autoren geben in ihrer Arbeit notwendige Bedingungen an; es ist nicht bekannt, welche Bedingungen hinreichend sind. Neu an diesem Angriff war erstens, dass dieser Angriff nicht auf Statistik, sondern auf Algebra beruhte. Dadurch erscheinen Angriffe möglich, die nur geringe Mengen von Geheimtext brauchen. Zweitens steigt damit die Sicherheit eines Produktalgorithmus nicht mehr exponentiell mit der Rundenzahl.

Aktuell wird sehr intensiv auf diesem Gebiet geforscht: z.B. stellten Murphy und Robshaw auf der Crypto 2002 ein Papier vor [Robshaw2002a], das die Kryptoanalyse drastisch verbessern könnte: der Aufwand für 128-Bit-Schlüssel wurde auf 2^{100} geschätzt, indem sie AES als Spezialfall eines Algorithmus BES (Big Encryption System) darstellten, der eine besonders "runde" Struktur hat. Aber auch 2^{100} Rechenschritte liegen jenseits dessen, was praktisch in absehbarer Zeit realisierbar ist. Bei 256 Bit Schlüssellänge schätzen die Autoren den Aufwand für den XSL-Angriff auf 2^{200} Operationen.

Weitere Details dazu stehen unter:

`http://www.cryptosystem.net/aes`
`http://www.minrank.org/aes/`

Für 256-AES wäre der Angriff ebenfalls viel besser als Brute-force, aber auch noch weiter außerhalb der Reichweite realisierbarer Rechenpower.

Die Diskussion ist z.Zt. sehr kontrovers: Don Coppersmith (einer der DES-Erfinder) z.B. bezweifelt die generelle Durchführbarkeit des Angriffs: XLS liefere für AES gar keine Lösung [Coppersmith2002]. Dann würde auch die Optimierung von Murphy und Robshaw [Robshaw2002b] nicht greifen.

1.1.2 Aktueller Stand der Brute-force-Angriffe auf symmetrische Verfahren (RC5)

Anhand der Blockchiffre RC5 kann der aktuelle Stand von Brute-force-Angriffen auf symmetrische Verschlüsselungsverfahren gut erläutert werden.

Als Brute-force-Angriff (exhaustive search, trial-and-error) bezeichnet man das vollständige Durchsuchen des Schlüsselraumes: dazu müssen keine besonderen Analysetechniken eingesetzt werden. Stattdessen wird der Ciphertext mit allen möglichen Schlüsseln des Schlüsselraums entschlüsselt und geprüft, ob der resultierende Text einen sinnvollen Klartext ergibt. Bei einer Schlüssellänge von 64 Bit sind dies maximal 2^{64} = 18.446.744.073.709.551.616 oder rund 18 Trillionen zu überprüfende Schlüssel[3].

Um zu zeigen, welche Sicherheit bekannte symmetrische Verfahren wie DES, Triple-DES oder RC5 haben, veranstaltet z.B. RSA Security sogenannte Cipher-Challenges[4]. Unter kontrollierten Bedingungen werden Preise ausgelobt, um Ciphertexte, verschlüsselt mit verschiedenen Verfahren und Schlüssellängen, zu entschlüsseln und den symmetrischen Schlüssel zu ermitteln. Damit werden theoretische Resultate praktisch bestätigt.

Dass das „alte“ Standard-Verfahren DES mit der fixen Schlüssellänge von 56 Bit nicht mehr sicher ist, wurde schon im Januar 1999 von der Electronic Frontier Foundation (EFF) demonstriert, als sie mit Deep Crack eine DES-verschlüsselte Nachricht in weniger als einem Tag knackten[5].

Der aktuelle Rekord für starke symmetrische Verfahren liegt bei 64 Bit langen Schlüsseln. Dazu wurde das Verfahren RC5 benutzt, eine Blockchiffre mit variabler Schlüssellänge.

Die RC5-64 Challenge wurde im September 2002 nach 5 Jahren vom distributed.net-Team gelöst[6]. Insgesamt arbeiteten 331.252 Personen gemeinsam über das Internet zusammen. Getestet wurden rund 15 Trillionen Schlüssel, bis der richtige Schlüssel gefunden wurde.

Damit ist gezeigt, dass symmetrische Verfahren (auch wenn sie keinerlei kryptographische Schwächen haben) mit 64 Bit langen Schlüsseln keine geeigneten Verfahren mehr sind, um sensible Daten länger geheim zu halten.

Ähnliche Cipher-Challenges gibt es auch für asymmetrische Verfahren (siehe Kapitel 3.11.4).

1.2 Asymmetrische Verschlüsselung[7]

Bei der *asymmetrischen* Verschlüsselung hat jeder Teilnehmer ein persönliches Schlüsselpaar, das aus einem *geheimen* und einem *öffentlichen* Schlüssel besteht. Der öffentliche Schlüssel wird, der Name deutet es an, öffentlich bekanntgemacht, zum Beispiel in einem Schlüsselverzeichnis im Internet.

---

[3] Mit CrypTool v1.3 können Sie über das Menü **Analyse \ Ciphertext only** ebenfalls Brute-force-Analysen von modernen symmetrischen Verfahren durchführen: um in einer sinnvollen Zeit auf einem Einzel-PC ein Ergebnis zu erhalten, sollten Sie nicht mehr als 20 Bit des Schlüssels als unbekannt kennzeichnen.

[4] http://www.rsasecurity.com/rsalabs/challenges/secretkey/index.html

[5] http://www.rsasecurity.com/rsalabs/challenges/des3/index.html

[6] http://distributed.net/pressroom/news-20020926.html

[7] Mit CrypTool v1.3 können Sie über das Menü **Ver-/Entschlüsseln \ Asymmetrisch** mit RSA ver- und entschlüsseln.

Möchte Alice[8] mit Bob kommunizieren, so sucht sie Bobs öffentlichen Schlüssel aus dem Verzeichnis und benutzt ihn, um ihre Nachricht an ihn zu verschlüsseln. Diesen verschlüsselten Text schickt sie dann an Bob, der mit Hilfe seines geheimen Schlüssels den Text wieder entschlüsseln kann. Da einzig Bob Kenntnis von seinem geheimen Schlüssel hat, ist auch nur er in der Lage, an ihn adressierte Nachrichten zu entschlüsseln. Selbst Alice als Absenderin der Nachricht kann aus der von ihr versandten (verschlüsselten) Nachricht den Klartext nicht wieder herstellen. Natürlich muss sichergestellt sein, dass man aus dem öffentlichen Schlüssel nicht auf den geheimen Schlüssel schließen kann.

Veranschaulichen kann man sich ein solches Verfahren mit einer Reihe von einbruchssicheren Briefkästen. Wenn ich eine Nachricht verfasst habe, so suche ich den Briefkasten mit dem Namensschild des Empfängers und werfe den Brief dort ein. Danach kann ich die Nachricht selbst nicht mehr lesen oder verändern, da nur der legitime Empfänger im Besitz des Schlüssels für den Briefkasten ist.

Vorteil von asymmetrischen Verfahren ist das einfache Schlüsselmanagement. Betrachten wir wieder ein Netz mit n Teilnehmern. Um sicherzustellen, dass jeder Teilnehmer jederzeit eine verschlüsselte Verbindung zu jedem anderen Teilnehmer aufbauen kann, muss jeder Teilnehmer ein Schlüsselpaar besitzen. Man braucht also $2n$ Schlüssel oder n Schlüsselpaare. Ferner ist im Vorfeld einer Übertragung kein sicherer Kanal notwendig, da alle Informationen, die zur Aufnahme einer vertraulichen Kommunikation notwendig sind, offen übertragen werden können. Hier ist lediglich auf die Unverfälschtheit (Integrität und Authentizität) des öffentlichen Schlüssels zu achten. Nachteil: Im Vergleich zu symmetrischen Verfahren sind reine asymmetrische Verfahren jedoch um ein Vielfaches langsamer.

Das bekannteste asymmetrische Verfahren ist der RSA-Algorithmus[9], der nach seinen Entwicklern Ronald Rivest, Adi Shamir und Leonard Adleman benannt wurde. Der RSA-Algorithmus wurde 1978 veröffentlicht. Das Konzept der asymmetrischen Verschlüsselung wurde erstmals von Whitfield Diffie und Martin Hellman in Jahre 1976 vorgestellt. Heute spielen auch die Verfahren nach ElGamal eine bedeutende Rolle, vor allem die Schnorr-Varianten im DSA (Digital Signature Algorithm).

1.3 Hybridverfahren[10]

Um die Vorteile von symmetrischen und asymmetrischen Techniken gemeinsam nutzen zu können, werden (zur Verschlüsselung) in der Praxis meist Hybridverfahren verwendet.

---

[8] Zur Beschreibung kryptographischer Protokolle werden den Teilnehmern oft Namen gegeben (vergleiche [Schneier1996, S. 23]). Alice und Bob führen alle allgemeinen 2-Personen-Protokolle durch, wobei Alice dies initiiert und Bob antwortet. Die Angreifer werden als Eve (eavesdropper = passiver Lauscher) und Mallory (malicious active attacker = böswilliger, aktiver Abgreifer) bezeichnet.

[9] RSA wird in diesem Skript ab Kapitel 3.10 ausführlich beschrieben. Das RSA-Kryptosystem kann mit CrypTool v1.3 über das Menü **Einzelverfahren \ RSA-Demo \ RSA-Kryptosystem** in vielen Variationen nachvollzogen werden. Die aktuellen Forschungsergebnisse im Umfeld von RSA werden in Kapitel 3.11 beschrieben.

[10] In CrypTool v1.3 erhalten Sie über das Menü **Ver-/Entschlüsseln \ Hybrid-Demo** eine Visualisierung dieses Verfahrens: darin werden die einzelnen Schritte und ihre Abhängigkeiten mit konkreten Zahlen gezeigt.

Hier werden die Daten mittels symmetrischer Verfahren verschlüsselt: der Schlüssel ist ein vom Absender zufällig[11] generierter Sitzungsschlüssel (session key), der nur für diese Nachricht verwendet wird. Anschließend wird dieser Sitzungsschlüssel mit Hilfe des asymmetrischen Verfahrens verschlüsselt und zusammen mit der Nachricht an den Empfänger übertragen. Der Empfänger kann den Sitzungsschlüssel mit Hilfe seines geheimen Schlüssels bestimmen und mit diesem dann die Nachricht entschlüsseln. Auf diese Weise nutzt man das bequeme Schlüsselmanagement asymmetrischer Verfahren und kann trotzdem große Datenmengen schnell und effektiv mit symmetrischen Verfahren verschlüsseln.

1.4 Weitere Details

Neben den weiteren Kapiteln in diesem Skript, der umfangreichen Fachliteratur und vielen Stellen im Internet enthält auch die Online-Hilfe von CrypTool sehr viele weitere Informationen zu den einzelnen symmetrischen und asymmetrischen Verschlüsselungsverfahren.

[11] Die Erzeugung zufälliger Zahlen ist ein wichtiger Bestandteil kryptographisch sicherer Verfahren. Mit CrypTool v1.3 können Sie über das Menü **Einzelverfahren \ Zufallsdaten erzeugen** verschiedene Zufallszahlengeneratoren ausprobieren. Über das Menü **Analyse \ Zufallstests** können Sie verschiedene Testverfahren für Zufallsdaten auf binäre Dokumente anwenden.
Bisher konzentriert sich CrypTool auf kryptographisch starke Pseudozufallszahlengeneratoren. Nur im Secude-Generator wird eine „echte“ Zufallsquelle einbezogen.

Literatur

[Nichols1996] Randall K. Nichols,
Classical Cryptography Course, Volume 1 and 2,
Aegean Park Press 1996.

[Schmeh2001] Klaus Schmeh,
Kryptografie und Public-Key Infrastrukturen im Internet,
dpunkt.verlag, 2. akt. und erw. Auflage 2001.
Sehr gut lesbares, hoch aktuelles und umfangreiches Buch über Kryptographie. Geht auch auf praktische Probleme, wie Standardisierung oder real existierende Software, ein.

[Schneier1996] Bruce Schneier,
Applied Cryptography, Protocols, Algorithms, and Source Code in C,
Wiley 1994, 2nd edition 1996.

[Coppersmith2002] Don Coppersmith,
Re: Impact of Courtois and Pieprzyk results,
19.09.2002, in den „AES Discussion Groups“ unter
http://aes.nist.gov/aes/

[Courtois2002] Nicolas Courtois, Josef Pieprzyk,
Cryptanalysis of Block Ciphers with Overdefined Systems of Equations,
received 10 Apr 2002, last revised 9 Nov 2002.
Eine davon verschiedene „compact version“ des ersten XSL-Angriffs wurde auf der Asiacrypt im Dezember 2002 veröffentlicht.
http://eprint.iacr.org/2002/044

[Ferguson2001] Niels Ferguson, Richard Schroeppel, Doug Whiting,
A simple algebraic representation of Rijndael, Draft 1. Mai 2001,
http://www.xs4all.nl/~vorpal/pubs/rdalgeq.html

[Lucks-DuD2002] Stefan Lucks, Rüdiger Weis,
Neue Ergebnisse zur Sicherheit des Verschlüsselungsstandards AES, in DuD 12/2002.

[Robshaw2002a] S.P. Murphy, M.J.B. Robshaw,
Essential Algebraic Structure within the AES, 5. Juni 2002, auf der Crypto 2002,
http://www.isg.rhul.ac.uk/~mrobshaw/rijndael/rijndael.html

[Robshaw2002b] S.P. Murphy, M.J.B. Robshaw,
Comments on the Security of the AES and the XSL Technique, 26. September 2002,
http://www.isg.rhul.ac.uk/~mrobshaw/rijndael/rijndael.html

[Wobst-iX2002] Reinhard Wobst,
Angekratzt - Kryptoanalyse von AES schreitet voran, in iX 12/2002,
und der Leserbrief dazu von Johannes Merkle in der iX 2/2003.

Web-Links

1. Das AES-/Rijndael-Kryptosystem
   http://www.cryptosystem.net/aes
   http://www.minrank.org/aes/

2. „AES Discussion Groups" beim NIST
   http://aes.nist.gov/aes

3. distributed.net: „RC5-64 has been solved"
   http://distributed.net/pressroom/news-20020926.html

4. RSA: „The RSA Secret Key Challenge"
   http://www.rsasecurity.com/rsalabs/challenges/secretkey/index.html

5. RSA: „DES Challenge"
   http://www.rsasecurity.com/rsalabs/challenges/des3/index.html

6. Weiterführende Links auf der CrypTool-Homepage
   http://www.cryptool.de

2 Primzahlen

(Bernhard Esslinger, besslinger@web.de, Mai 1999, Updates: Nov. 2000, Dez. 2001, Feb. 2003)

> *Albert Einstein*[12]*:*
> Der Fortschritt lebt vom Austausch des Wissens.

2.1 Was sind Primzahlen?

Primzahlen sind ganze, positive Zahlen größer gleich 2, die man nur durch 1 und durch sich selbst teilen kann. Alle anderen natürlichen Zahlen größer gleich 2 lassen sich durch Multiplikation von Primzahlen bilden.

Somit bestehen die *natürlichen* Zahlen $\mathbb{N} = \{1, 2, 3, 4, \cdots\}$ aus

- der Zahl 1 (dem Einheitswert)
- den Primzahlen (primes) und
- den zusammengesetzten Zahlen (composite numbers).

Primzahlen haben aus 3 Gründen besondere Bedeutung erlangt:

- Sie werden in der Zahlentheorie als die Grundbausteine der natürlichen Zahlen betrachtet, anhand derer eine Menge genialer mathematischer Überlegungen geführt wurden.
- Sie haben in der modernen Kryptographie (Public Key Kryptographie) große praktische Bedeutung erlangt. Das verbreitetste Public Key Verfahren ist die Ende der siebziger Jahre erfundene RSA-Verschlüsselung. Nur die Verwendung (großer) Primzahlen für bestimmte Parameter garantiert die Sicherheit des Algorithmus sowohl beim RSA-Verfahren als auch bei noch moderneren Verfahren (digitale Signatur, Elliptische Kurven).
- Die Suche nach den größten bekannten Primzahlen hat wohl bisher keine praktische Verwendung, erfordert aber die besten Rechner, gilt als hervorragender Benchmark (Möglichkeit zur Leistungsbestimmung von Computern) und führt zu neuen Formen der Berechnungen auf mehreren Computern
  (siehe auch: http://www.mersenne.org/prime.htm).

Von Primzahlen ließen sich im Laufe der letzten zwei Jahrtausende sehr viele Menschen faszinieren. Der Ehrgeiz, zu neuen Erkenntnissen über Primzahlen zu gelangen, führte dabei oft zu genialen Ideen und Schlussfolgerungen. Im folgenden wird in einer leicht verständlichen Art in die mathematischen Grundlagen der Primzahlen eingeführt. Dabei klären wir auch, was über die Verteilung (Dichte, Anzahl von Primzahl in einem bestimmten Intervall) der Primzahlen bekannt ist oder wie Primzahltests funktionieren.

[12] deutscher Physiker und Nobelpreisträger, 14.03.1879−14.04.1955

2.2 Primzahlen in der Mathematik

Jede ganze Zahl hat Teiler. Die Zahl 1 hat nur einen, nämlich sich selbst. Die Zahl 12 hat die sechs Teiler $1, 2, 3, 4, 6, 12$. Viele Zahlen sind nur teilbar durch sich selbst und durch 1. Bezüglich der Multiplikation sind dies die „Atome“ im Bereich der Zahlen. Diese Zahlen nennt man Primzahlen.

In der Mathematik ist eine etwas andere (aber äquivalente) Definition üblich.

Definition 2.1. *Eine ganze Zahl $p \in \mathbf{N}$ heißt Primzahl , wenn $p > 1$ und p nur die trivialen Teiler ± 1 und $\pm p$ besitzt.*

Per definitionem ist die Zahl 1 keine Primzahl. Im weiteren bezeichnet der Buchstabe p stets eine Primzahl.

Die Primzahlenfolge startet mit

$$2,\ 3,\ 5,\ 7,\ 11,\ 13,\ 17,\ 19,\ 23,\ 29,\ 31,\ 37,\ 41,\ 43,\ 47,\ 53,\ 59,\ 61,\ 67,\ 71,\ 73,\ 79,\ 83,\ 89,\ 97, \cdots .$$

Unter den ersten 100 Zahlen gibt es genau 25 Primzahlen. Danach nimmt ihr prozentualer Anteil stets ab. Primzahlen können nur auf eine einzige *triviale* Weise zerlegt werden:

$$5 = 1 \cdot 5, \quad 17 = 1 \cdot 17, \quad 1013 = 1 \cdot 1.013, \quad 1.296.409 = 1 \cdot 1.296.409.$$

Alle Zahlen, die 2 und mehr von 1 verschiedene Faktoren haben, nennt man *zusammengesetzte* Zahlen. Dazu gehören

$$4 = 2 \cdot 2, \quad 6 = 2 \cdot 3$$

aber auch Zahlen, die *wie Primzahlen aussehen*, aber doch keine sind:

$$91 = 7 \cdot 13, \quad 161 = 7 \cdot 23, \quad 767 = 13 \cdot 59.$$

Satz 2.1. *Jede ganze Zahl m größer als 1 besitzt einen kleinsten Teiler größer als 1. Dieser ist eine Primzahl p. Sofern m nicht selbst eine Primzahl ist, gilt: p ist kleiner oder gleich der Quadratwurzel aus m.*

Aus den Primzahlen lassen sich alle ganzen Zahlen größer als 1 zusammensetzen — und das sogar in einer eindeutigen Weise. Dies besagt der 1. Hauptsatz der Zahlentheorie (= Hauptsatz der elementaren Zahlentheorie = fundamental theorem of arithmetic = fundamental building block of all positive integers).

Satz 2.2. *Jedes Element n größer als 1 der natürlichen Zahlen lässt sich als Produkt $n = p_1 \cdot p_2 \cdot \cdots \cdot p_m$ von Primzahlen schreiben. Sind zwei solche Zerlegungen*

$$n = p_1 \cdot p_2 \cdots p_m = p'_1 \cdot p'_2 \cdots p'_{m'}$$

gegeben, dann gilt nach eventuellem Umsortieren $m = m'$ und für alle i: $p_i = p'_i$.
($p_1, p_2, \ldots, p_m$ nennt man die Primfaktoren von n)

In anderen Worten: Jede natürliche Zahl außer der 1 lässt sich auf genau eine Weise als Produkt von Primzahlen schreiben, wenn man von der Reihenfolge der Faktoren absieht. Die Faktoren sind also eindeutig (die *Expansion in Faktoren* ist eindeutig)! Zum Beispiel ist

$$60 = 2 \cdot 2 \cdot 3 \cdot 5 = 2^2 \cdot 3^1 \cdot 5^1$$

Und das ist — bis auf eine veränderte Reihenfolge der Faktoren — die einzige Möglichkeit, die Zahl 60 in Primfaktoren zu zerlegen. Wenn man nicht nur Primzahlen als Faktoren zulässt, gibt es mehrere Möglichkeiten der Zerlegung in Faktoren und die Eindeutigkeit (uniqueness) geht verloren:

$$60 = 1 \cdot 60 = 2 \cdot 30 = 4 \cdot 15 = 5 \cdot 12 = 6 \cdot 10 = 2 \cdot 3 \cdot 10 = 2 \cdot 5 \cdot 6 = 3 \cdot 4 \cdot 5 = \cdots .$$

Der folgende Absatz wendet sich eher an die mit der mathematischen Logik vertrauteren Menschen: Der 1. Hauptsatz ist nur scheinbar selbstverständlich. Man kann viele andere Zahlenmengen (ungleich der positiven ganzen Zahlen größer als 1) konstruieren, bei denen selbst eine Zerlegung in die Primfaktoren dieser Mengen nicht eindeutig ist: In der Menge $M = \{1, 5, 10, 15, 20, \cdots\}$ gibt es unter der Multiplikation kein Analogon zum Hauptsatz. Die ersten fünf Primzahlen dieser Folge sind $5, 10, 15, 20, 30$ (beachte: 10 ist prim, da innerhalb dieser Menge 5 kein Teiler von 10 ist — das Ergebnis 2 ist kein Element der gegebenen Grundmenge M). Da in M gilt:

$$100 = 5 \cdot 20 = 10 \cdot 10$$

und sowohl $5, 10, 20$ Primzahlen dieser Menge sind, ist hier die Zerlegung in Primfaktoren nicht eindeutig.

2.3 Wie viele Primzahlen gibt es?

Für die natürlichen Zahlen sind die Primzahlen vergleichbar mit den Elementen in der Chemie oder den Elementarteilchen in der Physik (vgl. [Blum1999, S. 22]).

Während es nur 92 natürliche chemische Elemente gibt, ist die Anzahl der Primzahlen unbegrenzt. Das wusste schon der Grieche Euklid[13] im dritten vorchristlichen Jahrhundert.

Satz 2.3 (Euklid[14]). *Die Folge der Primzahlen bricht nicht ab, es gibt also unendlich viele Primzahlen.*

Sein Beweis, dass es unendlich viele Primzahlen gibt, gilt bis heute als ein Glanzstück mathematischer Überlegung und Schlussfolgerung (Widerspruchsbeweis). Er nahm an, es gebe nur endlich

[13] Euklid, griechischer Mathematiker des 4./3. Jahrhunderts vor Christus. Wirkte an der Akademie in Alexandria und verfasste mit den „Elementen“ das bekannteste systematische Lehrbuch der griechischen Mathematik.

[14] Die üblich gewordene Benennung soll nicht sagen, dass Euklid der Entdecker des Satzes ist, da dieser nicht bekannt ist. Der Satz wird bereits in Eulkids „Elementen“ (Buch IX, Satz 20) formuliert und bewiesen. Die dortige Formulierung ist insofern bemerkenswert, als sie das Wort „unendlich“ nicht verwendet; sie lautet

Οί πρῶτοι ἀριθμοὶ πλείους εἰσὶ παντὸς τοῦ προτεθέντος πλήθ ους πρώτων ἀριθμῶν,

zu deutsch: Die Primzahlen sind mehr als jede vorgegebene Menge von Primzahlen.

viele Primzahlen und damit eine größte Primzahl. Daraus zog er solange logische Schlüsse, bis er auf einen offensichtlichen Widerspruch stieß. Damit musste etwas falsch sein. Da sich in die Schlusskette kein Lapsus eingeschlichen hatte, konnte es nur die Annahme sein. Demnach musste es unendlich viele Primzahlen geben!

Euklid's Widerspruchsbeweis führt die Argumentation wie folgt:

Annahme: Es gibt *endlich* viele Primzahlen.

Schluss: Dann lassen sie sich auflisten $p_1 < p_2 < p_3 < \cdots < p_n$, wobei n für die (endliche) Anzahl der Primzahlen steht. p_n wäre also die größte Primzahl. Nun betrachtet Euklid die Zahl $a = p_1 \cdot p_2 \cdots p_n + 1$. Diese Zahl kann keine Primzahl sein, da sie in unserer Primzahlenliste nicht auftaucht. Also muss sie durch eine Primzahl teilbar sein. D.h. es gibt eine natürliche Zahl i zwischen 1 und n, so dass p_i die Zahl a teilt. Natürlich teilt p_i auch das Produkt $a-1 = p_1 \cdot p_2 \cdots p_n$, da p_i ja ein Faktor von $a-1$ ist. Da p_i die Zahlen a und $a-1$ teilt, teilt sie auch die Differenz dieser Zahlen. Daraus folgt: p_i teilt $a-(a-1) = 1$. p_i müsste also 1 teilen und das ist unmöglich.

Widerspruch: Unsere Annahme war falsch.

Also gibt es *unendlich* viele Primzahlen (siehe Übersicht unter 2.8.1 über die Anzahl von Primzahlen in verschiedenen Intervallen).

Wir erwähnen hier auch noch eine andere, auf den ersten Blick überraschende Tatsache, dass nämlich in der Folge aller Primzahlen $p_1, p_2, \cdots$ Lücken von beliebig großer Länge n auftreten. Unter den n aufeinanderfolgenden natürlichen Zahlen

$$(n+1)! + 2, \cdots, (n+1)! + (n+1),$$

ist keine eine Primzahl, da ja in ihnen der Reihe nach die Zahlen $2, \cdots, n+1$ als echte Teiler enthalten sind (Dabei bedeutet $n!$ das Produkt der ersten n natürlichen Zahlen, also $n! = n * (n-1) * \cdots * 3 * 2 * 1$).

2.4 Die Suche nach sehr großen Primzahlen

Die größten heute bekannten Primzahlen haben mehrere hunderttausend Stellen. Das ist unvorstellbar groß. Die Anzahl der Elementarteilchen im Universum wird auf „nur" eine 80-stellige Zahl geschätzt (siehe Übersicht unter 2.8.3 über verschiedene Größenordnungen / Dimensionen).

2.4.1 Spezielle Zahlentypen – Mersennezahlen

Nahezu alle bekannten riesig großen Primzahlen sind spezielle Kandidaten, sogenannte *Mersennezahlen* der Form $2^p - 1$, wobei p eine Primzahl ist. Marin Mersenne (1588-1648) war ein

französischer Priester und Mathematiker. Nicht alle Mersennezahlen sind prim:

$$\begin{aligned} 2^2 - 1 = 3 &\quad \Rightarrow \text{prim} \\ 2^3 - 1 = 7 &\quad \Rightarrow \text{prim} \\ 2^5 - 1 = 31 &\quad \Rightarrow \text{prim} \\ 2^7 - 1 = 127 &\quad \Rightarrow \text{prim} \\ 2^{11} - 1 = 2.047 = 23 \cdot 89 &\quad \Rightarrow \text{NICHT prim!} \end{aligned}$$

Dass Mersennezahlen nicht immer Primzahlen (Mersenne-Primzahlen) sind, wusste auch schon Mersenne (siehe Exponent $p = 11$). Eine Mersennezahl, die prim ist, wird Mersenne-Primzahl genannt.
Dennoch ist ihm der interessante Zusammenhang zu verdanken, dass eine Zahl der Form $2^n - 1$ keine Primzahl ist, wenn n eine zusammengesetzte Zahl ist:

Satz 2.4 (Mersenne). *Wenn $2^n - 1$ eine Primzahl ist, dann folgt, n ist ebenfalls eine Primzahl (oder anders formuliert: $2^n - 1$ ist nur dann prim, wenn n prim ist).*

Beweis
Der Beweis des Satzes von Mersenne kann durch Widerspruch durchgeführt werden. Wir nehmen also an, dass es eine zusammengesetzte natürliche Zahl n (mit echter Zerlegung) $n = n_1 \cdot n_2$ gibt, mit der Eigenschaft, dass $2^n - 1$ eine Primzahl ist.

Wegen

$$\begin{aligned} (x^r - 1)((x^r)^{s-1} + (x^r)^{s-2} + \cdots + x^r + 1) &= ((x^r)^s + (x^r)^{s-1} + (x^r)^{s-2} + \cdots + x^r) \\ &\quad -((x^r)^{s-1} + (x^r)^{s-2} + \cdots + x^r + 1) \\ &= (x^r)^s - 1 = x^{rs} - 1, \end{aligned}$$

folgt

$$2^{n_1 n_2} - 1 = (2^{n_1} - 1)((2^{n_1})^{n_2 - 1} + (2^{n_1})^{n_2 - 2} + \cdots + 2^{n_1} + 1).$$

Da $2^n - 1$ eine Primzahl ist, muss einer der obigen beiden Faktoren auf der rechte Seite gleich 1 sein. Dies kann nur dann der Fall sein, wenn $n_1 = 1$ oder $n_2 = 1$ ist. Dies ist aber ein Widerspruch zu unserer Annahme. Deshalb ist unsere Annahme falsch. Also gibt es keine zusammengesetzte Zahl n, so dass $2^n - 1$ eine Primzahl ist. □

Leider gilt dieser Satz nur in einer Richtung (die Umkehrung gilt nicht, keine Äquivalenz): das heißt, dass es prime Exponenten gibt, für die die zugehörige Mersennezahl **nicht** prim ist (siehe das obige Beispiel $2^{11} - 1$, wo 11 prim ist, aber $2^{11} - 1$ nicht).

Mersenne behauptete, dass $2^{67} - 1$ eine Primzahl ist. Auch zu dieser Behauptung gibt es eine interessante mathematische Historie: Zuerst dauerte es über 200 Jahre, bis Edouard Lucas (1842-1891) bewies, dass diese Zahl zusammengesetzt ist. Er argumentierte aber indirekt und kannte keinen der Faktoren. Dann zeigte Frank Nelson Cole 1903, aus welchen Faktoren diese Primzahl besteht:

$$2^{67} - 1 = 147.573.952.589.676.412.927 = 193.707.721 \cdot 761.838.257.287.$$

Er gestand, 20 Jahre an der Faktorisierung (Zerlegung in Faktoren)[15] dieser 21-stelligen Dezimalzahl gearbeitet zu haben!

Dadurch, dass man bei den Exponenten der Mersennezahlen nicht alle natürlichen Zahlen verwendet, sondern nur die Primzahlen, engt man den *Versuchsraum* deutlich ein. Die derzeit bekannten Mersenne-Primzahlen gibt es für die Exponenten

2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1.279, 2.203, 2.281, 3.217, 4.253, 4.423, 9.689, 9.941, 11.213, 19.937, 21.701, 23.207, 44.497, 86.243, 110.503, 132.049, 216.091, 756.839, 859.433, 1.257.787, 1.398.269, 2.976.221, 3.021.377, 6.972.593, 13.466.917.

Damit sind heute 39 Mersenne-Primzahlen bekannt. Für die ersten 38 Mersenne-Primzahlen weiß man inzwischen, dass diese Liste vollständig ist. Die Exponenten bis zur 39. bekannten Mersenne-Primzahl sind noch nicht vollständig geprüft (vergleiche Kapitel 2.5 Primzahltests).

Die 19. Zahl mit dem Exponenten 4.253 war die erste mit mindestens 1.000 Stellen im Zehnersystem (der Mathematiker Samual Yates prägte dafür den Ausdruck *titanische* Primzahl; sie wurde 1961 von Hurwitz gefunden); die 27. Zahl mit dem Exponenten 44.497 war die erste mit mindestens 10.000 Stellen im Zehnersystem (Yates prägte dafür den Ausdruck *gigantische* Primzahl. Diese Bezeichnungen sind heute längst veraltet).

Man findet diese Zahlen unter den URLs

http://reality.sgi.com/chongo/prime/prime_press.html
http://www.utm.edu/

M-37 – Januar 1998

Die 37. Zahl Mersenne-Primzahl,

$$2^{3.021.377} - 1$$

wurde im Januar 1998 gefunden und hat 909.526 Stellen im Zehnersystem, was 33 Seiten in der FAZ entspricht!

M-38 – Juni 1999

Die 38. Mersenne-Primzahl, genannt M-38,

$$2^{6.972.593} - 1$$

wurde im Juni 1999 gefunden und hat 2.098.960 Stellen im Zehnersystem (das entspricht rund 77 Seiten in der FAZ).

Inzwischen (Stand Feb. 2003) wurden alle Exponenten kleiner als 6.972.593 geprüft, so dass dies wirklich die 38. Mersenne-Primzahl ist. D.h. es ist sicher, dass zwischen der 37. Mersenne-Primzahl und dieser Zahl keine weitere Mersenne-Primzahl existiert.

[15] Mit CrypTool v1.3 können Sie Zahlen auf folgende Weise faktorisieren: Menü **Einzelverfahren \ RSA-Demo \ Faktorisieren einer Zahl**.

M13466917 – Dezember 2001

Diese Zahl wurde als 39. Mersenne-Primzahl gefunden (und schon als M-39 bezeichnet, obwohl noch nicht sicher ist, ob es zwischen M-38 und M13466917 nicht noch eine weitere Mersenne-Primzahl gibt),

$$2^{13.466.917} - 1$$

am 6.12.2001 – genau genommen war am 6.12.2001 die Verifikation der am 14.11.2001 von dem kanadischen Studenten Michael Cameron gefundenen Primzahl abgeschlossen. Diese Zahl hat rund 4 Millionen Stellen (genau 4.053.946 Stellen). Allein zu ihrer Darstellung

$$924947738006701322247758 \cdots 113007385547025625907 1$$

bräuchte man in der FAZ knapp 200 Seiten.

GIMPS

Mit der 39. Mersenne-Primzahl hat das 1996 gegründete GIMPS-Projekt (Great Internet Mersenne-Prime Search) bereits zum 5. Mal die größte Mersenne-Zahl entdeckt, die erwiesenermaßen prim ist (http://www.mersenne.org).

Am GIMPS-Projekt beteiligen sich z.Zt. rund 130.000 freiwillige Amateure und Experten, die ihre Rechner in das von der Firma entropia organisierte „primenet“ einbinden, um mit verteilten Computerprogrammen solche Zahlen zu finden.

Zur Zeit sind die 4 größten bekannten Primzahlen auch Mersenne-Primzahlen. Erst die 5.-größte bekannte Primzahl hat eine andere Form: sie gehört zu den sogenannten verallgemeinerten Fermat-Primzahlen.

2.4.2 EFF

Angefacht wird diese Suche noch zusätzlich durch einen Wettbewerb, den die Nonprofit-Organisation EFF (Electronic Frontier Foundation) mit den Mitteln eines unbekannten Spenders gestartet hat. Den Teilnehmern winken Gewinne im Gesamtwert von 500.000 USD, wenn sie die längste Primzahl finden. Dabei sucht der unbekannte Spender nicht nach dem schnellsten Rechner, sondern er will auf die Möglichkeiten des *cooperative networking* aufmerksam machen:
http://www.eff.org/coopawards/prime-release1.html

Der Entdecker von M-38 erhielt für die Entdeckung der ersten Primzahl mit über 1 Million Dezimalstellen von der EFF eine Prämie von 50.000 USD. Die nächste Prämie von 100.000 USD gibt es von der EFF für eine Primzahl mit mehr als 10 Millionen Dezimalstellen.

Edouard Lucas (1842-1891) hielt über 70 Jahre den Rekord der größten bekannten Primzahl, indem er nachwies, dass $2^{127} - 1$ prim ist. Solange wird wohl kein neuer Rekord mehr Bestand haben.

2.5 Primzahltests

Für die Anwendung sicherer Verschlüsselungsverfahren braucht man sehr große Primzahlen (im Bereich von $2^{2.048}$, das sind Zahlen im Zehnersystem mit über 600 Stellen!).

Bisher haben wir nach den Primfaktoren gesucht, um zu entscheiden, ob eine Zahl prim ist. Wenn aber auch der kleinste Primfaktor riesig ist, dauert die Suche zu lange. Die Zerlegung in Faktoren mittels rechnerischer systematischer Teilung oder mit dem Sieb des Eratosthenes ist mit heutigen Computern anwendbar für Zahlen mit bis zu circa 20 Stellen im Zehnersystem. Die größte Zahl, die bisher in ihre beiden annähernd gleich großen Primfaktoren zerlegt werden konnte, hatte 158 Stellen (vgl. Kapitel 3.11.4).

Ist aber etwas über die *Bauart* (spezielle Struktur) der fraglichen Zahl bekannt, gibt es sehr hochentwickelte Verfahren, die deutlich schneller sind. Diese Verfahren beantworten nur die Primalitätseigenschaft einer Zahl, können aber nicht die Primfaktoren sehr großer zusammengesetzter Zahlen bestimmen.

Fermat hatte im 17. Jahrhundert an Mersenne geschrieben, dass er vermute, dass alle Zahlen der Form

$$F(n) = 2^{2^n} + 1$$

für alle ganzen Zahlen n größer oder gleich 0 prim seien (siehe unten, Kapitel 2.6).

Schon im 19. Jahrhundert wusste man, dass die 29-stellige Zahl

$$F(7) = 2^{2^7} + 1$$

keine Primzahl ist. Aber erst 1970 fanden Morrison/Billhart ihre Zerlegung.

$$\begin{aligned} F(7) &= 340.282.366.920.938.463.463.374.607.431.768.211.457 \\ &= 59.649.589.127.497.217 \cdot 5.704.689.200.685.129.054.721 \end{aligned}$$

Auch wenn sich Fermat bei seiner Vermutung irrte, so stammt in diesem Zusammenhang von ihm doch ein sehr wichtiger Satz: Der (kleine) Fermatsche Satz, den Fermat im Jahr 1640 aufstellte, ist der Ausgangspunkt vieler schneller Primzahltests (siehe Kap. 3.8.3).

Satz 2.5 („kleiner“ Fermat). *Sei p eine Primzahl und a eine beliebige ganze Zahl, dann gilt für alle a*

$$a^p \equiv a \bmod p.$$

Eine alternative Formulierung lautet:
Sei p eine Primzahl und a eine beliebige ganze Zahl, die kein Vielfaches von p ist (also $a \not\equiv 0 \bmod p$), dann gilt $a^{p-1} \equiv 1 \bmod p$.

Wer mit dem Rechnen mit Resten (Modulo-Rechnung) nicht so vertraut ist, möge den Satz einfach so hinnehmen oder Kapitel 3 „Einführung in die elementare Zahlentheorie mit Beispielen“ lesen. Wichtig ist, dass aus diesem Satz folgt, dass wenn diese Gleichheit für irgendein ganzes

a nicht erfüllt ist, dann ist p keine Primzahl! Die Tests lassen sich (zum Beispiel für die erste Formulierung) leicht mit der *Testbasis* $a = 2$ durchführen.

Damit hat man ein Kriterium für Nicht-Primzahlen, also einen negativen Test, aber noch keinen Beweis, dass eine Zahl a prim ist. Leider gilt die Umkehrung zum Fermatschen Satz nicht, sonst hätten wir einen einfachen Beweis für die Primzahleigenschaft (man sagt auch, man hätte dann ein einfaches Primzahlkriterium).

Anmerkung: Zahlen n, die die Eigenschaft

$$2^n \equiv 2 \bmod n$$

erfüllen, aber nicht prim sind, bezeichnet man als *Pseudoprimzahlen*. Die erste Pseudoprimzahl (also keine Primzahl) ist

$$341 = 11 \cdot 31.$$

Es gibt Zahlen, die den Fermat-Test mit allen Basen bestehen und doch nicht prim sind: diese Zahlen heißen *Carmichaelzahlen*. Die erste ist

$$561 = 3 \cdot 11 \cdot 17.$$

Ein stärkerer Test stammt von Miller/Rabin: er wird nur von sogenannten *starken Pseudoprimzahlen* bestanden. Wiederum gibt es starke Pseudoprimzahlen, die keine Primzahlen sind, aber das passiert deutlich seltener als bei den (einfachen) Pseudoprimzahlen. Die kleinste starke Pseudoprimzahl zur Basis 2 ist

$$15.841 = 7 \cdot 31 \cdot 73.$$

Testet man alle 4 Basen $2, 3, 5$ und 7, so findet man bis $25 \cdot 10^9$ nur eine starke Pseudoprimzahl, also eine Zahl, die den Test besteht und doch keine Primzahl ist.

Weiterführende Mathematik hinter dem Rabin-Test gibt dann die Wahrscheinlichkeit an, mit der die untersuchte Zahl prim ist (solche Wahrscheinlichkeiten liegen heutzutage bei circa 10^{-60}).

Ausführliche Beschreibungen zu Tests, um herauszufinden, ob eine Zahl prim ist, finden sich zum Beispiel unter:

http://www.utm.edu/research/primes/mersenne.shtml
http://www.utm.edu/research/primes/prove/index.html

2.6 Weitere Spezial-Zahlentypen und die Suche nach einer Formel für Primzahlen

Derzeit sind keine brauchbaren, offenen (also nicht rekursiven) Formeln bekannt, die nur Primzahlen liefern (rekursiv bedeutet, dass zur Berechnung der Funktion auf dieselbe Funktion in Abhängigkeit einer kleineren Variablen zugegriffen wird). Die Mathematiker wären schon zufrieden, wenn sie eine Formel fänden, die wohl Lücken lässt (also nicht alle Primzahlen liefert), aber sonst keine zusammengesetzten Zahlen (Nicht-Primzahlen) liefert.

Optimal wäre, man würde für das Argument n sofort die n-te Primzahl bekommen, also für $f(8) = 19$ oder für $f(52) = 239$.

Ideen dazu finden sich in

http://www.utm.edu/research/primes/notes/faq/p_n.html.

Die Tabelle unter 2.8.2 enthält die exakten Werte für die n-ten Primzahlen für ausgewählte n.

Die folgende Aufzählung enthält die verbreitetsten Ansätze für "Primzahlformeln":

1. *Mersennezahlen* $f(n) = 2^n - 1$ für n prim:
   Wie oben gesehen, liefert diese Formel wohl relativ viele große Primzahlen, aber es kommt – wie für $n = 11$ $[f(n) = 2.047]$ – immer wieder vor, dass das Ergebnis auch bei primen Exponenten nicht prim ist.
   Heute kennt man alle Mersenne-Primzahlen mit bis zu ca. 2.000.000 Dezimalstellen (M-38) (http://perso.wanadoo.fr/yves.gallot/primes/index.html).

2. $F(k, n) = k \cdot 2^n \pm 1$ für n prim und k kleine Primzahlen:
   Für diese Verallgemeinerung der Mersennezahlen gibt es (für kleine k) ebenfalls sehr schnelle Primzahltests (vgl. [Knuth1981]). Praktisch ausführen lässt sich das zum Beispiel mit der Software Proths von Yves Gallot
   (http://www.prothsearch.net/index.html).

3. *Fermatzahlen*[16] $F(m) = 2^{2^m} + 1$:
   Wie oben erwähnt, schrieb Fermat an Mersenne, dass er vermutet, dass alle Zahlen dieser Form prim seien. Erstaunlicherweise wäre es für ihn möglich gewesen, mit dem auf seinem kleinen Satz beruhenden negativen Primzahltest für $n = 5$ ein positives Ergebnis zu erhalten.

   $$\begin{array}{lll}
   F(0) = 2^{2^0} + 1 = 2^1 + 1 & = 3 & \mapsto \text{ prim} \\
   F(1) = 2^{2^1} + 1 = 2^2 + 1 & = 5 & \mapsto \text{ prim} \\
   F(2) = 2^{2^2} + 1 = 2^4 + 1 & = 17 & \mapsto \text{ prim} \\
   F(3) = 2^{2^3} + 1 = 2^8 + 1 & = 257 & \mapsto \text{ prim} \\
   F(4) = 2^{2^4} + 1 = 2^{16} + 1 & = 65.537 & \mapsto \text{ prim} \\
   F(5) = 2^{2^5} + 1 = 2^{32} + 1 & = 4.294.967.297 = 641 \cdot 6.700.417 & \mapsto \text{ NICHT prim!} \\
   F(6) = 2^{2^6} + 1 = 2^{64} + 1 & = 18.446.744.073.709.551.617 & \\
   & = 274.177 \cdot 67.280.421.310.721 & \mapsto \text{ NICHT prim!} \\
   F(7) = 2^{2^7} + 1 = 2^{128} + 1 & = \text{(siehe S. 23)} & \mapsto \text{ NICHT prim!}
   \end{array}$$

   Innerhalb des Projektes "Distributed Search for Fermat Number Dividers", das von Leonid Durman angeboten wird, gibt es ebenfalls Fortschritte beim Finden von neuen Primzahl-Riesen (http://www.fermatsearch.org/ – diese Webseite hat Verknüpfungen zu Seiten in russisch, italienisch und deutsch).

   Die entdeckten Faktoren können sowohl zusammengesetzte natürliche als auch prime natürliche Zahlen sein.

[16] Die Fermatschen Primzahlen spielen unter anderem eine wichtige Rolle in der Kreisteilung. Wie Gauss bewiesen hat, ist das reguläre p-Eck für eine Primzahl $p > 2$ dann und nur dann mit Zirkel und Lineal konstruierbar, wenn p eine Fermatsche Primzahl ist.

Am 22. Februar 2003 entdeckte John Cosgrave

- die größte bis dahin bekannte zusammengesetzte Fermatzahl und
- die größte bis dahin bekannte prime Nicht-Mersennezahl mit 645.817 Dezimalstellen.

Die Fermatzahl

$$F[2.145.351] = 2^{(2^{2.145.351})} + 1$$

ist teilbar durch die Primzahl

$$p = 3 * 2^{2.145.353} + 1$$

Diese Primzahl p ist die größte bis dahin bekannte prime verallgemeinerte Mersennezahl. Sie wurde nur ein paar Tage später entdeckt, nachdem Michael Angel die größte bis dahin bekannte prime verallgemeinerte Fermatzahl (siehe unten) entdeckte. Dadurch wurde die erste im GIMPS-Projekt gefundene Mersenne-Primzahl M-35 = M1398269 auf den 7. Platz verwiesen.

Zu diesem Erfolg trugen bei: NewPGen von Paul Jobling's, PRP von George Woltman's, Proth von Yves Gallot's Programm und die Proth-Gallot-Gruppe am St. Patrick's College, Dublin.

Weitere Details finden sich unter
http://www.fermatsearch.org/history/cosgrave_record.htm/

4. *Verallgemeinerte Fermatzahlen*[17] $F(b, m) = b^{2^m} + 1$:
Verallgemeinerte Fermatzahlen kommen häufiger vor als Mersennezahlen gleicher Größe, so dass wahrscheinlich noch viele gefunden werden können, die die großen Lücken zwischen den Mersenne-Primzahlen verkleinern. Fortschritte in der Zahlentheorie haben es ermöglicht, dass Zahlen, deren Repräsentation nicht auf eine Basis von 2 beschränkt ist, nun mit fast der gleichen Geschwindigkeit wie Mersennezahlen getestet werden können.

   Yves Gallot schrieb das Programm Proth.exe für die Untersuchung verallgemeinerter Fermatzahlen.

   Mit diesem Programm fand Michael Angel am 16. Februar 2003 eine prime verallgemeinerte Fermatzahl mit 628.808 Dezimalstellen, die zum damaligen Zeitpunkt zur 5.-größten bis dahin bekannten Primzahl wurde:

$$b^{2^{17}} + 1 = 62.722^{131.072} + 1.$$

   Dadurch wurde die erste im GIMPS-Projekt gefundene Mersenne-Primzahl

$$\text{M-35} = \text{M1398269}$$

   auf den 6. Platz verwiesen.

   Weitere Details finden sich unter

[17] Hier ist die Basis b nicht notwendigerweise 2 !

http://www.prothsearch.net/index.html
http://perso.wanadoo.fr/yves.gallot/primes/index.html

5. Carmichaelzahlen: s.o.

6. Pseudoprimzahlen: s.o.

7. starke Pseudoprimzahlen: s.o.

8. Idee aufgrund von Euklids Beweis (unendlich viele Primzahlen) $p_1 \cdot p_2 \cdots p_n + 1$:

$$\begin{array}{lll} 2{\cdot}3+1 & = 7 & \mapsto \text{ prim} \\ 2{\cdot}3{\cdot}5+1 & = 31 & \mapsto \text{ prim} \\ 2{\cdot}3{\cdot}5{\cdot}7+1 & = 211 & \mapsto \text{ prim} \\ 2{\cdot}3{\cdot}\cdots 11+1 & = 2311 & \mapsto \text{ prim} \\ 2\cdot 3\cdots 13+1 & = 59\cdot 509 & \mapsto \text{ NICHT prim!} \\ 2\cdot 3\cdots 17+1 & = 19\cdot 97\cdot 277 & \mapsto \text{ NICHT prim!} \end{array}$$

9. Wie oben, nur statt +1: $p_1 \cdot p_2 \cdots p_n - 1$

$$\begin{array}{lll} 2\cdot 3-1 & = 5 & \mapsto \text{ prim} \\ 2\cdot 3\cdot 5-1 & = 29 & \mapsto \text{ prim} \\ 2\cdot 3\cdots 7-1 & = 11\cdot 19 & \mapsto \text{ NICHT prim!} \\ 2\cdot 3\cdots 11-1 & = 2309 & \mapsto \text{ prim} \\ 2\cdot 3\cdots 13-1 & = 30029 & \mapsto \text{ prim} \\ 2\cdot 3\cdots 17-1 & = 61\cdot 8369 & \mapsto \text{ NICHT prim!} \end{array}$$

10. *Euklidzahlen* $e_n = e_0 \cdot e_1 \cdots e_{n-1} + 1$ mit n größer oder gleich 1 und $e_0 := 1$.
e_{n-1} ist nicht die $(n-1)$-te Primzahl, sondern die zuvor hier gefundene Zahl. Diese Formel ist leider nicht offen, sondern rekursiv. Die Folge startet mit

$$\begin{array}{lll} e_1 = 1+1 & = 2 & \mapsto \text{ prim} \\ e_2 = e_1+1 & = 3 & \mapsto \text{ prim} \\ e_3 = e_1\cdot e_2+1 & = 7 & \mapsto \text{ prim} \\ e_4 = e_1\cdot e_2\cdot e_3+1 & = 43 & \mapsto \text{ prim} \\ e_5 = e_1\cdot e_2\cdots e_4+1 & = 13\cdot 139 & \mapsto \text{ NICHT prim!} \\ e_6 = e_1\cdot e_2\cdots e_5+1 & = 3.263.443 & \mapsto \text{ prim} \\ e_7 = e_1\cdot e_2\cdots e_6+1 & = 547\cdot 607\cdot 1.033\cdot 31.051 & \mapsto \text{ NICHT prim!} \\ e_8 = e_1\cdot e_2\cdots e_7+1 & = 29.881\cdot 67.003\cdot 9.119.521\cdot 6.212.157.481 & \mapsto \text{ NICHT prim!} \end{array}$$

Auch $e_9, \cdots, e_{17}$ sind zusammengesetzt, so dass dies auch keine brauchbare Formel ist. Bemerkung: Das Besondere an diesen Zahlen ist, dass sie jeweils paarweise keinen gemeinsamen Teiler außer 1 haben, sie sind also *relativ zueinander prim*.

11. $f(n) = n^2 + n + 41$:
Diese Folge hat einen sehr *erfolgversprechenden* Anfang, aber das ist noch lange kein Beweis.

$$\begin{array}{ll}
f(0) = 41 & \mapsto \text{ prim} \\
f(1) = 43 & \mapsto \text{ prim} \\
f(2) = 47 & \mapsto \text{ prim} \\
f(3) = 53 & \mapsto \text{ prim} \\
f(4) = 61 & \mapsto \text{ prim} \\
f(5) = 71 & \mapsto \text{ prim} \\
f(6) = 83 & \mapsto \text{ prim} \\
f(7) = 97 & \mapsto \text{ prim} \\
\vdots & \\
f(39) = 1.601 & \mapsto \text{ prim} \\
f(40) = 11 \cdot 151 & \mapsto \text{ NICHT prim!} \\
f(41) = 41 \cdot 43 & \mapsto \text{ NICHT prim!}
\end{array}$$

Die ersten 40 Werte sind Primzahlen (diese haben die auffallende Regelmäßigkeit, dass ihr Abstand beginnend mit dem Abstand 2 jeweils um 2 wächst), aber der 41. und der 42. Wert sind keine Primzahlen. Dass $f(41)$ keine Primzahl sein kann, lässt sich leicht überlegen: $f(41) = 41^2 + 41 + 41 = 41(41 + 1 + 1) = 41 \cdot 43$.

12. $f(n) = n^2 - 79 \cdot n + 1.601$:
Diese Funktion liefert für die Werte $n = 0$ bis $n = 79$ stets Primzahlwerte. Leider ergibt $f(80) = 1.681 = 11 \cdot 151$ keine Primzahl. Bis heute kennt man keine Funktion, die mehr aufeinanderfolgende Primzahlen annimmt. Andererseits kommt jede Primzahl doppelt vor (erst in der absteigenden, dann in der aufsteigenden Folge), so dass sie insgesamt 40 verschiedene Primzahlwerte liefert (dieselben wie die Funktion aus Punkt 11).

$f(0) = 1.601$	$\mapsto$	prim	$f(28) = 173$	$\mapsto$	prim
$f(1) = 1.523$	$\mapsto$	prim	$f(29) = 151$	$\mapsto$	prim
$f(2) = 1.447$	$\mapsto$	prim	$f(30) = 131$	$\mapsto$	prim
$f(3) = 1.373$	$\mapsto$	prim	$f(31) = 113$	$\mapsto$	prim
$f(4) = 1.301$	$\mapsto$	prim	$f(32) = 97$	$\mapsto$	prim
$f(5) = 1.231$	$\mapsto$	prim	$f(33) = 83$	$\mapsto$	prim
$f(6) = 1.163$	$\mapsto$	prim	$f(34) = 71$	$\mapsto$	prim
$f(7) = 1.097$	$\mapsto$	prim	$f(35) = 61$	$\mapsto$	prim
$f(8) = 1.033$	$\mapsto$	prim	$f(36) = 53$	$\mapsto$	prim
$f(9) = 971$	$\mapsto$	prim	$f(37) = 47$	$\mapsto$	prim
$f(10) = 911$	$\mapsto$	prim	$f(38) = 43$	$\mapsto$	prim
$f(11) = 853$	$\mapsto$	prim	$f(39) = 41$	$\mapsto$	prim
$f(12) = 797$	$\mapsto$	prim	$f(40) = 41$	$\mapsto$	prim
$f(13) = 743$	$\mapsto$	prim	$f(41) = 43$	$\mapsto$	prim
$f(14) = 691$	$\mapsto$	prim	$f(42) = 47$	$\mapsto$	prim
$f(15) = 641$	$\mapsto$	prim	$f(43) = 53$	$\mapsto$	prim
$f(16) = 593$	$\mapsto$	prim	$\cdots$		
$f(17) = 547$	$\mapsto$	prim	$f(77) = 1.447$	$\mapsto$	prim
$f(18) = 503$	$\mapsto$	prim	$f(78) = 1.523$	$\mapsto$	prim
$f(19) = 461$	$\mapsto$	prim	$f(79) = 1.601$	$\mapsto$	prim
$f(20) = 421$	$\mapsto$	prim	$f(80) = 11 \cdot 151$	$\mapsto$	NICHT prim!
$f(21) = 383$	$\mapsto$	prim	$f(81) = 41 \cdot 43$	$\mapsto$	NICHT prim!
$f(22) = 347$	$\mapsto$	prim	$f(82) = 1.847$	$\mapsto$	prim
$f(21) = 383$	$\mapsto$	prim	$f(83) = 1.933$	$\mapsto$	prim
$f(22) = 347$	$\mapsto$	prim	$f(84) = 43 \cdot 47$	$\mapsto$	NICHT prim!
$f(23) = 313$	$\mapsto$	prim			
$f(24) = 281$	$\mapsto$	prim			
$f(25) = 251$	$\mapsto$	prim			
$f(26) = 223$	$\mapsto$	prim			
$f(27) = 197$	$\mapsto$	prim			

13. Polynomfunktionen $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0$ (a_i aus $\mathbb{Z}$, $n \geq 1$):

Es existiert kein solches Polynom, das für alle x aus $\mathbb{Z}$ ausschließlich Primzahlwerte annimmt. Zum Beweis sei auf [Padberg1996, S. 83 f.], verwiesen, wo sich auch weitere Details zu Primzahlformeln finden.

Damit ist es hoffnungslos, weiter nach Formeln vom Typ 11. und 12. zu suchen.

14. Catalan, nach dem auch die sog. *Catalanzahlen* $A(n) = (1/(n+1)) * (2n)!/(n!)^2$ benannt sind, äußerte die Vermutung, dass C_4 eine Primzahl ist:

$$\begin{aligned} C_0 &= 2, \\ C_1 &= 2^{C_0} - 1, \\ C_2 &= 2^{C_1} - 1, \\ C_3 &= 2^{C_2} - 1, \\ C_4 &= 2^{C_3} - 1, \cdots \end{aligned}$$

(siehe http://www.utm.edu/research/primes/mersenne.shtml unter Conjectures and Unsolved Problems).

Diese Folge ist ebenfalls rekursiv definiert und wächst sehr schnell. Besteht sie nur aus Primzahlen?

$$\begin{array}{lll} C(0) = 2 & & \mapsto \text{ prim} \\ C(1) = 2^2 - 1 & = 3 & \mapsto \text{ prim} \\ C(2) = 2^3 - 1 & = 7 & \mapsto \text{ prim} \\ C(3) = 2^7 - 1 & = 127 & \mapsto \text{ prim} \\ C(4) = 2^{127} - 1 & = 170.141.183.460.469.231.731.687.303.715.884.105.727 & \mapsto \text{ prim} \end{array}$$

Ob C_5 bzw. höhere Elemente prim sind, ist (noch) nicht bekannt, aber auch nicht wahrscheinlich. Bewiesen ist jedenfalls nicht, dass diese Formel nur Primzahlen liefert.

2.7 Dichte und Verteilung der Primzahlen

Wie Euklid herausfand, gibt es unendlich viele Primzahlen. Einige unendliche Mengen sind aber *dichter* als andere. Innerhalb der natürlichen Zahlen gibt es unendlich viele gerade, ungerade und quadratische Zahlen.

Nach folgenden Gesichtspunkten gibt es mehr gerade Zahlen als quadratische:

- die Größe des n-ten Elements:
  Das n-te Element der geraden Zahlen ist $2n$; das n-te Element der Quadratzahlen ist n^2. Weil für alle $n > 2$ gilt: $2n < n^2$, kommt die n-te gerade Zahl viel früher als die n-te quadratische Zahl. Daher sind die geraden Zahlen dichter verteilt, und wir können sagen, es gibt mehr gerade als quadratische Zahlen.
- die Anzahl der Werte, die kleiner oder gleich einem bestimmten *Dachwert* x aus $\mathbb{R}$ ist:
  Es gibt $[x/2]$ solcher gerader Zahlen und $[\sqrt{x}]$ Quadratzahlen. Da für große x der Wert $x/2$ viel größer ist als die Quadratwurzel aus 2, können wir wiederum sagen, es gibt mehr gerade Zahlen.

Satz 2.6. *Für große n gilt: Der Wert der n-ten Primzahl $P(n)$ ist asymptotisch zu $n \cdot ln(n)$, d.h. der Grenzwert des Verhältnisses $P(n)/(n \cdot \ln n)$ ist gleich 1, wenn n gegen unendlich geht.*

Ähnlich wird die Anzahl der Primzahlen $PI(x)$ definiert, die den Dachwert x nicht übersteigen:

Satz 2.7. *$PI(x)$ ist asymptotisch zu $x/ln(x)$.*

Dies ist der **Primzahlsatz** (prime number theorem). Er wurde von Legendre (1752-1833) und Gauss (1777-1855) aufgestellt und erst über 100 Jahre später bewiesen.

(Siehe Übersicht unter 2.8.1 über die Anzahl von Primzahlen in verschiedenen Intervallen).

Es gilt für große n, dass $P(n)$ zwischen $2n$ und n^2 liegt. Somit gibt es also weniger Primzahlen als gerade natürliche Zahlen, aber es gibt mehr Primzahlen als Quadratzahlen.

Diese Formeln, die nur für n gegen unendlich gelten, können durch präzisere Formeln ersetzt werden. Für $x \geq 67$ gilt:

$$ln(x) - 1,5 < x/PI(x) < ln(x) - 0,5$$

Im Bewusstsein, dass $PI(x) = x/\ln x$ nur für sehr große x (x gegen unendlich) gilt, kann man folgende Übersicht erstellen:

x	$ln(x)$	$x/ln(x)$	$PI(x)(gezählt)$	$PI(x)/(x/ln(x))$
10^3	$6,908$	144	168	$1,160$
10^6	$13,816$	72.386	78.498	$1,085$
10^9	$20,723$	48.254.942	50.847.534	$1,054$

Für eine Binärzahl (Zahl im Zweiersystem) der Länge 250 Bit (2^{250} ist ungefähr $= 1,809251 * 10^{75}$) gilt:

$$PI(250) = 2^{250}/(250 \cdot \ln 2) \text{ ist ungefähr } = 2^{250}/173,28677 = 1,045810 \cdot 10^{73}.$$

Es ist also zu erwarten, dass sich innerhalb der Zahlen der Bitlänge kleiner als 250 Zahlen ungefähr 10^{73} Primzahlen befinden (ein beruhigendes Ergebnis?!).

Man kann das auch so formulieren: Betrachtet man eine *zufällige* natürliche Zahl n, so sind die Chancen, dass diese Zahl prim ist, circa $1/\ln(n)$. Nehmen wir zum Beispiel Zahlen in der Gegend von 10^{16}, so müssen wir ungefähr (durchschnittlich) $16 \cdot \ln 10 = 36,8$ Zahlen betrachten, bis wir eine Primzahl finden. Ein genaue Untersuchung zeigt: Zwischen $10^{16} - 370$ und $10^{16} - 1$ gibt es 10 Primzahlen.

Unter der Überschrift *How Many Primes Are There* finden sich unter

`http://www.utm.edu/research/primes/howmany.shtml`

viele weitere Details.

$PI(x)$ lässt sich leicht per

`http://www.math.Princeton.EDU/~arbooker/nthprime.html`

bestimmen.

Die **Verteilung** der Primzahlen weist viele Unregelmäßigkeiten auf, für die bis heute kein System gefunden wurde: einerseits liegen viele eng benachbart wie 2 und 3, 11 und 13, 809 und 811, andererseits tauchen auch längere Primzahllücken auf. So liegen zum Beispiel zwischen 113 und 127, 293 und 307, 317 und 331, 523 und 541, 773 und 787, 839 und 853 sowie zwischen 887 und 907 keine Primzahlen.
Details siehe:

http://www.utm.edu/research/primes/notes/gaps.html

Gerade dies macht einen Teil des Ehrgeizes der Mathematiker aus, ihre Geheimnisse herauszufinden.

Sieb des Eratosthenes

Ein einfacher Weg, alle $PI(x)$ Primzahlen kleiner oder gleich x zu berechnen, ist das Sieb des Erathostenes. Er fand schon im 3. Jahrhundert vor Christus einen sehr einfach automatisierbaren Weg, das herauszufinden. Zuerst werden ab 2 alle Zahlen bis x aufgeschrieben, die 2 umkreist und dann streicht man alle Vielfachen von 2. Anschließend umkreist man die kleinste noch nicht umkreiste oder gestrichene Zahl (3), streicht wieder alle ihre Vielfachen, usw. Durchzuführen braucht man das nur bis zu der größten Zahl, deren Quadrat kleiner oder gleich x ist.

Abgesehen von 2 sind Primzahlen nie gerade. Abgesehen von 2 und 5 haben Primzahlen nie die Endziffern 2, 5 oder 0. Also braucht man sowieso nur Zahlen mit den Endziffern 1, 3, 7, 9 zu betrachten (es gibt unendlich viele Primzahlen mit jeder dieser letzten Ziffern; vergleiche [Tietze1973, Bd. 1, S. 137]).

Inzwischen findet man im Internet auch viele fertige Programme, oft mit komplettem Quellcode, so dass man auch selbst mit großen Zahlen experimentieren kann. Ebenfalls zugänglich sind große Datenbanken, die entweder viele Primzahlen oder die Zerlegung in Primfaktoren vieler zusammengesetzter Zahlen enthalten. Im **Cunningham-Projekt** zum Beispiel werden die Faktoren aller zusammengesetzten Zahlen bestimmt, die sich in folgender Weise bilden:

$$f(n) = b^n \pm 1 \quad \text{für } b = 2, 3, 5, 6, 7, 10, 11, 12$$

(b ist ungleich der Vielfachen von schon benutzten Basen wie $4, 8, 9$).

Details hierzu finden sich unter:

http://www.cerias.purdue.edu/homes/ssw/cun

2.8 Anmerkungen

Bewiesene Aussagen / Sätze zu Primzahlen

- Zu jeder Zahl n aus $\mathbf{N}$ gibt es n aufeinanderfolgende natürliche Zahlen, die keine Primzahlen sind. Ein Beweis findet sich in [Padberg1996, S. 79].
- Der ungarische Mathematiker Paul Erdös (1913-1996) bewies: Zwischen jeder beliebigen Zahl ungleich 1 und ihrem Doppelten gibt es mindestens eine Primzahl. Er bewies das Theorem nicht als erster, aber auf einfachere Weise als andere vor ihm. Gerade diese Vermutung legt nahe, dass wir auch heute noch nicht in aller Tiefe den Zusammenhang zwischen der Addition und der Multiplikation der natürlichen Zahlen verstanden haben.
- Es existiert eine reelle Zahl a, so dass die Funktion $f : \mathbf{N} \to \mathbb{Z}$ mit $n \mapsto a^{3^n}$ für alle n nur Primzahlenwerte annimmt (siehe [Padberg1996, S. 82]). Leider macht die Bestimmung von a Probleme (siehe unten).

Unbewiesene Aussagen / Vermutungen zu Primzahlen

- Der deutsche Mathematiker Christian Goldbach (1690-1764) vermutete: Jede gerade natürliche Zahl größer 2 lässt sich als die Summe zweier Primzahlen darstellen. Mit Computern ist die Goldbachsche Vermutung für alle geraden Zahlen bis $4*10^{14}$ verifiziert[18], aber allgemein noch nicht bewiesen[19].
- Der deutsche Mathematiker Bernhard Riemann (1826-1866) stellte eine bisher unbewiesene, aber auch nicht widerlegte (?) Formel für die Verteilung von Primzahlen auf, die die Abschätzung weiter verbessern würde.

Offene Fragestellungen

Primzahlzwillinge sind Primzahlen, die den Abstand 2 voneinander haben, zum Beispiel 5 und 7 oder 101 und 103. Primzahldrillinge gibt es dagegen nur eines: 3, 5, 7. Bei allen anderen Dreierpacks aufeinanderfolgender ungerader Zahlen ist immer eine durch 3 teilbar und somit keine Primzahl.

- Offen ist die Anzahl der Primzahlzwillinge: unendlich viele oder eine begrenzte Anzahl? Das bis heute bekannte größte Primzahlzwillingspaar ist $1.693.965 \cdot 2^{66.443} \pm 1$.
- Gibt es eine Formel für die Anzahl der Primzahlzwillinge pro Intervall?

---

[18] Dass die Goldbachsche Vermutung wahr ist, d.h. für alle geraden natürlichen Zahlen größer als 2 gilt, wird heute allgemein nicht mehr angezweifelt. Der Mathematiker Jörg Richstein vom Institut für Informatik der Universität Gießen hat 1999 die geraden Zahlen bis 400 Billionen untersucht und kein Gegenbeispiel gefunden (Siehe http://www.informatik.uni-giessen.de/staff/richstein/de/Goldbach.html). Trotzdem ist das kein allgemeiner Beweis.
Dass die Goldbachsche Vermutung trotz aller Anstrengungen bis heute nicht bewiesen wurde, fördert allerdings einen Verdacht: Seit den bahnbrechenden Arbeiten des österreichischen Mathematikers Kurt Gödel ist bekannt, dass nicht jeder wahre Satz in der Mathematik auch beweisbar ist (Siehe http://www.mathematik.ch/mathematiker/goedel.html). Möglicherweise hat Goldbach also Recht, und trotzdem wird nie ein Beweis gefunden werden. Das wiederum lässt sich aber vermutlich auch nicht beweisen.

[19] Der englische Verlag *Faber* und die amerikanische Verlagsgesellschaft *Bloomsbury* publizierten 2000 das 1992 erstmal veröffentlichte Buch „Onkel Petros und die Goldbachsche Vermutung“ von Apostolos Doxiadis (deutsch bei Lübbe 2000 und bei BLT als Taschenbuch 2001). Es ist die Geschichte eines Mathematikprofessors, der daran scheitert, ein mehr als 250 Jahre altes Rätsel zu lösen.
Um die Verkaufszahlen zu fördern, schreiben die beiden Verlage einen Preis von 1 Million USD aus, wenn jemand die Vermutung beweist – veröffentlicht in einer angesehenen mathematischen Fachzeitschrift bis 2004 (Siehe http://www.mscs.dal.ca/~dilcher/Goldbach/index.html).
Erstaunlicherweise dürfen nur englische und amerikanische Mathematiker daran teilnehmen.
Der Beweis, der der Goldbach-Vermutung bisher am nächsten kommt, wurde 1966 von Chen Jing-Run bewiesen – in einer schwer nachvollziehbaren Art und Weise: Jede gerade Zahl größer 2 ist die Summe einer Primzahl und des Produkts zweier Primzahlen. Z.B. $20 = 5 + 3 * 5$.
Die wichtigsten Forschungsergebnisse zur Goldbachschen Vermutung sind zusammengefasst in dem von Wang Yuan herausgegebenen Band: „Goldbach Conjecture“, 1984, World Scientific Series in Pure Maths, Vol. 4.
Gerade diese Vermutung legt nahe, dass wir auch heute noch nicht in aller Tiefe den Zusammenhang zwischen der Addition und der Multiplikation der natürlichen Zahlen verstanden haben.

- Der Beweis oben zu der Funktion $f : N \to Z$ mit $n \mapsto a^{3^n}$ garantiert nur die Existenz einer solchen Zahl a. Wie kann diese Zahl a bestimmt werden, und wird sie einen Wert haben, so dass die Funktion auch von praktischem Interesse ist?
- Gibt es unendlich viele Mersenne-Primzahlen?
- Gibt es unendlich viele Fermatsche Primzahlen?
- Gibt es einen Polynomialzeit-Algorithmus zur Zerlegung einer Zahl in ihre Primfaktoren (vgl. [Klee1997, S. 167])? Diese Frage kann man auf die beiden folgenden Fragestellungen aufsplitten:
  - Gibt es einen Polynomialzeit-Algorithmus, der entscheidet, ob eine Zahl prim ist?
  - Gibt es einen Polynomialzeit-Algorithmus, mit dem sich für eine zusammengesetzte Zahl n ein nicht-trivialer (d.h. von 1 und von n verschiedener) Teiler von n berechnen lässt?[20]

Weitere interessante Themen rund um Primzahlen Nicht betrachtet wurden in diesem Kapitel folgende eher zahlentheoretische Themen wie Teilbarkeitsregeln, Modulo-Rechnung, modulare Inverse, modulare Potenzen und Wurzeln, chinesischer Restesatz, Eulersche Phi-Funktion, perfekte Zahlen. Auf einige dieser Themen geht das nächste Kapitel ein.

[20] Vergleiche auch Kapitel 3.11.3 und Kapitel 3.11.4.

2.8.1 Anzahl von Primzahlen in verschiedenen Intervallen

Zehnerintervall		Hunderterintervall		Tausenderintervall	
Intervall	Anzahl	Intervall	Anzahl	Intervall	Anzahl
1-10	4	1-100	25	1-1.000	168
11-20	4	101-200	21	1.001-2.000	135
21-30	2	201-300	16	2.001-3.000	127
31-40	2	301-400	16	3.001-4.000	120
41-50	3	401-500	17	4.001-5.000	119
51-60	2	501-600	14	5.001-6.000	114
61-70	2	601-700	16	6.001-7.000	117
71-80	3	701-800	14	7.001-8.000	107
81-90	2	801-900	15	8.001-9.000	110
91-100	1	901-1.000	14	9.001-10.000	112

Weitere Intervalle:

Intervall	Anzahl	Durchschnittl. Anzahl pro 1000
1 - 10.000	1.229	122,900
1 - 100.000	9.592	95,920
1 - 1.000.000	78.498	78,498
1 - 10.000.000	664.579	66,458
1 - 100.000.000	5.761.455	57,615
1 - 1.000.000.000	50.847.534	50,848
1 - 10.000.000.000	455.052.512	45,505

2.8.2 Indizierung von Primzahlen (n-te Primzahl)

Index	Genauer Wert	Gerundeter Wert	Bemerkung
1	2	2	
2	3	3	
3	5	5	
4	7	7	
5	11	11	
6	13	13	
7	17	17	
8	19	19	
9	23	23	
10	29	29	
100	541	541	
1000	7917	7917	
664.559	9.999.991	9,99999E+06	Alle Primzahlen bis zu 1E+07 waren am Beginn des 20. Jahrhunderts bekannt.
1E+06	15.485.863	1,54859E+07	
6E+06	104.395.301	1,04395E+08	Diese Primzahl wurde 1959 entdeckt.
1E+07	179.424.673	1,79425E+08	
1E+09	22.801.763.489	2,28018E+10	
1E+12	29.996.224.275.833	2,99962E+13	

Bemerkung: Mit Lücke wurden früh sehr große Primzahlen entdeckt.

Web-Links (URLs):
http://www.math.Princeton.EDU/~arbooker/nthprime.html.

Ausgabe der n-ten Primzahl
Vergleiche `http://www.utm.edu/research/primes/notes/by_year.html`.

2.8.3 Größenordnungen / Dimensionen in der Realität

Bei der Beschreibung kryptographischer Protokolle und Algorithmen treten Zahlen auf, die so groß bzw. so klein sind, dass sie einem intuitiven Verständnis nicht zugänglich sind. Es kann daher nützlich sein, Vergleichszahlen aus der uns umgebenden realen Welt bereitzustellen, so dass man ein Gefühl für die Sicherheit kryptographischer Algorithmen entwickeln kann. Die angegebenen Werte stammen teilweise aus [Schwenk1996] und [Schneier1996, S.18].

Wahrscheinlichkeit, dass Sie auf ihrem nächsten Flug entführt werden	$5,5 \cdot 10^{-6}$
Wahrscheinlichkeit für 6 Richtige im Lotto	$7,1 \cdot 10^{-8}$
Jährliche Wahrscheinlichkeit, von einem Blitz getroffen zu werden	10^{-7}
Risiko, von einem Meteoriten erschlagen zu werden	$1,6 \cdot 10^{-12}$

Zeit bis zur nächsten Eiszeit (in Jahren)	14000	(2^{14})
Zeit bis die Sonne verglüht (in Jahren)	10^9	(2^{30})
Alter der Erde (in Jahren)	10^9	(2^{30})
Alter des Universums (in Jahren)	10^{10}	(2^{34})
Anzahl der Atome der Erde	10^{51}	(2^{170})
Anzahl der Atome der Sonne	10^{57}	(2^{190})
Anzahl der Atome im Universum (ohne dunkle Materie)	10^{77}	(2^{265})
Volumen des Universums (in cm^3)	10^{84}	(2^{280})

2.8.4 Spezielle Werte des Zweier- und Zehnersystems

Dualsystem	Zehnersystem
2^{10}	1.024
2^{40}	$1,09951 \cdot 10^{12}$
2^{56}	$7,20576 \cdot 10^{16}$
2^{64}	$1,84467 \cdot 10^{19}$
2^{80}	$1,20893 \cdot 10^{24}$
2^{90}	$1,23794 \cdot 10^{27}$
2^{112}	$5,19230 \cdot 10^{33}$
2^{128}	$3,40282 \cdot 10^{38}$
2^{150}	$1,42725 \cdot 10^{45}$
2^{160}	$1,46150 \cdot 10^{48}$
2^{250}	$1,80925 \cdot 10^{75}$
2^{256}	$1,15792 \cdot 10^{77}$
2^{320}	$2,13599 \cdot 10^{96}$
2^{512}	$1,34078 \cdot 10^{154}$
2^{768}	$1,55252 \cdot 10^{231}$
2^{1024}	$1,79769 \cdot 10^{308}$
2^{2048}	$3,23170 \cdot 10^{616}$

Berechnung zum Beispiel per GMP: `http://www.gnu.ai.mit.edu`.

Literatur

[Aaronson2003] Scott Aaronson,
The Prime Facts: From Euclid to AKS,
http://www.cs.berkeley.edu/~aaronson/prime.ps.
Dieses schöne Paper wurde mir erst nach Vollendung dieses Artikels bekannt: es bietet ebenfalls einen einfachen, didaktisch gut aufbereiteten und trotzdem nicht oberflächlichen Einstieg in dieses Thema (ist leider nur in Englisch verfügbar).

[Bartholome1996] A. Bartholomé, J. Rung, H. Kern,
Zahlentheorie für Einsteiger, Vieweg 1995, 2. Auflage 1996.

[Blum1999] W. Blum,
Die Grammatik der Logik, dtv, 1999.

[Bundschuh1998] Peter Bundschuh,
Einführung in die Zahlentheorie, Springer 1988, 4. Auflage 1998.

[Doxiadis2000] Apostolos Doxiadis,
Onkel Petros und die Goldbachsche Vermutung,
Bloomsbury 2000 (deutsch bei Lübbe 2000 und bei BLT als Taschenbuch 2001).

[Graham1989] R.E. Graham, D.E. Knuth, O. Patashnik,
Concrete Mathematics, Addison-Wesley, 1989.

[Klee1997] V. Klee, S. Wagon,
Ungelöste Probleme in der Zahlentheorie und der Geometrie der Ebene,
Birkhäuser Verlag, 1997.

[Knuth1981] Donald E. Knuth,
The Art of Computer Programming, vol 2: Seminumerical Algorithms,
Addison-Wesley, 1969, 2nd edition 1981.

[Lorenz1993] F. Lorenz,
Algebraische Zahlentheorie, BI Wissenschaftsverlag, 1993.

[Padberg1996] F. Padberg,
Elementare Zahlentheorie, Spektrum Akademischer Verlag 1988, 2. Auflage 1996.

[Pieper1983] H. Pieper,
Zahlen aus Primzahlen, Verlag Harri Deutsch 1974, 3. Auflage 1983.

[Richstein1999] J. Richstein,
Verifying the Goldbach Conjecture up to $4 * 10^{14}$, Mathematics of Computation.

[Schneier1996] Bruce Schneier,
Applied Cryptography, Protocols, Algorithms, and Source Code in C,
Wiley and Sons, 2nd edition 1996.

[Schroeder1999] M.R. Schroeder,
Number Theory in Science and Communication,
Springer 1984, 3rd edition 1997, Corrected Printing 1999.

[Schwenk1996] J. Schwenk
Conditional Access, in taschenbuch der telekom praxis 1996,
Hrgb. B. Seiler, Verlag Schiele und Schön, Berlin.

[Tietze1973] H. Tietze,
Gelöste und ungelöste mathematische Probleme, Verlag C.H. Beck 1959, 6. Auflage 1973.

Web-Links

1. GIMPS (Great Internet Mersenne-Prime Search)
   www.mersenne.org ist die Homepage des GIMPS-Projekts,
   http://www.mersenne.org/prime.htm
2. Die Proth Search Page mit dem Windows-Programm von Yves Gallot
   http://prothsearch.net/index.html
3. Verallgemeinerte Fermat-Primzahlen-Suche
   http://perso.wanadoo.fr/yves.gallot/primes/index.html
4. Verteilte Suche nach Teilern von Fermatzahlen
   http://www.fermatsearch.org/
5. An der Universität von Tennessee findet man umfangreiche Forschungsergebnisse über Primzahlen.
   http://www.utm.edu/
6. Den besten Überblick zu Primzahlen (weil sehr aktuell und vollständig) bieten m.E. „The Prime Pages“ von Chris Caldwell.
   http://www.utm.edu/research/primes
7. Beschreibungen u.a. zu Primzahltests
   http://www.utm.edu/research/primes/mersenne.shtml
   http://www.utm.edu/research/primes/prove/index.html
8. Ausgabe der n-ten Primzahl
   http://www.utm.edu/research/primes/notes/by_year.html
9. Der Supercomputerhersteller SGI Cray Research beschäftigte nicht nur hervorragende Mathematiker, sondern benutzte die Primzahltests auch als Benchmarks für seine Maschinen.
   http://reality.sgi.com/chongo/prime/prime_press.html
10. http://www.eff.org/coop-awards/prime-release1.html
11. http://www.informatik.tu-darmstadt.de/TI/LiDIA/
12. http://www.math.Princeton.EDU/~arbooker/nthprime.html
13. http://www.cerias.purdue.edu/homes/ssw/cun
14. http://www.informatik.uni-giessen.de/staff/richstein/de/Goldbach.html
15. http://www.mathematik.ch/mathematiker/goedel.html
16. http://www.mscs.dal.ca/~dilcher/goldbach/index.html

Dank

Für das sehr konstruktive Korrekturlesen dieses Artikels: Hr. Henrik Koy und Hr. Roger Oyono.

3 Einführung in die elementare Zahlentheorie mit Beispielen

(Bernhard Esslinger, besslinger@web.de, Juli 2001, Updates: Nov. 2001, Juni 2002, Feb. 2003)

Diese „Einführung“ bietet einen Einstieg für mathematisch Interessierte. Erforderlich sind nicht mehr Vorkenntnisse als die, die im Grundkurs Mathematik am Gymnasium vermittelt werden.

Wir haben uns bewusst an „Einsteigern“ und „Interessenten“ orientiert, und nicht an den Gepflogenheiten mathematischer Lehrbücher, die auch dann „Einführung“ genannt werden, wenn sie schon auf der 5. Seite nicht mehr auf Anhieb zu verstehen sind und sie eigentlich den Zweck haben, dass man danach auch spezielle Monographien zu dem Thema lesen können soll.

3.1 Mathematik und Kryptographie

Ein großer Teil der modernen, asymmetrischen Kryptographie beruht auf mathematischen Erkenntnissen – auf den Eigenschaften („Gesetzen”) ganzer Zahlen, die in der elementaren Zahlentheorie untersucht werden. „Elementar“ bedeutet hier, dass die zahlentheoretischen Fragestellungen im wesentlichen in der Menge der natürlichen und der ganzen Zahlen durchgeführt werden.

Weitere mathematische Disziplinen, die heute in der Kryptographie Verwendung finden, sind (vgl. [Bauer1995, S. 2], [Bauer2000, Seite 3]) :

- Gruppentheorie
- Kombinatorik
- Komplexitätstheorie
- Ergodentheorie
- Informationstheorie.

Die Zahlentheorie oder Arithmetik (hier wird mehr der Aspekt des Rechnens mit Zahlen betont) wurde von Carl Friedrich Gauss als besondere mathematische Disziplin begründet. Zu ihren elementaren Gegenständen gehören: größter gemeinsamer Teiler[21] (ggT), Kongruenzen (Restklassen), Faktorisierung, Satz von Euler-Fermat und primitive Wurzeln. Kernbegriff sind jedoch die Primzahlen und ihre multiplikative Verknüpfung.

Lange Zeit galt gerade die Zahlentheorie als Forschung pur, als Paradebeispiel für die Forschung im Elfenbeinturm. Sie erforschte die „geheimnisvollen Gesetze im Reich der Zahlen” und gab Anlass zu philosophischen Erörterungen, ob sie beschreibt, was überall in der Natur schon da ist, oder ob sie ihre Elemente (Zahlen, Operatoren, Eigenschaften) nicht künstlich konstruiert.

Inzwischen weiß man, dass sich zahlentheoretische Muster überall in der Natur finden. Zum Beispiel verhalten sich die Anzahl der links- und der rechtsdrehenden Spiralen einer Sonnenblume

[21] Auf ggT, englisch gcd (greatest common divisor), geht dieser Artikel im Anhang A zu diesem Kapitel ein.

zueinander wie zwei aufeinanderfolgende Fibonacci-Zahlen[22], also z.B. wie 21 : 34.

Außerdem wurde spätestens mit den zahlentheoretischen Anwendungen der modernen Kryptographie klar, dass eine jahrhundertelang als theoretisch geltende Disziplin praktische Anwendung findet, nach deren Experten heute eine hohe Nachfrage auf dem Arbeitsmarkt besteht.

Anwendungen der (Computer-)Sicherheit bedienen sich heute der Kryptographie, weil Kryptographie als mathematische Disziplin einfach besser und beweisbarer ist als alle im Laufe der Jahrhunderte erfundenen „kreativen" Verfahren der Substitution und besser als alle ausgefeilten physischen Techniken wie beispielsweise beim Banknotendruck [Beutelspacher1996, S. 4].

In diesem Artikel werden in einer leicht verständlichen Art die grundlegenden Erkenntnisse der elementaren Zahlentheorie anhand vieler Beispiele vorgestellt – auf Beweise wird (fast) vollständig verzichtet (diese finden sich in den mathematischen Lehrbüchern).

Ziel ist nicht die umfassende Darstellung der zahlentheoretischen Erkenntnisse, sondern das Aufzeigen der wesentlichen Vorgehensweisen. Der Umfang des Stoffes orientiert sich daran, das RSA-Verfahren verstehen und anwenden zu können.

Dazu wird sowohl an Beispielen als auch in der Theorie erklärt, wie man in endlichen Mengen rechnet und wie dies in der Kryptographie Anwendung findet. Insbesondere wird auf die klassischen Public Key-Verfahren Diffie-Hellman (DH) und RSA eingegangen.

Außerdem war es mir wichtig, fundierte Aussagen zur Sicherheit des RSA-Verfahrens zu machen.

> *Carl Friedrich Gauss (30.4.1777−23.2.1855):*
> Die Mathematik ist die Königin der Wissenschaften, die Zahlentheorie aber ist die Königin der Mathematik.

3.2 Einführung in die Zahlentheorie

Die Zahlentheorie entstand aus Interesse an den positiven ganzen Zahlen $1, 2, 3, 4, \cdots$, die auch als die Menge der *natürlichen Zahlen* $\mathbb{N}$ bezeichnet werden. Sie sind die ersten mathematischen Konstrukte der menschlichen Zivilisation. Nach Kronecker[23] hat sie der liebe Gott geschaffen, nach Dedekind[24] der menschliche Geist. Das ist je nach Weltanschauung ein unlösbarer Widerspruch oder ein und dasselbe.

Im Altertum gab es keinen Unterschied zwischen Zahlentheorie und Numerologie, die einzelnen Zahlen mystische Bedeutung zumaß. So wie sich während der Renaissance (ab dem 14. Jahrhundert) die Astronomie allmählich von der Astrologie und die Chemie von der Alchemie löste, so

[22] Die Folge der Fibonacci-Zahlen $(a_i)_{i\in\mathbb{N}}$ ist definiert durch die „rekursive" Vorschrift $a_1 := a_2 := 1$ und für alle Zahlen $n = 1, 2, 3, \cdots$ definiert man $a_{n+2} := a_{n+1} + a_n$. Zu dieser historischen Folge gibt es viele interessante Anwendungen in der Natur (siehe z.B. [Graham1994, S. 290 ff] oder die Web-Seite von Ron Knott: hier dreht sich alles um Fibonacci-Zahlen). Die Fibonacci-Folge ist gut verstanden und wird heute als wichtiges Werkzeug in der Mathematik benutzt.

[23] Leopold Kronecker, deutscher Mathematiker, 1823−1891.

[24] Julius Wilhelm Richard Dedekind, deutscher Mathematiker, 06.10.1831−12.02.1916.

ließ auch die Zahlentheorie die Numerologie hinter sich.

Die Zahlentheorie faszinierte schon immer Amateure wie auch professionelle Mathematiker. Im Unterschied zu anderen Teilgebieten der Mathematik können viele der Probleme und Sätze auch von Laien verstanden werden, andererseits widersetzten sich die Lösungen zu den Problemen und die Beweise zu den Sätzen oft sehr lange den Mathematikern. Es ist also leicht, gute Fragen zu stellen, aber es ist ganz etwas anderes, die Antwort zu finden. Ein Beispiel dafür ist der sogenannte letzte (oder große) Satz von Fermat[25].

Bis zur Mitte des 20. Jahrhunderts wurde die Zahlentheorie als das reinste Teilgebiet der Mathematik angesehen – ohne Verwendung in der wirklichen Welt. Mit dem Aufkommen der Computer und der digitalen Kommunikation änderte sich das: die Zahlentheorie konnte einige unerwartete Antworten für reale Aufgabenstellungen liefern. Gleichzeitig halfen die Fortschritte in der EDV, dass die Zahlentheoretiker große Fortschritte machten im Faktorisieren großer Zahlen, in der Bestimmung neuer Primzahlen, im Testen von (alten) Vermutungen und beim Lösen bisher unlösbarer numerischer Probleme.

Die moderne Zahlentheorie besteht aus Teilgebieten wie

- Elementare Zahlentheorie
- Algebraische Zahlentheorie
- Analytische Zahlentheorie
- Geometrische Zahlentheorie
- Kombinatorische Zahlentheorie
- Numerische Zahlentheorie und
- Wahrscheinlichkeitstheorie.

Die verschiedenen Teilgebiete beschäftigen sich alle mit Fragestellungen zu den ganzen Zahlen (positive und negative ganze Zahlen und die Null), gehen diese jedoch mit verschiedenen Methoden an.

Dieser Artikel beschäftigt sich nur mit dem Teilgebiet der elementaren Zahlentheorie.

[25] Thema der Schul-Mathematik ist der Satz von Pythagoras, wo gelehrt wird, dass in einem rechtwinkligen Dreieck gilt: $a^2 + b^2 = c^2$, wobei a, b die Schenkellängen sind und c die Länge der Hypothenuse ist. Fermats berühmte Behauptung war, dass für ganzzahlige Exponenten $n > 2$ immer die Ungleichheit $a^n + b^n \neq c^n$ gilt. Leider fand Fermat auf dem Brief, wo er die Behauptung aufstellte, nicht genügend Platz, um den Satz zu beweisen. Der Satz konnte erst über 300 Jahre später bewiesen werden [Wiles1994, S. 433-551].

3.2.1 Konvention

Wird nichts anderes gesagt, gilt:

- Die Buchstaben $a, b, c, d, e, k, n, m, p, q$ stehen für ganze Zahlen.
- Die Buchstaben i und j stehen für natürliche Zahlen.
- Der Buchstabe p steht stets für eine Primzahl.
- Die Mengen $\mathbb{N} = \{1, 2, 3, \cdots\}$ und $\mathbb{Z} = \{\cdots, -3, -2, -1, 0, 1, 2, 3, \cdots\}$ sind die *natürlichen* und die *ganzen* Zahlen.

Joanne K. Rowling[26]*:*
Das ist nicht Zauberei, das ist Logik, ein Rätsel. Viele von den größten Zauberern haben keine Unze Logik im Kopf.

3.3 Primzahlen und der erste Hauptsatz der elementaren Zahlentheorie

Viele der Probleme in der elementaren Zahlentheorie beschäftigen sich mit Primzahlen.

Jede ganze Zahl hat Teiler oder Faktoren. Die Zahl 1 hat nur einen, nämlich sich selbst. Die Zahl 12 hat die sechs Teiler 1, 2, 3, 4, 6 und 12[27]. Viele Zahlen sind nur teilbar durch sich selbst und durch 1. Bezüglich der Multiplikation sind dies die „Atome" im Bereich der Zahlen.

Definition 3.1. Primzahlen *sind natürliche Zahlen größer als* 1*, die nur durch* 1 *und sich selbst teilbar sind.*

Per Definition ist 1 keine Primzahl.

Schreibt man die Primzahlen in aufsteigender Folge (Primzahlenfolge), so ergibt sich

$$2,\ 3,\ 5,\ 7,\ 11,\ 13,\ 17,\ 19,\ 23,\ 29,\ 31,\ 37,\ 41,\ 43,\ 47,\ 53,\ 59,\ 61,\ 67,\ 71,\ 73,\ 79,\ 83,\ 89,\ 97, \cdots .$$

Unter den ersten 100 Zahlen gibt es genau 25 Primzahlen. Danach nimmt ihr prozentualer Anteil ab, wird aber nie Null.

Primzahlen treten als ganze Zahlen nicht selten auf. Allein im letzten Jahrzehnt waren drei Jahre prim: 1993, 1997 und 1999. Wären sie selten, könnte die Kryptographie auch nicht so mit ihnen arbeiten, wie sie es tut.

Primzahlen können nur auf eine einzige („triviale") Weise zerlegt werden:

$$\begin{aligned} 5 &= 1 * 5 \\ 17 &= 1 * 17 \\ 1.013 &= 1 * 1.013 \\ 1.296.409 &= 1 * 1.296.409. \end{aligned}$$

Definition 3.2. *Natürliche Zahlen größer* 1*, die keine Primzahlen sind, heißen* **zusammengesetzte Zahlen***: diese haben mindestens zwei von* 1 *verschiedene Faktoren.*

[26] Joanne K. Rowling, „Harry Potter und der Stein der Weisen", Carlsen, (c) 1997, Kapitel „Durch die Falltür", S. 310, Hermine.

[27] Aufgrund der großen Teilerzahl von 12 findet sich diese Zahl – und Vielfache dieser Zahl – oft im alltäglichen wieder: Die 12 Stunden-Skala der Uhr, die 60 Minuten einer Stunde, die 360 Grad-Skala der Winkelmessung, usw. Teilt man diese Skalen in Bruchteile auf, so ergeben sich in vielen Fällen die Brüche als ganze Zahlen. Mit diesen kann man im Kopf einfacher rechnen als mit gebrochenen Zahlen.

Beispiele für die Primfaktorzerlegung solcher Zahlen:

$$\begin{aligned} 4 &= 2 * 2 \\ 6 &= 2 * 3 \\ 91 &= 7 * 13 \\ 161 &= 7 * 23 \\ 767 &= 13 * 59 \\ 1.029 &= 3 * 7^3 \\ 5.324 &= 22 * 11^3. \end{aligned}$$

Satz 3.1. *Jede zusammengesetzte Zahl a besitzt einen kleinsten Teiler größer als* 1*. Dieser Teiler ist eine Primzahl p und kleiner oder gleich der Quadratwurzel aus a.*

Aus den Primzahlen lassen sich alle ganzen Zahlen größer als 1 zusammensetzen – und das sogar in einer *eindeutigen* Weise.

Dies besagt der 1. *Hauptsatz der Zahlentheorie* (= Hauptsatz der elementaren Zahlentheorie = fundamental theorem of arithmetic = fundamental building block of all positive integers). Er wurde das erste Mal präzise von Carl Friedrich Gauss in seinen Disquisitiones Arithmeticae (1801) formuliert.

Satz 3.2. Gauss 1801 *Jede natürliche Zahl a größer als* 1 *lässt sich als Produkt von Primzahlen schreiben. Sind zwei solche Zerlegungen $a = p_1 * p_2 * \cdots * p_n = q_1 * q_2 * \cdots * q_m$ gegeben, dann gilt nach eventuellem Umsortieren $n = m$ und für alle i: $p_i = q_i$.*

In anderen Worten: Jede natürliche Zahl außer der 1 lässt sich auf genau eine Weise als Produkt von Primzahlen schreiben, wenn man von der Reihenfolge der Faktoren absieht. Die Faktoren sind also eindeutig (die „Expansion in Faktoren" ist eindeutig)!

Zum Beispiel ist $60 = 2 * 2 * 3 * 5 = 2^2 * 3 * 5$. Und das ist — bis auf eine veränderte Reihenfolge der Faktoren — die einzige Möglichkeit, die Zahl 60 in Primfaktoren zu zerlegen.

Wenn man nicht nur Primzahlen als Faktoren zulässt, gibt es mehrere Möglichkeiten der Zerlegung in Faktoren und die *Eindeutigkeit* (uniqueness) geht verloren:

$$60 = 1 * 60 = 2 * 30 = 4 * 15 = 5 * 12 = 6 * 10 = 2 * 3 * 10 = 2 * 5 * 6 = 3 * 4 * 5 = \cdots$$

Der 1. Hauptsatz ist nur scheinbar selbstverständlich. Man kann viele andere Zahlenmengen[28] konstruieren, bei denen eine multiplikative Zerlegung in die Primfaktoren dieser Mengen *nicht* eindeutig ist.

Für eine mathematische Aussage ist es deshalb nicht nur wichtig, für welche Operation sie definiert wird, sondern auch auf welcher Grundmenge diese Operation definiert wird.

Weitere Details zu den Primzahlen (z.B. wie der „Kleine Satz von Fermat" zum Testen sehr großer Zahlen auf ihre Primzahleigenschaft benutzt werden kann) finden sich in diesem Skript in dem Artikel über Primzahlen.

[28] Diese Mengen werden speziell aus der Menge der natürlichen Zahlen gebildet. Ein Beispiel findet sich in diesem Skript auf Seite 18 am Ende von Kapitel 2.2

3.4 Teilbarkeit, Modulus und Restklassen

Werden ganze Zahlen addiert, subtrahiert oder multipliziert, ist das Ergebnis stets wieder eine ganze Zahl.

Die Division zweier ganzer Zahlen ergibt nicht immer eine ganze Zahl. Wenn man z.B. 158 durch 10 teilt, ist das Ergebnis die Dezimalzahl $15,8$. Dies ist keine ganze Zahl!

Teilt man 158 dagegen durch 2, ist das Ergebnis 79 eine ganze Zahl. In der Zahlentheorie sagt man, 158 ist *teilbar* durch 2, aber nicht durch 10. Allgemein sagt man:

Definition 3.3. *Eine ganze Zahl n ist* **teilbar** *durch eine ganze Zahl d, wenn der Quotient n/d eine ganze Zahl c ist, so dass $n = c * d$.*

Die Zahl n wird *Vielfaches* von d genannt; d wird *Teiler, Divisor* oder *Faktor* von n genannt.

Mathematisch schreibt man das: $d|n$ (gelesen: „d teilt n"). Die Schreibweise $d \not| n$ bedeutet, dass d die Zahl n nicht teilt.

Also gilt in unserem obigen Beispiel: $10 \not| 158$, aber $2|158$.

3.4.1 Die Modulo-Operation – Rechnen mit Kongruenzen

Bei Teilbarkeitsuntersuchungen kommt es nur auf die Reste der Division an: Teilt man eine Zahl n durch m, so benutzt man oft die folgende Schreibweise:

$$\frac{n}{m} = c + \frac{r}{m},$$

wobei c eine ganze Zahl ist und r eine Zahl mit den Werten $0, 1, \cdots, m-1$. Diese Schreibweise heißt Division mit Rest. Dabei heißt c der ganzzahlige „Quotient" und r der „Rest" der Division.

Beispiel:

$$\frac{19}{7} = 2 + \frac{5}{7} \quad (m = 7, c = 2, r = 5)$$

Was haben die Zahlen $5, 12, 19, 26, \cdots$ bei der Division durch 7 gemeinsam? Es ergibt sich immer der Rest $r = 5$. Bei der Division durch 7 sind nur die folgenden Reste möglich:

$$r = 0, 1, 2, \cdots, 6.$$

Wir fassen bei der Division durch 7 die Zahlen, die den gleichen Rest r ergeben, in die „Restklasse r modulo 7" zusammen. Zwei Zahlen a und b, die zur gleichen Restklasse modulo 7 gehören, bezeichnen wir als „kongruent modulo 7". Oder ganz allgemein:

Definition 3.4. *Als* **Restklasse** *r* **modulo** *m bezeichnet man alle ganzen Zahlen a, die bei der Division durch m denselben Rest r haben.*

Beispiele:

Restklasse 0 modulo $4 = \{x | x = 4 * n;\ n \in \mathbb{N}\} = \{\dots, -16, -12, -8, -4, 0, 4, 8, 12, 16, \dots\}$

Restklasse 3 modulo $4 = \{x | x = 4 * n + 3;\ n \in \mathbb{N}\} = \{\dots, -13, -9, -5, -1, 3, 7, 11, 15, \dots\}$

Da modulo m nur die Reste $0, 1, 2, \cdots, m-1$ möglich sind, rechnet die modulare Arithmetik in endlichen Mengen. Zu jedem Modul m gibt es genau m Restklassen.

Definition 3.5. *Zwei Zahlen $a, b \in \mathbb{N}$ heißen* **restgleich oder kongruent bezüglich** $m \in \mathbb{N}$ *genau dann, wenn beim Teilen durch m der gleiche Rest bleibt.*

Man schreibt: $a \equiv b \pmod{m}$. Und sagt: *a ist kongruent b modulo m*. Das bedeutet, dass a und b zur gleichen Restklasse gehören. Der Modul ist also der Teiler. Diese Schreibweise wurde von Gauss eingeführt. Gewöhnlich ist der Teiler positiv, aber a und b können auch beliebige ganze Zahlen sein.

Beispiele:

$19 \equiv 12 \pmod{7}$, denn die Reste sind gleich: $19/7 = 2$ Rest 5 und $12/7 = 1$ Rest 5.

$23103 \equiv 0 \pmod{453}$, denn $23103/453 = 51$ Rest 0 und $0/453 = 0$ Rest 0.

Satz 3.3. *$a \equiv b \pmod{m}$ gilt genau dann, wenn die Differenz $(a - b)$ durch m teilbar ist, also wenn ein $q \in \mathbf{Z}$ existiert mit $(a - b) = q * m$.*

Diese beiden Aussagen sind also äquivalent.

Daraus ergibt sich: Wenn m die Differenz teilt, gibt es eine ganze Zahl q, so dass gilt: $a = b + q * m$. Alternativ zur Kongruenzschreibweise kann man auch die Teilbarkeitsschreibweise verwenden: $m|(a - b)$.

Beispiel äquivalenter Aussagen:
$35 \equiv 11 \pmod{3} \iff 35 - 11 \equiv 0 \pmod{3}$, wobei $35 - 11 = 24$ sich ohne Rest durch 3 teilen lässt, während $35 : 3$ und $11 : 3$ beide den Rest 2 ergeben.

Bemerkung:
Für die Summe $(a + b)$ gilt die obige Äquivalenz nicht!

Beispiel:
$11 \equiv 2 \pmod{3}$, also ist $11 - 2 \equiv 9 \equiv 0 \pmod{3}$; aber $11 + 2 = 13$ ist nicht durch 3 teilbar. Die Aussage von Satz 3.3 gilt für Summen nicht einmal in eine Richtung. Richtig ist sie bei Summen nur für den Rest 0 und nur in der folgenden Richtung: Teilt ein Teiler beide Summanden ohne Rest, teilt er auch die Summe ohne Rest.

Anwenden kann man die obige Äquivalenz von Satz 3.3, wenn man schnell und geschickt für große Zahlen entscheiden will, ob sie durch eine bestimmte Zahl teilbar sind.

Beispiel:
Ist 69.993 durch 7 teilbar?
Da die Zahl in eine Differenz zerlegt werden kann, wo einfach zu ersehen ist, dass jeder Operand durch 7 teilbar ist, ist auch die Differenz durch 7 teilbar: $69.993 = 70.000 - 7$.

Diese Überlegungen und Definitionen mögen recht theoretisch erscheinen, sind uns im Alltag aber so vertraut, dass wir die formale Vorgehensweise gar nicht mehr wahrnehmen: Bei der Uhr werden die 24 h eines Tages durch die Zahlen $1, 2, \cdots, 12$ repräsentiert. Die Stunden nach $12:00$ mittags erhält man als Reste einer Division durch 12. Wir wissen sofort, dass 2 Uhr nachmittags dasselbe wie $14:00$ ist.

Diese „modulare", also auf die Divisionsreste bezogene Arithmetik ist die Basis der asymmetrischen Verschlüsselungsverfahren. Kryptographische Berechnungen spielen sich also nicht wie das Schulrechnen unter den reellen Zahlen ab, sondern unter Zeichenketten begrenzter Länge, das heißt unter positiven ganzen Zahlen, die einen gewissen Wert nicht überschreiten dürfen. Aus diesem und anderen Gründen wählt man sich eine große Zahl m und „rechnet modulo m", das heißt, man ignoriert ganzzahlige Vielfache von m und rechnet statt mit einer Zahl nur mit dem Rest bei Division dieser Zahl durch m. Dadurch bleiben alle Ergebnisse im Bereich von 0 bis $m - 1$.

3.5 Rechnen in endlichen Mengen

3.5.1 Gesetze beim modularen Rechnen

Aus Sätzen der Algebra folgt, dass wesentliche Teile der üblichen Rechenregeln beim Übergang zum modularen Rechnen über der Grundmenge $\mathbb{Z}$ erhalten bleiben: Die Addition ist nach wie vor kommutativ. Gleiches gilt für die Multiplikation modulo m. Das Ergebnis einer Division[29] ist kein Bruch, sondern eine ganze Zahl zwischen 0 und $m - 1$.

Es gelten die bekannten Gesetze:

1. **Assoziativgesetz:**
   $((a + b) + c) \pmod m \equiv (a + (b + c)) \pmod m$.
   $((a * b) * c) \pmod m \equiv (a * (b * c)) \pmod m$.

2. **Kommutativgesetz:**
   $(a + b) \pmod m \equiv (b + a) \pmod m$.
   $(a * b) \pmod m \equiv (b * a) \pmod m$.

Assoziativgesetz und Kommutativgesetz gelten sowohl für die Addition als auch für die Multiplikation.

3. **Distributivgesetz:**
   $(a * (b + c)) \pmod m \equiv (a * b + a * c) \pmod m$.

[29] Die Division modulo m ist nur für Zahlen, die teilerfremd zu m sind, definiert. Vergleiche Fußnote 33 in Kapitel 3.6.1.

4. **Reduzierbarkeit:**
   $(a + b) \pmod m \equiv (a \pmod m + b \pmod m) \pmod m$.
   $(a * b) \pmod m \equiv (a \pmod m * b \pmod m) \pmod m$.

Es ist gleichgültig, in welcher Reihenfolge die Modulo-Operation durchgeführt wird.

5. **Existenz einer Identität (neutrales Element):**
   $(a + 0) \pmod m \equiv (0 + a) \pmod m \equiv a \pmod m$.
   $(a * 1) \pmod m \equiv (1 * a) \pmod m \equiv a \pmod m$.

6. **Existenz des inversen Elements:**
   Für jedes ganzzahlige a und m gibt es eine ganze Zahl $-a$, so dass gilt:
   $(a + (-a)) \pmod m \equiv 0 \pmod m$ (additive Inverse).
   Für jedes a ($a \not\equiv 0 \pmod p$) und p prim gibt es eine ganze Zahl a^{-1}, so dass gilt:
   $(a * a^{-1}) \pmod p \equiv 1 \pmod p$ (multiplikative Inverse).

7. **Abgeschlossenheit:**[30]
   $a, b \in G \Longrightarrow (a + b) \in G$.
   $a, b \in G \Longrightarrow (a * b) \in G$.

8. **Transitivität:**
   $[a \equiv b \bmod m,\ b \equiv c \bmod m] \Longrightarrow [a \equiv c \bmod m]$.

3.5.2 Muster und Strukturen

Generell untersuchen die Mathematiker „Strukturen“. Sie fragen sich z.B. bei $a * x \equiv b \bmod m$, welche Werte x für gegebene Werte a, b, m annehmen kann.

Insbesondere wird dies untersucht für den Fall, dass das Ergebnis b der Operation das neutrale Element ist. Dann ist x die Inverse von a bezüglich dieser Operation.

[30] Diese Eigenschaft wird innerhalb einer Menge immer bezüglich einer Operation definiert. Siehe Anhang B zu diesem Kapitel.

Seneca[31]*:*
Lang ist der Weg durch Lehren, kurz und wirksam durch Beispiele.

3.6 Beispiele für modulares Rechnen

Wir haben bisher gesehen:

Für zwei natürliche Zahlen a und m bezeichnet a mod m den Rest, den man erhält, wenn man a durch m teilt. Daher ist $a \pmod m$ stets eine Zahl zwischen 0 und $m-1$.

Zum Beispiel gilt: $1 \equiv 6 \equiv 41 \pmod 5$, denn der Rest ist jeweils 1.

Ein anderes Beispiel ist: $2000 \equiv 0 \pmod 4$, denn 4 geht in 2000 ohne Rest auf.

In der modularen Arithmetik gibt es nur eine eingeschränkte Menge nicht-negativer Zahlen. Deren Anzahl wird durch einen Modul m vorgegeben. Ist der Modul $m = 5$, werden nur die 5 Zahlen der Menge $\{0, 1, 2, 3, 4\}$ benutzt.

Ein Rechenergebnis größer als 4 wird dann „modulo" 5 umgeformt, d.h. es ist der Rest, der sich bei der Division des Ergebnisses durch 5 ergibt. So ist etwa $2 * 4 \equiv 8 \equiv 3 \pmod 5$, da 3 der Rest ist, wenn man 8 durch 5 teilt.

3.6.1 Addition und Multiplikation

Im folgenden werden

- die Additionstabelle[32] für mod 5 und
- die Multiplikationstabellen[33] für mod 5 und für mod 6

aufgestellt.

Beispiel Additionstabelle:

Das Ergebnis der Addition von 3 und 4 (mod 5) wird folgendermaßen bestimmt: berechne $3+4 = 7$ und ziehe solange die 5 vom Ergebnis ab, bis sich ein Ergebnis kleiner als der Modul ergibt: $7 - 5 = 2$. Also ist: $3 + 4 \equiv 2 \pmod 5$.

---

[31] Lucius Annaeus Seneca, philosophischer Schriftsteller und Dichter, 4 v. Chr. - 65 n. Chr.

[32] Bemerkung zur Subtraktion modulo 5:
$2 - 4 \equiv -2 \equiv 3 \bmod 5$.
Es gilt modulo 5 also nicht, dass $-2 = 2$ (siehe auch Anhang C zu diesem Kapitel).

[33] Bemerkung zur Division modulo 6:
Bei der Division darf nicht durch die Null geteilt werden, dies liegt an der besonderen Rolle der 0 als Identität bei der Addition:
für alle a gilt $a * 0 = 0$, denn $a * 0 = a * (0 + 0) = a * 0 + a * 0$. Es ist offensischtlich, dass 0 keine Inverse bzgl. der Multiplikation besitzt, denn sonst müsste gelten $0 = 0 * 0^{-1} = 1$. Vergleiche Fußnote 29.

Additionstabelle modulo 5:	+	0	1	2	3	4
	0	0	1	2	3	4
	1	1	2	3	4	0
	2	2	3	4	0	1
	3	3	4	0	1	2
	4	4	0	1	2	3

Beispiel Multiplikationstabelle:

Das Ergebnis der Multiplikation $4*4$ (mod 5) wird folgendermaßen berechnet: berechne $4*4 = 16$ und ziehe solange die 5 ab, bis sich ein Ergebnis kleiner als der Modul ergibt:

$$16 - 5 = 11;\ 11 - 5 = 6;\ 6 - 5 = 1.$$

Direkt ergibt es sich auch aus der Tabelle: $4 * 4 \equiv 1$ (mod 5), weil $16 : 5 = 3$ Rest 1. Die Multiplikation wird auf der Menge $\mathbb{Z}$ ohne 0 definiert.

Multiplikationstabelle modulo 5:	*	1	2	3	4
	1	1	2	3	4
	2	2	**4**	**1**	3
	3	3	**1**	**4**	2
	4	4	3	2	1

3.6.2 Additive und multiplikative Inversen

Aus den Tabellen kann man zu jeder Zahl die Inversen bezüglich der Addition und der Multiplikation ablesen.

Die Inverse einer Zahl ist diejenige Zahl, die bei Addition der beiden Zahlen das Ergebnis 0 und bei der Multiplikation das Ergebnis 1 ergibt. So ist die Inverse von 4 für die Addition mod 5 die 1 und für die Multiplikation mod 5 die 4 selbst, denn

$$4 + 1 = \ \ 5 \equiv 0 \ (\text{mod } 5);$$
$$4 * 4 = 16 \equiv 1 \ (\text{mod } 5).$$

Die Inverse von 1 bei der Multiplikation mod 5 ist 1; die Inverse modulo 5 von 2 ist 3 und weil die Multiplikation kommutativ ist, ist die Inverse von 3 wiederum die 2.

Wenn man zu einer beliebigen Zahl (hier 2) eine weitere beliebige Zahl (hier 4) addiert bzw. multipliziert und danach zum Zwischenergebnis (1 bzw. 3) die jeweilige Inverse der weiteren Zahl (1 bzw. 4) addiert[34] bzw. multipliziert, ist das Gesamtergebnis gleich dem Ausgangswert.

Beispiele:

$$2 + 4 \equiv 6 \equiv 1 \ (\text{mod } 5); \quad 1 + 1 \equiv 2 \equiv 2 \ (\text{mod } 5)$$
$$2 * 4 \equiv 8 \equiv 3 \ (\text{mod } 5); \quad 3 * 4 \equiv 12 \equiv 2 \ (\text{mod } 5)$$

In der Menge $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$ für die Addition und in der Menge $\mathbb{Z}_5 \setminus \{0\}$ für die Multiplikation haben alle Zahlen eine **eindeutige** Inverse bezüglich modulo 5.

Bei der modularen Addition ist das für jeden Modul (also nicht nur für 5) so.

Bei der modularen Multiplikation dagegen ist das nicht so:

Satz 3.4. *Für eine natürliche Zahl a aus der Menge $\{1, \cdots, m-1\}$ gibt es genau dann eine multiplikative Inverse, wenn sie mit dem Modul m teilerfremd*[35] *ist, d.h. wenn a und m keine gemeinsamen Primfaktoren haben.*

Da $m = 5$ eine Primzahl ist, sind die Zahlen 1 bis 4 teilerfremd zu 5, und es gibt mod 5 zu **jeder** dieser Zahlen eine multiplikative Inverse.

Ein Gegenbeispiel zeigt die Multiplikationstabelle für mod 6 (da der Modul 6 nicht prim ist, sind nicht alle Elemente aus $\mathbb{Z}_6 \setminus \{0\}$ zu 6 teilerfremd):

---

[34] Allgemein: $x + y + (-y) \equiv x \ (\text{mod } m)$ $[(-y) =$ additive Inverse zu $y \ (\text{mod } m)]$

[35] Es gilt: Zwei ganze Zahlen a und b sind genau dann teilerfremd, wenn $\text{ggT}(a, b) = 1$.
Desweiteren gilt: Ist p prim und a eine beliebige ganze Zahl, die kein Vielfaches von p ist, so sind beide Zahlen teilerfremd.
Weitere Bezeichnungen zum Thema Teilerfremdheit (mit $a_i \in \mathbb{Z}, i = 1, \cdots, n$):

1. $a_1, a_2, \cdots, a_n$ heißen *relativ prim* , wenn $\text{ggT}(a_1, \cdots, a_n) = 1$.
2. Für mehr als 2 Zahlen ist eine noch stärkere Anforderung:
   $a_1, \cdots, a_n$ heißen *paarweise relativ prim*, wenn für alle $i = 1, \cdots, n$ und $j = 1, \cdots, n$ mit $i \neq j$ gilt: $\text{ggT}(a_i, a_j) = 1$.

Beispiel: $2, 3, 6$ sind relativ prim, da $\text{ggT}(2, 3, 6) = 1$. Sie sind nicht paarweise prim, da $\text{ggT}(2, 6) = 2 > 1$.

Multiplikationstabelle modulo 6:	*	1	2	3	4	5
	1	1	2	3	4	5
	2	2	**4**	**0**	**2**	4
	3	3	**0**	**3**	**0**	3
	4	4	**2**	**0**	**4**	2
	5	5	4	3	2	1

Neben der 0 haben hier auch die Zahlen $2, 3$ und 4 keine eindeutige Inverse (man sagt auch, sie haben **keine** Inverse, weil es die elementare Eigenschaft einer Inversen ist, eindeutig zu sein).

Die Zahlen $2, 3$ und 4 haben mit dem Modul 6 den Faktor 2 oder 3 gemeinsam. Nur die zu 6 teilerfremden Zahlen 1 und 5 haben multiplikative Inverse, nämlich jeweils sich selbst.

Die Anzahl der zum Modul m teilerfremden Zahlen ist auch die Anzahl derjenigen Zahlen, die eine multiplikative Inverse haben (vgl. unten die Euler-Funktion $J(m)$).

Für die beiden in den Multiplikationstabellen verwendeten Moduli 5 und 6 bedeutet dies: Der Modul 5 ist bereits eine Primzahl. Also gibt es in mod 5 genau $J(5) = 5 - 1 = 4$ mit dem Modul teilerfremde Zahlen, also alle von 1 bis 4.

Da 6 keine Primzahl ist, zerlegen wir 6 in seine Faktoren: $6 = 2 * 3$. Daher gibt es in mod 6 genau $J(6) = (2 - 1) * (3 - 1) = 1 * 2 = 2$ Zahlen, die eine multiplikative Inverse haben, nämlich die 1 und die 5.

Für große Moduli scheint es nicht einfach, die Tabelle der multiplikativen Inversen zu berechnen (das gilt nur für die in den oberen Multiplikationstabellen fett markierten Zahlen). Mit Hilfe des kleinen Satzes von Fermat kann man dafür einen einfachen Algorithmus aufstellen [Pfleeger1997, S. 80]. Schnellere Algorithmen werden z.B. in [Knuth1998] beschrieben[36].

Kryptographisch ist nicht nur die Eindeutigkeit der Inversen, sondern auch das Ausschöpfen des gesamten Wertebereiches eine wichtige Eigenschaft.

Satz 3.5. *Sei* $a, i \in \{1, \cdots, m-1\}$ *mit* $\text{ggT}(a, m) = 1$, *dann nimmt für eine bestimmte Zahl* a *das Produkt* $a * i$ mod m *alle Werte aus* $\{1, \cdots, m-1\}$ *an (erschöpfende Permutation der Länge* $m - 1$*)*[37].

Die folgenden drei Beispiele[38] veranschaulichen Eigenschaften der multiplikativen Inversen (hier sind nicht mehr die vollständigen Multiplikationstabellen angegeben, sondern nur die Zeilen für die Faktoren 5 und 6).

[36] Mit dem erweiterten Satz von Euklid (erweiterter ggT) kann man die multiplikative Inverse berechnen und die Invertierbarkeit bestimmen (Siehe Anhang A zu diesem Kapitel). Alternativ kann auch die Primitivwurzel genutzt werden.

[37] Vergleiche auch Satz 3.14 in Kapitel 3.9 Multiplikative Ordnung und Primitivwurzel.

[38] In Anhang D zu diesem Kapitel finden Sie den Quellcode zur Berechnung der Tabellen mit Mathematica und Pari-GP.

In der Multiplikationstabelle mod 17 wurde für $i = 1, 2, \cdots, 18$ berechnet:

$(5 * i)/17 = a$ Rest r und hervorgehoben $5 * i \equiv 1 \pmod{17}$,

$(6 * i)/17 = a$ Rest r und hervorgehoben $6 * i \equiv 1 \pmod{17}$.

Gesucht ist das i, für das der Produktrest $a * i$ modulo 17 mit $a = 5$ bzw. $a = 6$ den Wert 1 hat.

Tabelle 1: Multiplikationstabelle modulo 17 **(für** $a = 5$ **und** $a = 6$**)**

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$5 * i$	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
Rest	5	10	15	3	8	13	**1**	6	11	16	4	9	14	2	7	12	0	5
$6 * i$	6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108
Rest	6	12	**1**	7	13	2	8	14	3	9	15	4	10	16	5	11	0	6

Da sowohl 5 als auch 6 jeweils teilerfremd zum Modul $m = 17$ sind, kommen zwischen $i = 1, \cdots, m$ für die Reste alle Werte zwischen $0, \cdots, m - 1$ vor (vollständige m-Permutation).

Die multiplikative Inverse von 5 **(mod** 17**) ist** 7**, die Inverse von** 6 **(mod** 17**) ist** 3**.**

Tabelle 2: Multiplikationstabelle modulo 13 **(für** $a = 5$ **und** $a = 6$**)**

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$5 * i$	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
Rest	5	10	2	7	12	4	9	**1**	6	11	3	8	0	5	10	2	7	12
$6 * i$	6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108
Rest	6	12	5	11	4	10	3	9	2	8	**1**	7	0	6	12	5	11	4

Da sowohl 5 als auch 6 auch zum Modul $m = 13$ jeweils teilerfremd sind, kommen zwischen $i = 1, \cdots, m$ für die Reste alle Werte zwischen $0, \cdots, m - 1$ vor.

Die multiplikative Inverse von 5 **(mod** 13**) ist** 8**, die Inverse von** 6 **(mod** 13**) ist** 11**.**

Die folgende Tabelle enthält ein Beispiel dafür, wo der Modul m und die Zahl ($a = 6$) *nicht* teilerfremd sind.

Tabelle 3: Multiplikationstabelle modulo 12 (für $a = 5$ und $a = 6$)

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
5*i	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
Rest	5	10	3	8	**1**	6	11	4	9	2	7	0	5	10	3	8	1	6
6*i	6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108
Rest	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0

Berechnet wurde $(5 * i) \pmod{12}$ und $(6 * i) \pmod{12}$. Da 6 und der Modul $m = 12$ nicht teilerfremd sind, kommen zwischen $i = 1, \cdots, m$ nicht alle Werte zwischen $0, \cdots, m-1$ vor und 6 hat mod 12 auch keine Inverse!

Die multiplikative Inverse von 5 (mod 12) ist 5. Die Zahl 6 hat keine Inverse (mod 12).

3.6.3 Potenzieren

Das Potenzieren ist in der modularen Arithmetik definiert als wiederholtes Multiplizieren – wie üblich, nur dass jetzt Multiplizieren etwas anderes ist. Es gelten mit kleinen Einschränkungen die üblichen Rechenregeln wie

$$\begin{aligned} a^{b+c} &= a^b * a^c, \\ (a^b)^c &= a^{b*c} = a^{c*b} = (a^c)^b. \end{aligned}$$

Analog der modularen Addition und der modularen Multiplikation funktioniert das modulare Potenzieren:

$$3^2 \equiv 9 \equiv 4 \pmod{5}.$$

Auch aufeinanderfolgendes Potenzieren geht analog:

Beispiel 1:

$$(4^3)^2 \equiv 64^2 \equiv 4096 \equiv 1 \pmod{5}.$$

(1) Reduziert man bereits **Zwischenergebnisse** modulo 5, kann man schneller[39] rechnen, muss aber auch aufpassen, da sich dann nicht immer alles wie in der gewöhnlichen

[39] Die Rechenzeit der Multiplikation zweier Zahlen hängt normalerweise von der Länge der Zahlen ab. Dies sieht man, wenn man nach der Schulmethode z.B. $474 * 228$ berechnet: Der Aufwand steigt quadratisch, da $3 * 3$ Ziffern multipliziert werden müssen. Durch die Reduktion der Zwischenergebnisse werden die Zahlen deutlich kleiner.

Arithmetik verhält.

$$\begin{aligned}(4^3)^2 &\equiv (4^3 \pmod 5)^2 \pmod 5\\ &\equiv (64 \pmod 5)^2 \pmod 5\\ &\equiv 4^2 \pmod 5\\ &\equiv 16 \equiv 1 \pmod 5).\end{aligned}$$

(2) Aufeinanderfolgende Potenzierungen lassen sich in der gewöhnlichen Arithmetik auf eine einfache Potenzierung zurückführen, indem man die Exponenten miteinander multipliziert:

$$(4^3)^2 = 4^{3*2} = 4^6 = 4096.$$

In der modularen Arithmetik geht das nicht ganz so einfach, denn man erhielte:

$$(4^3)^2 \equiv 4^{3*2 \pmod 5} \equiv 4^{6 \pmod 5} \equiv 4^1 \equiv 4 \pmod 5).$$

Wie wir oben sahen, ist das richtige Ergebnis aber 1 !!

(3) Deshalb ist für das fortgesetzte Potenzieren in der modularen Arithmetik die Regel etwas anders: man multipliziert die Exponenten nicht in (mod m), sondern in (mod $J(m)$).

Mit $J(5) = 4$ ergibt sich:

$$(4^3)^2 \equiv 4^{3*2 \pmod{J(5)}} \equiv 4^{6 \bmod 4} \equiv 4^2 \equiv 16 \equiv 1 \pmod 5).$$

Das liefert das richtige Ergebnis.

Satz 3.6. $(a^b)^c \equiv a^{b*c \pmod{J(m)}} \pmod m$.

Beispiel 2:

$$3^{28} \equiv 3^{4*7} \equiv 3^{4*7 \pmod{10}} \equiv 3^8 \equiv 6561 \equiv 5 \pmod{11}.$$

3.6.4 Schnelles Berechnen hoher Potenzen

Bei RSA-Ver- und Entschlüsselungen[40] müssen hohe Potenzen modulo m berechnet werden. Schon die Berechnung (100^5) mod 3 sprengt den 32 Bit langen Long-Integer-Zahlenbereich, sofern man zur Berechnung von a^n getreu der Definition a tatsächlich n-mal mit sich selbst multipliziert. Bei sehr großen Zahlen wäre selbst ein schneller Computerchip mit einer einzigen Exponentiation länger beschäftigt als das Weltall existiert. Glücklicherweise gibt es für die Exponentiation (nicht aber für das Logarithmieren) eine sehr wirksame Abkürzung.

[40] Siehe Kapitel 3.10 Beweis des RSA-Verfahrens mit Euler-Fermat und Kapitel 3.13 Das RSA-Verfahren mit konkreten Zahlen.

Wird der Ausdruck anhand der Rechenregeln der modularen Arithmetik anders aufgeteilt, sprengt die Berechnung nicht einmal den 16 Bit langen Short-Integer-Bereich:

$$(a^5) \equiv (((a^2 \pmod m))^2 \pmod m) * a) \pmod m).$$

Dies kann man verallgemeinern, indem man den Exponenten binär darstellt. Beispielsweise würde die Berechnung von a^n für $n = 37$ auf dem naiven Wege 36 Multiplikationen erfordern. Schreibt man jedoch n in der Binärdarstellung als $100101 = 1 * 2^5 + 1 * 2^2 + 1 * 2^0$, so kann man umformen: $a^{37} = a^{2^5+2^2+2^0} = a^{2^5} * a^{2^2} * a^1$.

Beispiel 3: $87^{43} \pmod{103}$.
Da $43 = 32 + 8 + 2 + 1$, 103 prim , $43 < J(103)$ ist und

die Quadrate (mod 103) vorab berechnet werden können

$$\begin{aligned} 87^2 &\equiv 50 \pmod{103}, \\ 87^4 &\equiv 50^2 \equiv 28 \pmod{103}, \\ 87^8 &\equiv 28^2 \equiv 63 \pmod{103}, \\ 87^{16} &\equiv 63^2 \equiv 55 \pmod{103}, \\ 87^{32} &\equiv 55^2 \equiv 38 \pmod{103}, \end{aligned}$$

gilt[41]:

$$\begin{aligned} 87^{43} &\equiv 87^{32+8+2+1} \pmod{103} \\ &\equiv 87^{32} * 87^8 * 87^2 * 87 \pmod{103} \\ &\equiv 38 * 63 * 50 * 87 \equiv 85 \pmod{103}. \end{aligned}$$

Die Potenzen $(a^2)^k$ sind durch fortgesetztes Quadrieren leicht zu bestimmen. Solange sich a nicht ändert, kann ein Computer sie vorab berechnen und – bei ausreichend Speicherplatz – abspeichern. Um dann im Einzelfall a^n zu finden, muss er nur noch genau diejenigen $(a^2)^k$ miteinander multiplizieren, für die an der k-ten Stelle der Binärdarstellung von n eine Eins steht. Der typische Aufwand für $n = 600$ sinkt dadurch von 2^{600} auf $2 * 600$ Multiplikationen! Dieser häufig verwendete Algorithmus heißt „Square and Multiply".

3.6.5 Wurzeln und Logarithmen

Die Umkehrungen des Potenzierens sind ebenfalls definiert: Wurzeln und Logarithmen sind abermals ganze Zahlen, aber im Gegensatz zur üblichen Situation sind sie nicht nur mühsam, sondern bei sehr großen Zahlen in „erträglicher" Zeit überhaupt nicht zu berechnen.

Gegeben sei die Gleichung: $a \equiv b^c \pmod m$.

[41] In Appendix D finden Sie den Beispielcode zum Nachrechnen der Square-and-Multiply Methode mit Mathematica und Pari-GP.

a) Logarithmieren (Bestimmen von c) – Diskretes Logarithmus Problem:

Wenn man von den drei Zahlen a, b, c, welche diese Gleichung erfüllen, a und b kennt, ist jede bekannte Methode, c zu finden, ungefähr so aufwendig wie das Durchprobieren aller m denkbaren Werte für c – bei einem typischen m in der Größenordnung von 10^{180} für 600-stellige Binärzahlen ein hoffnungsloses Unterfangen. Genauer ist für geeignet große Zahlen m der Aufwand nach heutigem Wissensstand proportional zu $\exp\left(C * (\log m[\log\log m]^2)^{1/3}\right)$ mit einer Konstanten $C > 1$.

b) Wurzel-Berechnung (Bestimmen von b):

Ähnliches gilt, wenn b die Unbekannte ist und die Zahlen a und c bekannt sind.
Wenn die Eulerfunktion $J(m)$ bekannt ist, findet man leicht[42] d mit $c * d \equiv 1 \pmod{J(m)}$ und erhält mit Satz 3.6

$$a^d \equiv (b^c)^d \equiv b^{c*d} \equiv b^{c*d \ (mod\ J(m))} \equiv b^1 \equiv b \pmod{m}$$

die *c-te Wurzel* b von a.

Für den Fall, dass $J(m)$ in der Praxis nicht bestimmt werden kann[43], ist die Berechnung der c-ten Wurzel schwierig. Hierauf beruhen die Sicherheitsannahmen für das RSA-Kryptosystem (Siehe Kapitel 4.3.1: Das RSA-Verfahren oder Kapitel 3.10: Beweis des RSA-Verfahrens mit Euler-Fermat).

Dagegen ist der Aufwand für die Umkehrung von Addition und Multiplikation nur proportional zu $\log m$ beziehungsweise $(\log m)^2$. Potenzieren (zu einer Zahl x berechne x^a mit festem a) und Exponentiation (zu einer Zahl x berechne a^x mit festem a) sind also typische Einwegfunktionen (siehe Übersicht über Einwegfunktionen im Skript oder in diesem Artikel).

3.7 Gruppen und modulare Arithmetik über $\mathbb{Z}_n$ und $\mathbb{Z}_n^*$

In der Zahlentheorie und in der Kryptographie spielen mathematische „*Gruppen*" eine entscheidende Rolle. Von Gruppen spricht man nur, wenn für eine definierte Menge und eine definierte Relation (eine Operation wie Addition oder Multiplikation) die folgenden Eigenschaften erfüllt sind:

- Abgeschlossenheit
- Existenz des neutralen Elements
- Existenz des inversen Elements und
- Gültigkeit des Assoziativgesetzes.

Die abgekürzte mathematische Schreibweise lautet: $(G, +)$ oder $(G, *)$.

[42] Siehe Anhang A zu diesem Kapitel: der größte gemeinsame Teiler (ggT) von ganzen Zahlen.
[43] Nach dem ersten Hauptsatz der Zahlentheorie und Satz 3.11 kann man $J(m)$ mit Hilfe der Primfaktorzerlegung von m bestimmen.

Definition 3.6. $\mathbb{Z}_n$ *:*

$$\mathbb{Z}_n \textit{ umfasst alle ganzen Zahlen von } 0 \textit{ bis } n-1 : \mathbb{Z}_n = \{0, 1, 2, \cdots, n-2, n-1\}.$$

$\mathbb{Z}_n$ ist eine häufig verwendete endliche Gruppe aus den natürlichen Zahlen. Sie wird manchmal auch als Restemenge R modulo n bezeichnet.

Beispielsweise rechnen 32 Bit-Computer (übliche PCs) mit ganzen Zahlen direkt nur in einer endlichen Menge, nämlich in dem Wertebereich $0, 1, 2, \cdots, 2^{32} - 1$.

Dieser Zahlenbereich ist äquivalent zur Menge $\mathbb{Z}_{2^{32}}$.

3.7.1 Addition in einer Gruppe

Definiert man auf einer solchen Menge die Operation mod+ mit

$$a \text{ mod+ } b := (a + b) \pmod{n},$$

so ist die Menge $\mathbb{Z}_n$ zusammen mit der Relation mod+ eine Gruppe, denn es gelten die folgenden Eigenschaften einer Gruppe für alle Elemente von $\mathbb{Z}_n$:

- a mod+ b ist ein Element von $\mathbb{Z}_n$ (Abgeschlossenheit),
- $(a \text{ mod+ } b) \text{ mod+ } c \equiv a \text{ mod+ } (b \text{ mod+ } c)$ (mod+ ist assoziativ),
- das neutrale Element ist die 0.
- jedes Element $a \in \mathbb{Z}_n$ besitzt bezüglich dieser Operation ein Inverses, nämlich $n - a$ (denn es gilt: $a \text{ mod+ } (n - a) \equiv a + (n - a) \pmod{n} \equiv n \equiv 0 \pmod{n}$).

Da die Operation kommutativ ist, d.h. es gilt $(a \text{ mod+ } b) = (b \text{ mod+ } a)$, ist diese Struktur sogar eine „kommutative Gruppe".

3.7.2 Multiplikation in einer Gruppe

Definiert man in der Menge $\mathbb{Z}_n$ die Operation mod* mit

$$a \text{ mod* } b := (a * b) \pmod{n},$$

so ist $\mathbb{Z}_n$ zusammen mit dieser Operation **normalerweise keine** Gruppe, weil nicht für jedes n alle Eigenschaften erfüllt sind.

Beispiele:

a) In $\mathbb{Z}_{15}$ besitzt z.B. das Element 5 kein Inverses. Es gibt nämlich kein a mit $5 * a \equiv 1(\text{ mod } 15)$. Jedes Modulo-Produkt mit 5 ergibt auf dieser Menge $5, 10$ oder 0.

b) In $\mathbb{Z}_{55} \setminus \{0\}$ besitzen z.B. die Elemente 5 und 11 keine multiplikativen Inversen. Es gibt nämlich kein a aus $\mathbb{Z}_{55}$ mit $5 * a \equiv 1(\bmod 55)$ und kein a mit $11 * a \equiv 1 \bmod 55$. Das liegt daran, dass 5 und 11 nicht teilerfremd zu 55 sind. Jedes Modulo-Produkt mit 5 ergibt auf dieser Menge $5, 10, 15, \cdots, 50$ oder 0. Jedes Modulo-Produkt mit 11 ergibt auf dieser Menge $11, 22, 33, 44$ oder 0.

Dagegen gibt es Teilmengen von $\mathbb{Z}_n$, die bezüglich mod* eine Gruppe bilden. Wählt man sämtliche Elemente aus $\mathbb{Z}_n$ aus, die teilerfremd zu n sind, so ist diese Menge eine Gruppe bezüglich mod*. Diese Menge bezeichnet man mit $\mathbb{Z}_n^*$.

Definition 3.7. $\mathbb{Z}_n^*$ *:*

$$\mathbb{Z}_n^* := \{a \in \mathbb{Z}_n \big| \text{ggT}(a, n) = 1\}.$$

$\mathbb{Z}_n^*$ wird manchmal auch als *reduzierte Restemenge* R' modulo n bezeichnet.

Beispiel: Für $n = 10 = 2 * 5$ gilt:

vollständige Restmenge $R = \mathbb{Z}_n = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

reduzierte Restemenge $R' = \mathbb{Z}_n^* = \{1, 3, 7, 9\} \longrightarrow J(n) = 4$.

Bemerkung: R' bzw. $\mathbb{Z}_n^*$ ist immer eine echte Teilmenge von R bzw. $\mathbb{Z}_n$, da 0 immer Element von $\mathbb{Z}_n$, aber nie Element von $\mathbb{Z}_n^*$ ist. Da 1 (per Definition) und $n - 1$ immer teilerfremd zu n sind, sind sie stets Elemente beider Mengen.

Wählt man irgendein Element aus $\mathbb{Z}_n^*$ und multipliziert es mit jedem anderen Element von $\mathbb{Z}_n^*$, so sind die Produkte[44] alle wieder in $\mathbb{Z}_n^*$ und außerdem sind die Ergebnisse eine eindeutige Permutation der Elemente von $\mathbb{Z}_n^*$. Da die 1 immer Element von $\mathbb{Z}_n^*$ ist, gibt es in dieser Menge eindeutig einen „Partner", so dass das Produkt 1 ergibt. Mit anderen Worten:

Satz 3.7. *Jedes Element in* $\mathbb{Z}_n^*$ *hat eine multiplikative Inverse.*

Beispiel für $a = 3$ modulo 10 mit $\mathbb{Z}_n^* = \{1, 3, 7, 9\}$:

$$\begin{aligned} 3 &\equiv 3 * 1 \pmod{10}, \\ 9 &\equiv 3 * 3 \pmod{10}, \\ 1 &\equiv 3 * 7 \pmod{10}, \\ 7 &\equiv 3 * 9 \pmod{10}. \end{aligned}$$

Die eindeutige Invertierung (Umkehrbarkeit) ist eine notwendige Bedingung für die Kryptographie (siehe Kapitel 3.10: Beweis des RSA-Verfahrens mit Euler-Fermat).

[44] Dies ergibt sich aus der Abgeschlossenheit von $\mathbb{Z}_n^*$ bezüglich der Multiplikation und der ggT-Eigenschaft:
$[a, b \in \mathbb{Z}_n^*] \Rightarrow [((a * b) \pmod n) \in \mathbb{Z}_n^*]$, genauer:
$[a, b \in \mathbb{Z}_n^*] \Rightarrow [\text{ggT}(a, n) = 1, \text{ggT}(b, n) = 1] \Rightarrow [\text{ggT}(a * b, n) = 1] \Rightarrow [((a * b) \pmod n) \in \mathbb{Z}_n^*]$.

Eric Berne[45]*:*
Die mathematische Spielanalyse postuliert Spieler, die rational reagieren. Die transaktionale Spielanalyse dagegen befasst sich mit Spielen, die unrational, ja sogar **irrational und damit wirklichkeitsnäher** sind.

3.8 Euler-Funktion, kleiner Satz von Fermat und Satz von Euler-Fermat

3.8.1 Muster und Strukturen

So wie Mathematiker die Struktur $a * x \equiv b \bmod m$ untersuchen (s. Kapitel 3.5.2), so interessiert sie auch die Struktur $x^a \equiv b \bmod m$.

Auch hierbei ist insbesondere der Fall interessant, wenn $b = 1$ ist (also den Werte der multiplikativen Einheit annimmt) und wenn $b = x$ ist (also die Abbildung einen Fixpunkt hat).

3.8.2 Die Euler-Funktion

Bei vorgegebenem n ist die Anzahl der Zahlen aus der Menge $\{1, \cdots, n-1\}$, die zu n teilerfremd sind, gleich dem Wert der Euler[46]-Funktion $J(n)$.

Definition 3.8. *Die Euler-Funktion*[47] *$J(n)$ gibt die Anzahl der Elemente von $\mathbb{Z}_n^*$ an.*

$J(n)$ gibt auch an, wieviele ganze Zahlen in mod n multiplikative Inverse haben. $J(n)$ lässt sich berechnen, wenn man die Primfaktorzerlegung von n kennt.

Satz 3.8. *Für eine Primzahl gilt: $J(p) = p - 1$.*

Satz 3.9. *Ist n das Produkt zweier verschiedenen Primzahlen p und q, so gilt:*

$$J(p * q) = (p - 1) * (q - 1) \quad \text{oder} \quad J(p * q) = J(p) * J(q).$$

Dieser Fall ist für das RSA-Verfahren wichtig.

Satz 3.10. *Ist $n = p_1 * p_2 * \cdots * p_k$, wobei p_1 bis p_k verschiedene Primzahlen sind (d.h. $p_i \neq p_j$ für $i \neq j$), dann gilt (als Verallgemeinerung von Satz 3.9):*

$$J(n) = (p_1 - 1) * (p_2 - 1) * \cdots * (p_k - 1).$$

Satz 3.11. *Verallgemeinert gilt für jede Primzahl p und jedes n aus $\mathbb{N}$:*

1. $J(p^n) = p^{n-1} * (p - 1)$.

[45] Eric Berne, „Spiele der Erwachsenen", rororo, (c) 1964, S. 235.
[46] Leonhard Euler, schweizer Mathematiker, 15.4.1707 – 18.9.1783
[47] Wird oft auch als Eulersche Phi-Funktion $\Phi(n)$ geschrieben.

2. *Ist* $n = p_1^{e_1} * p_2^{e_2} * \cdots * p_k^{e_k}$*, wobei* p_1 *bis* p_k *verschiedene Primzahlen sind, dann gilt:*

$$J(n) = [(p_1^{e_1-1}) * (p_1 - 1)] * \cdots * [(p_k^{e_k-1}) * (p_k - 1)] = n * ([(p_1 - 1)/p_1] * \cdots * [(p_k - 1)/p_k]).$$

Beispiele:

- $n = 70 = 2 * 5 * 7 \Longrightarrow$ nach Satz 3.10: $J(n) = 1 \cdot 4 \cdot 6 = 24$.
- $n = 9 = 3^2 \Longrightarrow$ nach Satz 3.11: $J(n) = 3^1 \cdot 2 = 6$, weil $\mathbb{Z}_9^* = \{1, 2, 4, 5, 7, 8\}$.
- $n = 2.701.125 = 3^2 * 5^3 * 7^4 \Longrightarrow$ nach Satz 3.11: $J(n) = [3^1 * 2] * [5^2 * 4] * [7^3 * 6] = 1.234.800$.

3.8.3 Der Satz von Euler-Fermat

Für den Beweis des RSA-Verfahrens brauchen wir den Satz von Fermat und dessen Verallgemeinerung (Satz von Euler-Fermat) – vergleiche Kapitel 2.5.

Satz 3.12. Kleiner Satz von Fermat[48] *Sei* p *eine Primzahl und* a *eine beliebige ganze Zahl, dann gilt*

$$a^p \equiv a \pmod{p}.$$

Eine alternative Formulierung des kleinen Satzes von Fermat lautet: Sei p eine Primzahl und a eine beliebige ganze Zahl, die teilerfremd zu p ist, dann gilt:

$$a^{p-1} \equiv 1 \pmod{p}.$$

Satz 3.13. Satz von Euler-Fermat (Verallgemeinerung des kleines Satzes von Fermat) *Für alle Elemente* a *aus der Gruppe* $\mathbb{Z}_n^*$ *gilt (d.h.* a *und* n *sind natürliche Zahlen, die teilerfremd zueinander sind):*

$$a^{J(n)} \equiv 1 \pmod{n}.$$

Dieser Satz besagt, dass wenn man ein Gruppenelement (hier a) mit der Ordnung der Gruppe (hier $J(n)$) potenziert, ergibt sich immer das neutrale Element der Multiplikation (die Zahl 1).

Die 2. Formulierung des kleinen Satzes von Fermat ergibt sich direkt aus dem Satz von Euler, wenn n eine Primzahl ist.

Falls n das Produkt zweier verschiedenen Primzahlen ist, kann man mit dem Satz von Euler in bestimmten Fällen sehr schnell das Ergebnis einer modularen Potenz berechnen. Es gilt: $a^{(p-1)*(q-1)} \equiv 1 \pmod{pq}$.

Beispiele zur Berechnung einer modularen Potenz:

- Mit $2 = 1 * 2$ und $6 = 2 * 3$, wobei 2 und 3 jeweils prim; $J(6) = 2$, da nur 1 und 5 zu 6 teilerfremd sind folgt $5^2 \equiv 5^{J(6)} \equiv 1 \pmod{6}$, ohne dass man die Potenz berechnen musste.
- Mit $792 = 22 * 36$ und $23 * 37 = 851$, wobei 23 und 37 jeweils prim folgt $31^{792} \equiv 31^{J(23*37)} \equiv 31^{J(851)} \equiv 1 \pmod{851}$.

[48] Pierre de Fermat, französischer Mathematiker, 17.8.1601 – 12.1.1665

3.8.4 Bestimmung der multiplikativen Inversen

Eine weitere interessante Anwendung ist ein Sonderfall der Bestimmung der multiplikativen Inverse mit Hilfe des Satzes von Euler-Fermat (multiplikative Inverse werden ansonsten mit dem erweiterten Euklid'schen Algorithmus ermittelt).

Beispiel:
Finde die multiplikative Inverse von 1579 modulo 7351.

Nach Euler-Fermat gilt: $a^{J(n)} = 1 \bmod n$ für alle a aus $\mathbb{Z}_n^*$. Teilt man beide Seiten durch a, ergibt sich: $a^{J(n)-1} \equiv a^{-1} \pmod{n}$. Für den Spezialfall, dass der Modul prim ist, gilt $J(n) = p-1$. Also gilt für die modulare Inverse a^{-1} von a:

$$a^{-1} \equiv a^{J(n)-1} \equiv a^{(p-1)-1} \equiv a^{p-2} \pmod{p}.$$

Für unser Beispiel bedeutet das:

> Da der Modul 7351 prim ist, ist $p - 2 = 7349$.
> $1579^{-1} \equiv 1579^{7349} \pmod{p}$.

Durch geschicktes Zerlegen des Exponenten kann man diese Potenz relativ einfach berechnen (siehe Kapitel 3.6.4 Schnelles Berechnen hoher Potenzen):

> $7349 = 4096 + 2048 + 1024 + 128 + 32 + 16 + 4 + 1$
> $1579^{-1} \equiv 4716 \pmod{7351}$.

3.8.5 Fixpunkte modulo 26

Laut Satz 3.6 werden die arithmetischen Operationen von modularen Ausdrücken in den Exponenten modulo $J(n)$ und nicht modulo n durchgeführt[49].

Wenn man in $a^{e*d} \equiv a^1 \pmod{n}$ die Inverse z.B. für den Faktor e im Exponenten bestimmen will, muss man modulo $J(n)$ rechnen.

Beispiel (mit Bezug zum RSA-Algorithmus):

Wenn man modulo 26 rechnet, aus welcher Menge können e und d kommen?

Lösung: Es gilt $e * d \equiv 1 \pmod{J(26)}$.

> Die reduzierte Restemenge $R' = \mathbb{Z}_{26}^* = \{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$ sind die Elemente in $\mathbb{Z}_{26}$, die eine multiplikative Inverse haben, also teilerfremd zu 26 sind.
>
> Die reduzierte Restemenge R'' enthält nur die Elemente aus R', die teilerfremd zu $J(26) = 12$ sind: $R'' = \{1, 5, 7, 11\}$.
>
> Für jedes e aus R'' gibt es ein d aus R'', so dass $a \equiv (a^e)^d \pmod{n}$.

[49] Für das folgende Beispiel wird der Modul wie beim RSA-Verfahren üblich mit „n" statt mit „m" bezeichnet.

Somit gibt es also zu jedem e in R'' genau ein (nicht unbedingt von e verschiedenes) Element, so dass gilt: $e * d \equiv 1 \pmod{J(26)}$.

Für alle e, die teilerfremd zu $J(n)$ sind, könnte man nach dem Satz von Euler-Fermat das d folgendermaßen berechnen:

$$\begin{aligned} d &\equiv e^{-1} \pmod{J(n)} \\ &\equiv e^{J(J(n))-1} \pmod{J(n)}, \quad \text{denn } a^{J(n)} \equiv 1 \pmod{n} \quad \text{entspricht } a^{J(n)-1} \equiv a^{-1} \pmod{n}. \end{aligned}$$

Besteht die Zahl n aus zwei verschiedenen Primfaktoren, so ist die Faktorisierung von n und das Finden von $J(n)$ ähnlich schwierig[50] (vergleiche Forderung 3 in Kapitel 3.10.1).

3.9 Multiplikative Ordnung und Primitivwurzel

Mathematiker stellen sich die Frage, unter welchen Bedingungen ergibt die wiederholte Anwendung einer Operation das neutrale Element (vergleiche Strukturen und Muster).

Für die i-fach aufeinander folgende modulare Multiplikation einer Zahl a mit $i = 1, \cdots, m-1$ ergibt sich als Produkt das neutrale Element der Multiplikation (1) nur dann, wenn a und m teilerfremd sind. Der Wert von i, für den das Produkt $a^i = 1$ ist, heißt multiplikative Ordnung von a.

Die Multiplikative Ordnung (order) und die Primitivwurzel (primitive root) sind zwei nützliche Konstrukte (Konzepte) der elementaren Zahlentheorie.

Definition 3.9. *Die* **multiplikative Ordnung** $ord_m(a)$ *einer ganzen Zahl* $a \pmod{m}$ *(wobei* a *und* m *teilerfremd sind) ist die kleinste ganze Zahl* e*, für die gilt:* $a^e \equiv 1 \pmod{m}$.

Die folgende Tabelle zeigt, dass in einer multiplikativen Gruppe (hier $\mathbb{Z}_{11}^*$) nicht notwendig alle Zahlen die gleiche Ordnung haben: Die Ordnungen sind $1, 2, 5$ und 10. Dabei bemerkt man:

1. Die Ordnungen sind alle Teiler von 10.
2. Die Zahlen $a = 2, 6, 7$ und 8 haben die Ordnung 10. Man sagt diese Zahlen haben in $\mathbb{Z}_{11}^*$ **maximale Ordnung** .

Beispiel 1:
Die folgende Tabelle[51] zeigt die Werte a^i mod 11 für die Exponenten $i = 1, 2, \cdots, 10$ und für die Basen $a = 1, 2, \cdots, 10$ sowie den sich für jedes a daraus ergebenden Wert $ord_{11}(a)$:

[50] Für $n = pq$ mit $p \neq q$ gilt $J(n) = (p-1) * (q-1) = n - (p+q) + 1$. Ferner sind die Zahlen p und q Lösungen der quadratischen Gleichung $x^2 - (p+q)x + pq = 0$.
Sind nur n und $J(n)$ bekannt, so gilt $pq = n$ und $p + q = n + 1 - J(n)$. Man erhält somit die Faktoren p und q von n, indem man die quadratische Gleichung

$$x^2 + (J(n) - n - 1)x + n = 0$$

löst.

[51] In Anhang D zu diesem Kapitel finden Sie den Quelltext zur Bestimmung der Tabelle mit Mathematica und Pari-GP.

Tabelle 4: Werte von $a^i \bmod 11, 1 \leq a, i < 11$ und zugehörige Ordnung von a modulo m.

	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9	i=10	$ord_{11}(a)$
a=1	1	1	1	1	1	1	1	1	1	1	1
a=2	2	4	8	5	10	9	7	3	6	1	10
a=3	3	9	5	4	**1**	3	9	5	4	1	5
a=4	4	5	9	3	**1**	4	5	9	3	1	5
a=5	5	3	4	9	**1**	5	3	4	9	1	5
a=6	6	3	7	9	10	5	8	4	2	1	10
a=7	7	5	2	3	10	4	6	9	8	1	10
a=8	8	9	6	4	10	3	2	5	7	1	10
a=9	9	4	3	5	**1**	9	4	3	5	1	5
a=10	10	1	10	1	10	1	10	1	10	1	2

Aus der Tabelle kann man ersehen, dass z.B. die Ordnung von 3 modulo 11 den Wert 5 hat.

Definition 3.10. *Sind a und m teilerfremd und gilt $ord_m(a) = J(m)$, (d.h. a hat maximale Ordnung), dann nennt man a eine* **Primitivwurzel** *von m.*

Nicht zu jedem Modul m gibt es eine Zahl a, die eine Primitivwurzel ist. In der obigen Tabelle ist nur $a = 2, 6, 7$ und 8 bezüglich mod 11 eine Primitivwurzel ($J(11) = 10$).

Mit Hilfe der Primitivwurzel kann man die Bedingungen klar herausarbeiten, wann Potenzen modulo m *eindeutig invertierbar* und die Berechnung in den Exponenten handhabbar sind.

Die folgenden beiden Tabellen zeigen multiplikative Ordnung und Primitivwurzel modulo 45 und modulo 46.

Beispiel 2:
Die folgende Tabelle[52] zeigt die Werte a^i mod 45 für die Exponenten $i = 1, 2, \cdots, 12$ und für die Basen $a = 1, 2, \cdots, 12$ sowie den sich für jedes a daraus ergebenden Wert $ord_{45}(a)$.

Tabelle 5: Werte von a^i mod $45, 1 \leq a, i < 13$**:**

$a \setminus i$	1	2	3	4	5	6	7	8	9	10	11	12	$ord_{45}(a)$	$J(45)$
1	1	1	1	1	1	1	1	1	1	1	1	1	1	24
2	2	4	8	16	32	19	38	31	17	34	23	1	12	24
3	3	9	27	36	18	9	27	36	18	9	27	36	—	24
4	4	16	19	31	34	1	4	16	19	31	34	1	6	24
5	5	25	35	40	20	10	5	25	35	40	20	10	—	24
6	6	36	36	36	36	36	36	36	36	36	36	36	—	24
7	7	4	28	16	22	19	43	31	37	34	13	1	12	24
8	8	19	17	1	8	19	17	1	8	19	17	1	4	24
9	9	36	9	36	9	36	9	36	9	36	9	36	—	24
10	10	10	10	10	10	10	10	10	10	10	10	10	—	24
11	11	31	26	16	41	1	11	31	26	16	41	1	6	24
12	12	9	18	36	27	9	18	36	27	9	18	36	—	24

$J(45)$ berechnet sich nach Satz 3.11: $J(45) = J(3^2 * 5) = [3^1 * 2] * [1 * 4] = 24$.

Da 45 keine Primzahl ist, gibt es nicht für alle Werte von a eine "Multiplikative Ordnung" (z.B. für die nicht zu 45 teilerfremden Zahlen $3, 5, 6, 9, 10, 12, \cdots$, da $45 = 3^2 * 5$).

Beispiel 3:
Hat 7 eine Primitivwurzel modulo 45?

Die Voraussetzung/Bedingung ggT$(7, 45) = 1$ ist erfüllt. Aus der Tabelle (Werte von a^i mod 45) kann man ersehen, dass die Zahl 7 keine Primitivwurzel von 45 ist, denn $ord_{45}(7) = 12 \neq 24 = J(45)$.

[52] In Anhang D zu diesem Kapitel Finden Sie den Quelltext zur Berechnung der Tabelle mit Mathematica und Pari-GP.

Beispiel 4:
Die folgende Tabelle[53] beantwortet die Frage, ob die Zahl 7 eine Primitivwurzel von 46 ist. Die Voraussetzung/Bedingung $\text{ggT}(7, 46) = 1$ ist erfüllt.

Tabelle 6: Werte von $a^i \bmod 46, 1 \leq a, i < 23$**:**

$a\backslash i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	ord
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	16	32	18	36	26	6	12	24	2	4	8	16	32	18	36	26	6	12	24	2	–
3	3	9	27	35	13	39	25	29	41	31	1	3	9	27	35	13	39	25	29	41	31	1	3	11
4	4	16	18	26	12	2	8	32	36	6	24	4	16	18	26	12	2	8	32	36	6	24	4	–
5	5	25	33	27	43	31	17	39	11	9	45	41	21	13	19	3	15	29	7	35	37	1	5	22
6	6	36	32	8	2	12	26	18	16	4	24	6	36	32	8	2	12	26	18	16	4	24	6	–
7	7	3	21	9	17	27	5	35	15	13	45	39	43	25	37	29	19	41	11	31	33	**1**	7	22
8	8	18	6	2	16	36	12	4	32	26	24	8	18	6	2	16	36	12	4	32	26	24	8	–
9	9	35	39	29	31	3	27	13	25	41	1	9	35	39	29	31	3	27	13	25	41	1	9	11
10	10	8	34	18	42	6	14	2	20	16	22	36	38	12	28	4	40	32	44	26	30	24	10	–
11	11	29	43	13	5	9	7	31	19	25	45	35	17	3	33	41	37	39	15	27	21	1	11	22
12	12	6	26	36	18	32	16	8	4	2	24	12	6	26	36	18	32	16	8	4	2	24	12	–
13	13	31	35	41	27	29	9	25	3	39	1	13	31	35	41	27	29	9	25	3	39	1	13	11
14	14	12	30	6	38	26	42	36	44	18	22	32	34	16	40	8	20	4	10	2	28	24	14	–
15	15	41	17	25	7	13	11	27	37	3	45	31	5	29	21	39	33	35	19	9	43	1	15	22
16	16	26	2	32	6	4	18	12	8	36	24	16	26	2	32	6	4	18	12	8	36	24	16	–
17	17	13	37	31	21	35	43	41	7	27	45	29	33	9	15	25	11	3	5	39	19	1	17	22
18	18	2	36	4	26	8	6	16	12	32	24	18	2	36	4	26	8	6	16	12	32	24	18	–
19	19	39	5	3	11	25	15	9	33	29	45	27	7	41	43	35	21	31	37	13	17	1	19	22
20	20	32	42	12	10	16	44	6	28	8	22	26	14	4	34	36	30	2	40	18	38	24	20	–
21	21	27	15	39	37	41	33	3	17	35	45	25	19	31	7	9	5	13	43	29	11	1	21	22
22	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	–
23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	–

$J(46)$ berechnet sich nach Satz 3.9: $J(46) = J(2 * 23) = 1 * 22 = 22$. Die Zahl 7 ist eine Primitivwurzel von 46, denn $ord_{46}(7) = 22 = J(46)$.

Satz 3.14. [54,55] *Für einen Modul n und a teilerfremd zu n gilt: $\{a^i (mod\ n) | i = 1, \ldots, J(n)\}$ ist gleich der multiplikativen Gruppe Z_n^* genau dann, wenn $ord_n(a) = J(n)$.*

[53] In Anhang D zu diesem Kapitel Finden Sie den Quelltext zur Berechnung der Tabelle mit Mathematica und Pari-GP.

[54] Für Primmoduli p haben alle a mit $0 < a < p$ die Ordnung $J(p) = p - 1$. Vergleiche dazu Tabelle 5. In diesem Fall nimmt $a^i (\text{mod } n)$ alle Werte $1, \ldots, p - 1$ an. Dieses Ausschöpfen des Wertebereiches ist eine wichtige kryptographische Eigenschaft (vergleiche Satz 3.5). Hiermit wird eine Permutation $\pi(p - 1)$ festgelegt.

[55] Tabelle 6 zeigt, dass bei zusammengesetzten Moduli n nicht alle a die maximale Ordnung $J(n)$ haben. In diesem Beispiel haben nur $5, 7, 11, 15, 17, 19$ und 21 die Ordnung 22.

3.10 Beweis des RSA-Verfahrens mit Euler-Fermat

Mit dem Satz von Euler-Fermat kann man in der Gruppe $\mathbb{Z}_n^*$ das RSA[56,57]-Verfahren „beweisen".

3.10.1 Grundidee der Public Key-Kryptographie

Die Grundidee bei der Public Key-Kryptographie besteht darin, dass alle Teilnehmer ein unterschiedliches Paar von Schlüsseln (P und S) besitzen und man für alle Empfänger die öffentlichen Schlüssel publiziert. So wie man die Telefonnummer einer Person aus dem Telefonbuch nachschlägt, kann man den öffentlichen Schlüssel P (public) des Empfängers aus einem Verzeichnis entnehmen. Außerdem hat jeder Empfänger einen geheimen Schlüssel S (secret), der zum Entschlüsseln benötigt wird und den niemand sonst kennt. Möchte der Sender eine Nachricht M (message) schicken, verschlüsselt er diese Nachricht mit dem öffentlichen Schlüssel P des Empfängers, bevor er sie abschickt:

der Chiffretext C (ciphertext) ergibt sich mit $C = E(P; M)$, wobei E (encryption) die Verschlüsselungsvorschrift ist. Der Empfänger benutzt seinen privaten Schlüssel S, um die Nachricht wieder mit der Entschlüsselungsvorschrift D (decryption) zu entschlüsseln: $M = D(S; C)$.

Damit dieses System mit jeder Nachricht M funktioniert, müssen folgende 4 **Forderungen** erfüllt sein:

1. $D(S; E(P; M)) = M$ für jedes M (Umkehrbarkeit) und M nimmt „sehr viele" verschiedene Werte an.
2. Alle (S, P)-Paare aller Teilnehmer sind verschieden (d.h. es muss viele davon geben).
3. Der Aufwand, S aus P herzuleiten, ist mindestens so hoch, wie das Entschlüsseln von M ohne Kenntnis von S.
4. Sowohl C als auch M lassen sich relativ einfach berechnen.

Die 1. Forderung ist eine generelle Bedingung für alle kryptographischen Verschlüsselungsalgorithmen.

---

[56] Das RSA-Verfahren ist das verbreitetste asymmetrische Kryptoverfahren. Es wurde 1978 von Ronald Rivest, Adi Shamir und Leonard Adleman entwickelt und kann sowohl zum Signieren wie zum Verschlüsseln eingesetzt werden. Kryptographen assoziieren mit der Abkürzung „RSA" immer dieses Verfahren - die folgende Anmerkung soll eher humorvoll zeigen, dass man jede Buchstabenkombination mehrfach belegen kann: in Deutschland gibt es sehr Interessen-lastige Diskussionen im Gesundheitswesen. Dabei wird mit „RSA" der vom Gesundheitsministerium geregelte „**R**isiko**S**trukturA**A**usgleich" in der gesetzlichen Krankenversicherung mit einem jährlichen Volumen von über 24 Milliarden DM bezeichnet.

[57] In der Literatur und in Filmen werden nicht nur klassische Verfahren benutzt (wie z. B. die geheime Nachricht, die Sherlock Holmes in der Geschichte „Die tanzenden Männchen" von Arthur Conan Doyle löst), sondern auch moderne Verfahren: z. B. in

- dem Film „Das Kartenhaus", 1992, wo sich die Autisten mit Hilfe von 5- und 6-stelligen Primzahlen unterhalten,
- der Geschichte „Der Dialog der Schwestern" von Dr. C. Elsner, 1999 (als PDF dem CrypTool-Paket beigelegt), wo die Heldinnen sich vertraulich mit einer Variante des RSA-Verfahrens unterhalten.

Die 2. Forderung kann leicht sichergestellt werden, weil es „sehr" viele Primzahlen gibt[58] und weil dies durch eine zentrale Stelle, die Zertifikate ausgibt, sichergestellt werden kann.

Die letzte Forderung macht das Verfahren überhaupt erst anwendbar. Dies geht, weil die modulare Potenzierung in linearer Zeit möglich ist (da die Zahlen längenbeschränkt sind).

Während Whitfield Diffie und Martin Hellman schon 1976 das generelle Schema formulierten, fanden erst Rivest, Shamir und Adleman ein konkretes Verfahren, das alle vier Bedingungen erfüllte.

3.10.2 Funktionsweise des RSA-Verfahrens

Man kann die Einzelschritte zur Durchführung des RSA-Verfahren folgendermaßen beschreiben (s. [Eckert2003, S. 213 ff] und [Sedgewick1990, S. 338 ff]). Schritt 1 bis 3 sind die Schlüsselerzeugung, Schritt 4 und 5 sind die Verschlüsselung, 6 und 7 die Entschlüsselung:

1. Wähle zufällig 2 verschiedene Primzahlen[59,60] p und q und berechne $n = p * q$[61]. Der Wert n wird als RSA-Modul bezeichnet[62].

2. Wähle zufällig $e \in \{2, \cdots, n-1\}$, so dass gilt:
   e ist teilerfremd zu $J(n) = (p-1) * (q-1)$. Zum Beispiel kann man e so wählen, dass gilt: $\max(p, q) < e < J(n) - 1$[63]. Danach kann man p und q „wegwerfen".

---

[58] Nach dem Primzahlsatz (prime number theorem) von Legendre und Gauss gibt es bis zur Zahl n asymptotisch $n/\ln(n)$ Primzahlen. Dies bedeutet beispielsweise: es gibt $6,5 * 10^{74}$ Primzahlen unterhalb von $n = 2^{256}$ $(1,1 * 10^{77})$ und $3,2 * 10^{74}$ Primzahlen unterhalb von $n = 2^{255}$. Zwischen 2^{256} und 2^{255} gibt es also allein $3,3 * 10^{74}$ Primzahlen mit genau 256 Bits. Diese hohe Zahl ist auch der Grund, warum man sie nicht einfach alle abspeichern kann.

[59] Compaq hatte in 2000 mit hohem Marketingaufwand das sogenannte Multiprime-Verfahren eingeführt. n war dabei das Produkt von zwei großen und einer relativ dazu kleinen Primzahl: $n = o * p * q$. Nach Satz 3.10 ist dann $J(n) = (o-1) * (p-1) * (q-1)$. Das Verfahren hat sich bisher nicht durchgesetzt.
Mit ein Grund dafür dürfte sein, dass Compaq ein Patent dafür angemeldet hat. Generell gibt es in Europa und in der Open Source Bewegung wenig Verständnis für Patente auf Algorithmen. Überhaupt kein Verständnis herscht außerhalb der USA, dass man auf den Sonderfall (3 Faktoren) eines Algorithmus (RSA) ein Patent beantragen kann, obwohl das Patent für den allgemeinen Fall schon fast abgelaufen ist.

[60] Für Primzahlen p und q mit $p = q$, und e, d mit $ed \equiv 1 \mod J(n)$ gilt i.a. nicht $(m^e)^d \equiv m \mod n$ für alle $m < n$. Seien z.B. $n = 5^2$ berechnet sich $J(n)$ nach Satz 3.5: $J(n) = 5 * 4 = 20$, $e = 3$, $d = 7$, $ed \equiv 21 \equiv 1 \mod J(n)$. Dann gilt $(5^3)^7 \equiv 0 \mod 25$.

[61] Das BSI (Bundesamt für Sicherheit in der Informationstechnik) empfiehlt, die Primfaktoren p und q ungefähr gleich groß zu wählen, aber nicht zu dicht beieinander, d.h. konkret etwa

$$0.5 < |\log_2(p) - \log_2(q)| < 30.$$

Die Primfaktoren werden unter Beachtung der genannten Nebenbedingung zufällig und unabhängig voneinander erzeugt (Siehe http://www.bsi.bund.de/esig/basics/techbas/krypto/bund02v7.pdf).

[62] In CrypTool wird der RSA-Modul mit einem großen „N" bezeichnet.

[63] Das Verfahren erlaubt es auch, d frei zu wählen und dann e zu berechnen. Dies hat aber praktische Nachteile. Normalerweise will man „schnell" verschlüsseln können und wählt deshalb einen öffentlichen Exponenten e so, dass er möglichst wenige binäre Einsen hat. Damit ist eine schnelle Exponentiation bei der Verschlüsselung möglich. Als besonders praktisch haben sich hierfür die Primzahlen $3, 17$ und 65537 erwiesen, da diese im Vergleich zum Modul n sehr kleine Bitlängen haben und in ihrer Binärdarstellung nur wenige Einsen aufweisen. Häufig verwendet wird die Zahl $65537 = 2^{16} + 1$, also binär: $10 \cdots 0 \cdots 01$ (diese Zahl ist prim und deshalb zu sehr vielen Zahlen teilerfremd).

3. Wähle $d \in \{1, \cdots, n-1\}$ mit $e * d = 1 \bmod J(n)$, d.h. d ist die multiplikative Inverse zu e modulo $J(n)$[64,65]. Danach kann man $J(n)$ „wegwerfen".

→ (n, e) ist der öffentliche Schlüssel P.

→ (n, d) ist der geheime Schlüssel S (es ist nur d geheim zu halten).

4. Zum Verschlüsseln wird die als (binäre) Zahl dargestellte Nachricht in Teile aufgebrochen, so dass jede Teilzahl kleiner als n ist.

5. Verschlüsselung des Klartextes (bzw. seiner Teilstücke) $M \in \{1, \cdots, n-1\}$:

$$C = E((n, e); M) := M^e \bmod n.$$

6. Zum Entschlüsseln wird das binär als Zahl dargestellte Chiffrat in Teile aufgebrochen, so dass jede Teilzahl kleiner als n ist.

7. Entschlüsselung des Chiffretextes (bzw. seiner Teilstücke) $C \in \{1, \cdots, n-1\}$:

$$M = D((n, d); C) := C^d \bmod n.$$

Die Zahlen d, e, n sind normalerweise sehr groß (z.B. d und e 300 Bit, n 600 Bit).

Bemerkung:

Die Sicherheit des RSA-Verfahrens hängt wie bei allen Public Key-Verfahren davon ab, dass man den privaten Key d nicht aus dem öffentlichen Key (n, e) berechnen kann.

Beim RSA-Verfahren bedeutet dies, dass

1. $J(n)$ für große zusammengesetzte n schwer zu berechnen ist, und
2. n für große n nur schwer in seine Primfaktoren zerlegt werden kann (Faktorisierungsproblem).

3.10.3 Beweis der Forderung 1 (Umkehrbarkeit)

Für Schlüsselpaare (n, e) und (n, d), die die in den Schritten 1 bis 3 des RSA-Verfahrens festgelegten Eigenschaften besitzen, muss für alle $M < n$ gelten:

$$M \equiv (M^e)^d \pmod{n} \quad \text{wobei} \quad (M^e)^d = M^{e*d}.$$

Das heißt, der oben angegebene Dechiffrieralgorithmus arbeitet korrekt.

Zu zeigen ist also: $M^{e*d} = M \pmod{n}$:

Wir zeigen das in 3 Schritten (s. [Beutelspacher1996, S. 131ff]).

Schritt 1:

[64] Aus Sicherheitsgründen darf d nicht zu klein sein.

[65] Je nach Implementierung wird zuerst d oder zuerst e bestimmt.

Im ersten Schritt zeigen wir:

$$M^{e*d} \equiv M \pmod{p}.$$

Dies ergibt sich aus den Voraussetzungen und dem Satz von Euler-Fermat (Satz 3.13). Da $n = p*q$ und $J(p*q) = (p-1)*(q-1)$ und da e und d so gewählt sind, dass $e*d \equiv 1 \pmod{J(n)}$, gibt es eine ganze Zahl k, so dass gilt: $e*d = 1 + k*(p-1)*(q-1)$.

$$\begin{aligned} M^{e*d} &\equiv M^{1+k*J(n)} \equiv M * M^{k*J(n)} \equiv M * M^{k*(p-1)*(q-1)} \pmod{p} \\ &\equiv M * (M^{p-1})^{k*(q-1)} \pmod{p} \quad \text{aufgrund des kleinen Fermat : } M^{p-1} \equiv 1 \pmod{p} \\ &\equiv M * (1)^{k*(q-1)} \pmod{p} \\ &\equiv M \pmod{p}. \end{aligned}$$

Die Voraussetzung für die Anwendung des vereinfachten Satzes von Euler-Fermat (kleiner-Fermat Satz 3.12) war, dass M und p teilerfremd sind.

Da das im allgemeinen nicht gilt, müssen wir noch betrachten, was ist, wenn M und p nicht teilerfremd sind: da p eine Primzahl ist, muss dann notwendigerweise p ein Teiler von M sein. Das heißt aber:

$$M \equiv 0 \pmod{p}$$

Wenn p die Zahl M teilt, so teilt p erst recht M^{e*d}. Also ist auch:

$$M^{e*d} \equiv 0 \pmod{p}.$$

Da p sowohl M als auch M^{e*d} teilt, teilt er auch ihre Differenz:

$$(M^{e*d} - M) \equiv 0 \pmod{p}.$$

Und damit gilt auch in diesem Spezialfall unsere zu beweisende Behauptung.

Schritt 2:

Völlig analog beweist man: $M^{e*d} \equiv M \pmod{q}$.

Schritt 3:

Nun führen wir die Behauptungen aus (a) und (b) zusammen für $n = p*q$, um zu zeigen:

$$M^{e*d} \equiv M \pmod{n} \text{ für alle } M < n.$$

Nach (a) und (b) gilt $(M^{e*d} - M) \equiv 0 \pmod{p}$ und $(M^{e*d} - M) \equiv 0 \pmod{q}$, also teilen p und q jeweils dieselbe Zahl $z = (M^{e*d} - M)$. Da p und q **verschiedenen** Primzahlen sind, muss dann auch ihr Produkt diese Zahl z teilen. Also gilt:

$$(M^{e*d} - M) \equiv 0 \pmod{p*q} \quad \text{oder} \quad M^{e*d} \equiv M \pmod{p*q} \quad \text{oder} \quad M^{e*d} \equiv M \pmod{n}.$$

□

1. Bemerkung:
Man kann die 3 Schritte auch kürzer zusammenfassen, wenn man Satz 3.13 (Euler-Fermat) benutzt (also nicht den vereinfachten Satz, wo $n = p$ gilt und der dem kleinen Satz von Fermat

entspricht):

$$(M^e)^d \equiv M^{e*d} \equiv M^{(p-1)(q-1)*k+1} \equiv (\underbrace{M^{(p-1)(q-1)}}_{\equiv M^{J(n)} \equiv 1 \pmod{n}})^k * M \equiv 1^k * M \equiv M \pmod{n}.$$

2. Bemerkung:
Beim Signieren werden die gleichen Operationen durchgeführt, aber zuerst mit dem geheimen Schlüssel d, und dann mit dem öffentlichen Schlüssel e. Das RSA-Verfahren ist auch für die Erstellung von digitalen Signaturen einsetzbar, weil gilt:

$$M \equiv (M^d)^e \pmod{n}.$$

3.11 Zur Sicherheit des RSA-Verfahrens[66]

Die Eignung des RSA-Verfahrens für digitale Signaturen und Verschlüsselung gerät immer wieder in die Diskussion, z.B. im Zusammenhang mit der Veröffentlichung aktueller Faktorisierungserfolge. Ungeachtet dessen ist das RSA-Verfahren seit seiner Veröffentlichung vor mehr als 20 Jahren unangefochtener De-facto-Standard (vgl. 6.1).

Die Sicherheit des RSA-Verfahrens basiert - wie die aller kryptographischen Verfahren - auf den folgenden 4 zentralen Säulen:

- der Komplexität des dem Problem zugrunde liegenden zahlentheoretischen Problems (hier der Faktorisierung großer Zahlen),
- der Wahl geeigneter Sicherheitsparameter (hier der Länge des Moduls n),
- der geeigneten Anwendung des Verfahrens sowie der Schlüsselerzeugung und
- der korrekten Implementierung des Algorithmus.

Die Anwendung und Schlüsselerzeugung wird heute gut beherrscht. Die Implementierung ist auf Basis einer Langzahlarithmetik sehr einfach.

In den folgenden Abschnitten werden die ersten beiden Punkte näher untersucht.

3.11.1 Komplexität

Ein erfolgreiches Entschlüsseln oder eine Signaturfälschung — ohne Kenntnis des geheimen Schlüssels — erfordert, die e-te Wurzel mod n zu ziehen. Der geheime Schlüssel, nämlich die multiplikative Inverse zu e mod $J(n)$, kann leicht bestimmt werden, wenn die Eulersche Funktion $J(n)$ bekannt ist. $J(n)$ wiederum lässt sich aus den Primfaktoren von n berechnen. Das Brechen des RSA-Verfahrens kann daher nicht schwieriger sein als das Faktorisieren des Moduls n.

[66] Große Teile der Kapitel 3.11.1 und 3.11.2 sind angelehnt an den Artikel „Vorzüge und Grenzen des RSA-Verfahrens“von F. Bourseau, D. Fox und C. Thiel [Bourseau2002].

Das beste heute bekannte Verfahren ist eine Weiterentwicklung des ursprünglich für Zahlen mit einer bestimmten Darstellung (z.B. Fermatzahlen) entwickelten General Number Field Sieve (GNFS). Die Lösungskomplexität des Faktorisierungsproblems liegt damit asymptotisch bei

$$O(l) = e^{c\cdot(l\cdot\ln 2)^{1/3}\cdot(\ln(l\cdot\ln(2))^{2/3}+o(l)}$$

Siehe:

A. Lenstra, H. Lenstra: *The development of the Number Field Sieve* [Lenstra1993].
Robert D. Silverman: *A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths* [Silverman2000].

Die Formel zeigt, dass das Faktorisierungsproblem zur Komplexitätsklasse der Probleme mit subexponentieller Berechnungskomplexität gehört (d.h. der Lösungsaufwand wächst asymptotisch nicht so stark wie e^l oder 2^l, sondern echt schwächer, z. B. wie $e^{\sqrt{l}}$). Diese Einordnung entspricht dem heutigen Kenntnisstand, sie bedeutet jedoch nicht, dass das Faktorisierungsproblem möglicherweise nicht auch mit (asymptotisch) polynominellem Aufwand gelöst werden kann (s. 3.11.3).

$O(l)$ gibt die Zahl der durchschnittlich erforderlichen Prozessor-Operationen abhängig von der Bitlänge l der zu faktorisierenden Zahl n an. Für den besten allgemeinen Faktorisierungsalgorithmus ist die Konstante $c = (64/9)^{1/173} = 1,923$.

Die umgekehrte Aussage, dass das RSA-Verfahren ausschließlich durch eine Faktorisierung von n gebrochen werden kann, ist bis heute nicht bewiesen. Zahlentheoretiker halten das "RSA-Problem" und das Faktorisierungsproblem für komplexitätstheoretisch äquivalent.

Siehe: *Handbook of Applied Cryptography* [Menezes2001].

3.11.2 Sicherheitsparameter aufgrund der Faktorisierungserfolge

Die Komplexität wird im wesentlichen von der Länge l des Moduls n bestimmt.

- 1994 wurde mit einer verteilten Implementierung des 1982 von Pomerance entwickelten Quadratic Sieve-Algorithmus (QS) nach knapp 8 Monaten der 1977 veröffentlichte 129-stellige RSA-Modul (428 Bit) faktorisiert.

  Siehe:

  C. Pomerance: *The quadratic sieve factoring algorithm* [Pomerance1984].

- 1999 wurde mit dem von Buhler, Lenstra und Pomerance entwickelten General Number Field Sieve-Algorithmus (GNFS), der ab ca. 116 Dezimalstellen effizienter ist als QS, nach knapp 5 Monaten ein 155-stelliger Modul (512 Bit) faktorisiert.

  Siehe:

  J.P. Buhler, H.W. Lenstra, C. Pomerance: *Factoring integers with the number field sieve* [Buhler1993].

Damit wurde klar, dass eine Modullänge von 512 Bit keinen Schutz mehr vor Angreifern darstellt.

In den letzten 20 Jahren wurden also erhebliche Fortschritte gemacht. Abschätzungen über die zukünftige Entwicklung der Sicherheit unterschiedlicher RSA-Modullängen differieren und hängen von verschiedenen Annahmen ab:

- Entwicklung der Rechnergeschwindigkeit (Gesetz von Moore: Verdopplung der Rechnerleistung alle 18 Monate) und des Grid-Computing.
- Entwicklung neuer Algorithmen.

Selbst ohne neue Algorithmen wurden in den letzten Jahren durchschnittlich ca. 10 Bit mehr pro Jahr berechenbar. Größere Zahlen erfordern mit den heute bekannten Verfahren einen immer größeren Arbeitsspeicher für die Lösungsmatrix. Dieser Speicherbedarf wächst mit der Wurzel des Rechenzeitbedarfs, also ebenfalls subexponentiell. Da in den letzten Jahren der verfügbare Hauptspeicher ebenso wie die Rechengeschwindigkeit exponentiell gewachsen ist, dürfte hierdurch kein zusätzlicher Engpass entstehen.

In dem oben erwähnten Artikel [Bourseau2002] prognostiziert Dirk Fox[67] einen annähernd linearen Verlauf der Faktorisierungserfolge unter Einbeziehung aller Faktoren: Pro Jahr kommen durchschnittlich 20 Bit dazu.

3.11.3 Weitere aktuelle Forschungsergebnisse aus der Zahlentheorie

Primzahlen sind Teil vieler hochaktueller Forschungsgebiete der Zahlentheorie. Die Fortschritte bei der Faktorisierung sind größ als noch vor 5 Jahren geschätzt – sie gehen nicht nur auf das Konto schnellerer Rechner, sondern sie sind auch in neuen Erkenntnissen begründet.

Die Sicherheit des RSA-Algorithmus basiert auf der empirischen Beobachtung, dass die Faktorisierung großer ganzer Zahlen ein schwieriges Problem ist. Besteht wie beim RSA-Algorithmus der zugrunde liegende Modul n aus dem Produkt zweier großer Primzahlen p, q (typische Längen: p, q $500 - 600$ bit, n 1024 bit), so lässt sich $n = pq$ aus p und q leicht bestimmen, jedoch ist es mit den bisher bekannten Faktorisierungsalgorithmen nicht möglich, p, q aus n zu gewinnen. Nur mit Kenntnis von p und q lässt sich jedoch der private aus dem öffentlichen Schlüssel ermitteln.

Die Entdeckung eines Algorithmus zur effizienten Faktorisierung von Produkten $n = pq$ großer Primzahlen würde daher den RSA-Algorithmus wesentlich beeinträchtigen. Je nach Effizienz der Faktorisierung im Vergleich zur Erzeugung von p, q, n müsste der verwendete Modul n (z.Zt. 1024 bit) erheblich vergrößert oder — im Extremfall — auf den Einsatz des RSA ganz verzichtet werden.

[67] Seine Firma Secorvo GmbH hat eine Stellungnahme zur Schlüssellängenempfehlung des BSI für den Bundesanzeiger abgegeben. In Kapitel 2.3.1 ihrer Stellungnahme geht sie sehr kompetent und verständlich auf RSA-Sicherheit ein (existiert nur in Deutsch):
http://www.secorvo.de/publikat/stellungnahme-algorithmenempfehlung-020307.pdf

Das Papier von Bernstein und seine Auswirkungen auf die Sicherheit des RSA-Algorithmus

Die im November 2001 veröffentlichte Arbeit "Circuits for integer factorization: a proposal" (siehe http://cr.yp.to/djb.html) von D.J. Bernstein [Bernstein2001] behandelt das Problem der Faktorisierung großer Zahlen. Die Kernaussage des Papers besteht darin, dass es möglich ist, die Implementierung des General Number Field Sieve-Algorithmus (GNFS) so zu verbessern, dass mit gleichem Aufwand wie bisher Zahlen mit 3-mal größerer Stellenzahl (Bit-Länge) faktorisiert werden können.

Wesentlich bei der Interpretation des Resultats ist die Definition des Aufwandes: Als Aufwand wird das Produkt von benötigter Rechenzeit und Kosten der Maschine (insbesondere des verwendeten Speicherplatzes) angesetzt. Zentral für das Ergebnis des Papiers ist die Beobachtung, dass ein wesentlicher Teil der Faktorisierung auf Sortierung zurückgeführt werden kann und mit dem Schimmlerschen Sortierschema ein Algorithmus zur Verfügung steht, der sich besonders gut für den Einsatz von Parallelrechnern eignet. Am Ende des Abschnittes 3 gibt Bernstein konkret an, dass die Verwendung von m^2 Parallelrechnern mit jeweils gleicher Menge an Speicherplatz mit Kosten in der Größenordnung von m^2 einhergeht — genau so wie ein einzelner Rechner mit m^2 Speicherzellen. Der Parallelrechner bewältigt die Sortierung von m^2 Zahlen jedoch (unter Verwendung der o. g. Sortierverfahrens) in Zeit proportional zu m, wohingegen der Einprozessorrechner Zeit proportional m^2 benötigt. Verringert man daher den verwendeten Speicherplatz und erhöht — bei insgesamt gleich bleibenden Kosten — die Anzahl der Prozessoren entsprechend, verringert sich die benötigte Zeit um die Größenordnung $1/m$. In Abschnitt 5 wird ferner angeführt, dass der massive Einsatz der parallelisierten Elliptic Curve-Methode von Lenstra die Kosten der Faktorisierung ebenfalls um eine Größenordnung verringert (ein Suchalgorithmus hat dann quadratische statt kubische Kosten). Alle Ergebnisse von Bernstein gelten nur asymptotisch für große Zahlen n. Leider liegen keine Abschätzungen über den Fehlerterm, d.h. die Abweichung der tatsächlichen Zeit von dem asymptotischen Wert, vor — ein Mangel, den auch Bernstein in seinem Papier erwähnt. Daher kann zur Zeit keine Aussage getroffen werden, ob die Kosten (im Sinne der Bernsteinschen Definition) bei der Faktorisierung z. Zt. verwendeter RSA-Zahlen ($1024-2048$ bit) bereits signifikant sinken würden.

Zusammenfassend lässt sich sagen, dass der Ansatz von Bernstein durchaus innovativ ist. Da die Verbesserung der Rechenzeit unter gleichbleibenden Kosten durch einen massiven Einsatz von Parallelrechnern erkauft wird, stellt sich die Frage nach der praktischen Relevanz. Auch wenn formal der Einsatz von einem Rechner über 1 sec und 1.000.000 Rechnern für je 1/1.000.000 sec dieselben Kosten erzeugen mag, ist die Parallelschaltung von 1.000.000 Rechnern praktisch nicht (oder nur unter immensen Fixkosten, insbesondere für die Vernetzung der Prozessoren) zu realisieren. Solche Fixkosten werden aber nicht in Ansatz gebracht. Die Verwendung verteilter Ansätze (distributed computing) über ein großes Netzwerk könnte einen Ausweg bieten. Auch hier müssten Zeiten und Kosten für Datenübertragung einkalkuliert werden.

Solange noch keine (kostengünstige) Hardware oder verteilte Ansätze entwickelt wurden, die auf dem Bernsteinschen Prinzip basieren, besteht noch keine akute Gefährdung des RSA. Es bleibt zu klären, ab welchen Größenordnungen von n die Asymptotik greift.

Arjen Lenstra, Adi Shamir et. al. haben das Bernstein-Paper analysiert [Lenstra2002]. Als Ergeb-

nis kommen Sie zu einer Bitlängen-Verbesserung der Faktorisierung um den Faktor 1.17 (anstatt Faktor 3 wie von Bernstein erwartet).

Die Zusammenfassung ihres Papers "Analysis of Bernstein's Factorization Circuit" lautet:

"... Bernstein proposed a circuit-based implementation of the matrix step of the number field sieve factorization algorithm. We show that under the non-standard cost function used in [1], these circuits indeed offer an asymptotic improvement over other methods but to a lesser degree than previously claimed: for a given cost, the new method can factor integers that are 1.17 times larger (rather than 3.01). We also propose an improved circuit design based on a new mesh routing algorithm, and show that for factorization of 1024-bit integers the matrix step can, under an optimistic assumption about the matrix size, be completed within a day by a device that costs a few thousand dollars. We conclude that from a practical standpoint, the security of RSA relies exclusively on the hardness of the relation collection step of the number field sieve."

Auch RSA Security kommt in ihrer Analyse der Bernstein-Arbeit [RSA Security 2002] vom 8. April 2002 erwartungsgemäß zu dem Ergebnis, dass RSA weiterhin als ungebrochen betrachtet werden kann.

Die Diskussion ist weiterhin im Gang.

Zum Zeitpunkt der Erstellung dieses Absatzes (Juni 2002) war nichts darüber bekannt, inwieweit die im Bernstein-Papier vorgeschlagenen theoretischen Ansätze realisiert wurden oder wieweit die Finanzierung seines Forschungsprojektes ist.

Verweise:

http://cr.yp.to/djb.html
http://www.counterpane.com/crypto-gram-0203.html#6
http://www.math.uic.edu

Das TWIRL-Device

Im Januar 2003 veröffentlichten Adi Shamir und Eran Tromer vom Weizmann Institute of Science den vorläufigen Draft *"Factoring Large Numbers with the TWIRL Device"*, in dem deutliche Bedenken gegen RSA-Schlüssellängen unter 1024 begründet werden [Shamir2003].

Das Abstract fasst ihre Ergebnisse folgendermaßen zusammen: "The security of the RSA cryptosystem depends on the difficulty of factoring large integers. The best current factoring algorithm is the Number Field Sieve (NFS), and its most difficult part is the sieving step. In 1999 a large distributed computation involving thousands of workstations working for many months managed to factor a 512-bit RSA key, but 1024-bit keys were believed to be safe for the next 15-20 years. In this paper we describe a new hardware implementation of the NFS sieving step ... which is 3-4 orders of magnitude more cost effective than the best previously published designs ... . Based on a detailed analysis of all the critical components (but without an actual implementation), we believe that the NFS sieving step for 1024-bit RSA keys can be completed in less than a year by a $10M device, and that the NFS sieving step for 512-bit RSA keys can be completed in less than ten minutes by a $10K device. Coupled with recent results about the difficulty of the NFS matrix step ... this raises some concerns about the security of these key sizes."

Damit erscheinen die aktuellen Empfehlungen des BSI, auf längere RSA-Schlüssellängen umzustellen, mehr als gerechtfertigt.

"Primes in P": Testen auf Primalität ist polynominal

Im August 2002 veröffentlichten die drei indischen Forscher M. Agrawal, N. Kayal und N. Saxena ihr Paper *"PRIMES in P"* über einen neuen Primzahltest-Algorithmus [Agrawal2002]. Sie entdeckten einen polynominalen deterministischen Algorithmus, um zu entscheiden, ob eine gegebene Zahl prim ist oder nicht.

Die Bedeutung dieser Entdeckung liegt darin, dass sie Zahlentheoretiker mit neuen Einsichten und Möglichkeiten für die weitere Forschung versorgt. Viele Menschen haben im Lauf der Jahrhunderte nach einem polynominalen Primzahltest gesucht, so dass dieses Ergebnis einen theoretischen Durchbruch darstellt. Es zeigt sich immer wieder, dass aus schon lange bekannten Fakten neue Ergebnisse generiert werden können.

Aber selbst die Autoren merken an, dass andere bekannte Algorithmen (z.B. ECPP) schneller sein können. Der neue Algorithmus funktioniert für alle positiven ganzen Zahlen. Dagegen verwendet das GIMPS-Projekt den Lucas-Lehmer-Primzahltest, der besondere Eigenschaften der Mersennezahlen ausnutzt. Dadurch ist der Lucas-Lehmer-Test viel schneller und erlaubt, Zahlen mit Millionen von Stellen zu testen, während generische Algorithmen auf Zahlen mit einigen tausend Stellen beschränkt sind. Nachteil der bisherigen schnellen Verfahren ist, dass sie probabilistisch sind, also ihr Ergebnis höchstwahrscheinlich, aber nicht ganz sicher ist.

Aktuelle Forschungsergebnisse dazu finden sich z.B. auf:
`http://www.mersenne.org/`
`http://fatphil.org/maths/AKS/`

> *Hermann Hesse*[68]*:*
> Damit das Mögliche entsteht, muss immer wieder das Unmögliche versucht werden.

3.11.4 Status der Faktorisierung von konkreten großen Zahlen

Eine hervorragende Übersicht über die Rekorde im Faktorisieren zusammengesetzter Zahlen mit unterschiedlichen Methoden findet sich auf der Webseite `http://www.crypto-world.com`. Der aktuelle Rekord (Stand Juni 2002) mit der GNFS-Methode (General Number Field Sieve) liegt in der Zerlegung einer 158-stelligen Zahl in ihre beiden Primfaktoren (diese haben 73 und 86 Dezimalstellen).

[68] deutsch/schweizerischer Schriftsteller und Nobelpreisträger, 02.07.1877−09.08.1962.

Dieser Rekord, aufgestellt am 18. Januar 2002 von Forschern der Universität Bonn[69], fand deutlich weniger Aufmerksamkeit in der Presse als die Lösung der RSA-Challenge[70] vom 22. Oktober 1999, als niederländische Forscher eine 155-stellige Zahl in ihre beiden 78-stellige Primfaktoren zerlegten (vergleiche Kapitel 3.11.2). Das liegt unter anderem daran, dass die 512 bit Zahl RSA-155 eine *magische* Grenze war.

Die Aufgabe der Bonner Wissenschaftler entsprang auch nicht einer Challenge, sondern die Aufgabe war, die letzten Primfaktoren der Zahl $2^{953} + 1$ zu finden (siehe "Wanted List" des Cunningham-Projekts[71]).

Die 6 kleineren, schon vorher gefundenen Primfaktoren dieser Zahl waren:

$$3, 1907, 425796183929,$$
$$1624700279478894385598779655842584377,$$
$$38023067385494413244321390912718281 21 \quad \text{und}$$
$$12806488683016667144480257612911587206 0027.$$

Wobei die drei kleinsten Faktoren leicht [72] bestimmt werden können. Die nächsten drei Primfaktoren wurden von P. Zimmerman[73], T. Grandlund[74] und R. Harley in den Jahren 1999 und 2000 mit der Methode der elliptischen Kurven gefunden.

Als letzter Faktor blieb der sogenannte Teiler „C158", von dem man bis dahin wusste, dass er zusammengesetzt ist, aber man kannte seine Primfaktoren nicht (die folgenden drei Zeilen sind eine einzige Zahl):

$$39505874583265144526419767800614481996020776460304936$$
$$45413937605157935562652945068360972784246821953509354$$
$$43058704902519956553357102097992264849779494429556 03$$

Die Faktorisierung von C158 ergab die beiden Primfaktoren:

$$33884958374667213943683932046721815228158303686049930480849258405552811 77$$

und

$$11658823406671259903148376558383270818131 01$$
$$22581463926004395209941313443341629245361 39.$$

Damit wurde die Zahl $2^{953} + 1$ vollständig in ihre 8 Primfaktoren zerlegt.

Verweise:

http://www.loria.fr/~zimmerma/records/gnfs158
http://www.crypto-world.com/FactorRecords.html
http://www.crypto-world.com/announcements/c158.txt

[69] http://www.uni-bonn.de/Aktuelles/Pressemitteilungen/pm02/pm035-02.html
[70] http://www.rsasecurity.com/rsalabs/challenges/factoring/numbers.html
[71] http://www.cerias.purdue.edu/homes/ssw/cun/
[72] Z.B. mit CrypTool über das Menü **Einzelverfahren \ RSA-Demonstration \ Faktorisieren einer Zahl**.
[73] http://www.loria.fr/~zimmerma/ecmnet
[74] http://www.swox.se/gmp/

Wie man sieht, sind die größten (aus 2 Primfaktoren) zusammengesetzten Zahlen, die man faktorisieren kann, deutlich kleiner als die Zahlen mit einer speziellen Struktur, für die Primzahltests in der Lage sind, Aussagen über ihre Primalität zu treffen (siehe Kapitel 2.4 und 2.5).

Joanne K. Rowling[75]*:*
Viel mehr als unsere Fähigkeiten sind es unsere Entscheidungen ..., die zeigen, wer wir wirklich sind.

3.12 Weitere zahlentheoretische Anwendungen in der Kryptographie

In der modernen Kryptographie werden die Ergebnisse der modularen Arithmetik extensiv angewandt. Hier werden exemplarisch einige wenige Beispiele aus der Kryptographie mit kleinen[76] Zahlen vorgestellt.

Die Chiffrierung eines Textes besteht darin, dass man aus einer Zeichenkette (Zahl) durch Anwenden einer Funktion (mathematische Operationen) eine andere Zahl erzeugt. Dechiffrieren heißt, diese Funktion umzukehren: aus dem Zerrbild, das die Funktion aus dem Klartext gemacht hat, das Urbild wiederherzustellen. Beispielsweise könnte der Absender einer vertraulichen Nachricht zum Klartext M eine geheimzuhaltende Zahl, den Schlüssel S, addieren und dadurch den Chiffretext C erhalten:

$$C = M + S.$$

Durch Umkehren dieser Operation, das heißt durch Subtrahieren von S, kann der Empfänger den Klartext rekonstruieren:

$$M = C - S.$$

Das Addieren von S macht den Klartext zuverlässig unkenntlich. Gleichwohl ist diese Verschlüsselung sehr schwach; denn wenn ein Abhörer auch nur ein zusammengehöriges Paar von Klar- und Chiffretext in die Hände bekommt, kann er den Schlüssel berechnen

$$S = C - M,$$

und alle folgenden mit S verschlüsselten Nachrichten mitlesen.
Der wesentliche Grund ist, dass Subtrahieren eine ebenso einfache Operation ist wie Addieren.

3.12.1 Einwegfunktionen

Wenn der Schlüssel auch bei gleichzeitiger Kenntnis von Klar- und Chiffretext nicht ermittelbar sein soll, braucht man eine Funktion, die einerseits relativ einfach berechenbar ist - man will ja chiffrieren können. Andererseits soll ihre Umkehrung zwar existieren (sonst würde beim Chiffrieren Information verlorengehen), aber de facto unberechenbar sein.

Was sind denkbare Kandidaten für eine solche **Einwegfunktion**? Man könnte an die Stelle der Addition die Multiplikation setzen; aber schon Grundschüler wissen, dass deren Umkehrung, die

[75] Joanne K. Rowling, „Harry Potter und die Kammer des Schreckens", Carlsen, (c) 1998, letztes Kapitel „Dobbys Belohnung", S. 343, Dumbledore.

[76] „Klein" bedeutet beim RSA-Verfahren, dass die Bitlängen der Zahlen sehr viel kleiner sind als 1024 Bit (das sind 308 Dezimalstellen). 1024 Bit gilt im Moment in der Praxis als Mindestlänge für einen sicheren Certification Authority-RSA-Modul.

Division, nur geringfügig mühsamer ist als die Multiplikation selbst. Man muss noch eine Stufe höher in der Hierarchie der Rechenarten gehen: Potenzieren ist immer noch eine relativ einfache Operation; aber ihre beiden Umkehrungen *Wurzelziehen* (finde b in der Gleichung $a = b^c$, wenn a und c bekannt sind) und *Logarithmieren* (in derselben Gleichung finde c, wenn a und b bekannt sind) sind so kompliziert, dass ihre Ausführung in der Schule normalerweise nicht mehr gelehrt wird.

Während bei Addition und Multiplikation noch eine gewisse Struktur wiedererkennbar ist, wirbeln Potenzierung und Exponentiation alle Zahlen wild durcheinander: Wenn man einige wenige Funktionswerte kennt, weiß man (anders als bei Addition und Multiplikation) noch kaum etwas über die Funktion im ganzen.

3.12.2 Das Diffie-Hellman Schlüsselaustausch-Protokoll (Key Exchange Protocol)

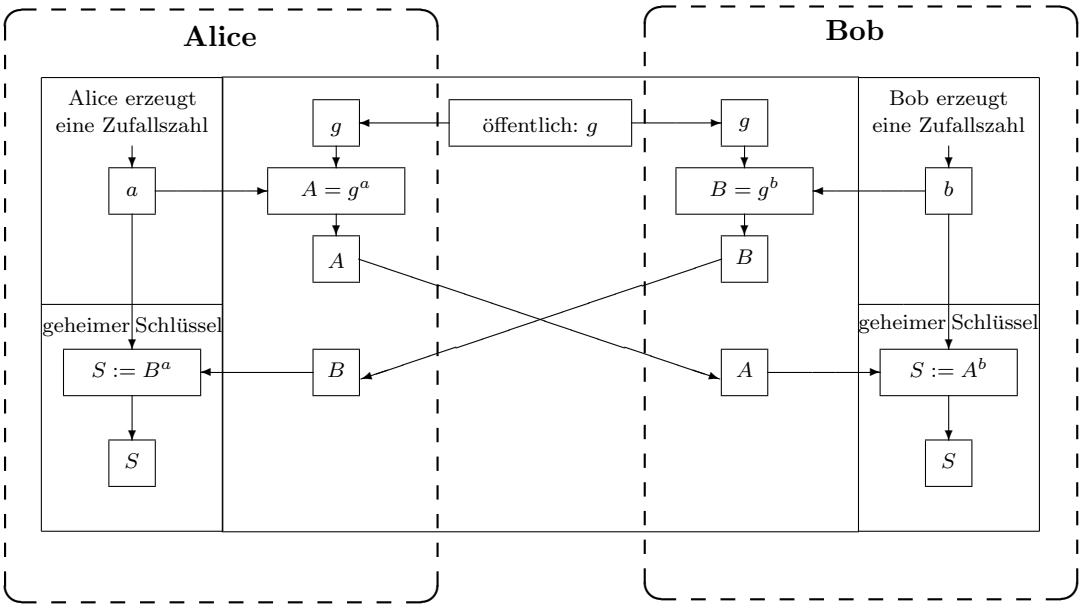
Das DH-Schlüsselaustauschprotokoll wurde 1976 in Stanford von Whitfield Diffie, Martin E. Hellman und Ralph Merkle erdacht[77].

Eine Einwegfunktion dient Alice und Bob[78] dazu, sich einen Schlüssel S, den Sessionkey, für die nachfolgende Verständigung zu verschaffen. Dieser ist dann ein Geheimnis, das nur diesen beiden bekannt ist. Alice wählt sich eine Zufallszahl a und hält sie geheim. Aus a berechnet sie mit der Einwegfunktion die Zahl $A = g^a$ und schickt sie an Bob. Der verfährt ebenso, indem er eine geheime Zufallszahl b wählt, daraus $B = g^b$ berechnet und an Alice schickt. Die Zahl g ist beliebig und darf öffentlich bekannt sein. Alice wendet die Einwegfunktion mit ihrer Geheimzahl a auf B an, Bob tut gleiches mit seiner Geheimzahl b und der empfangenen Zahl A.

Das Ergebnis S ist in beiden Fällen dasselbe, weil die Einwegfunktion kommutativ ist: $g^{a*b} = g^{b*a}$. Aber selbst Bob kann Alices Geheimnis a nicht aus den ihm vorliegenden Daten rekonstruieren, Alice wiederum Bobs Geheimnis b nicht ermitteln, und ein Lauscher, der g kennt und sowohl A als auch B mitgelesen hat, vermag daraus weder a noch b noch S zu berechnen.

[77] In CrypTool v1.3.04 ist dieses Austauschprotokoll visualisiert: Sie können die einzelnen Schritte mit konkreten Zahlen nachvollziehen per **Einzelverfahren \ Diffie-Hellman-Demo**.

[78] Alice und Bob werden standardmäßig als die beiden berechtigten Teilnehmer eines Protokolls bezeichnet (siehe [Schneier1996, Seite 23]).



Ablauf:

Alice und Bob wollen also einen geheimen Sessionkey S über einen abhörbaren Kanal aushandeln.

1. Sie wählen eine Primzahl p und eine Zufallszahl g, und tauschen diese Information offen aus.

2. Alice wählt nun a, eine Zufallszahl kleiner p und hält diese geheim.

   Bob wählt ebenso b, eine Zufallszahl kleiner p und hält diese geheim.

3. Alice berechnet nun $A \equiv g^a \pmod{p}$.
   Bob berechnet $B \equiv g^b \pmod{p}$.

4. Alice sendet das Ergebnis A an Bob.
   Bob sendet das Ergebnis B an Alice.

5. Um den nun gemeinsam zu benutzenden Sessionkey zu bestimmen, potenzieren sie beide jeweils für sich das jeweils empfangene Ergebnis mit ihrer geheimen Zufallszahl modulo p. Das heißt:

   - Alice berechnet $S \equiv B^a \pmod{p}$, und
   - Bob berechnet $S \equiv A^b \pmod{p}$.

Auch wenn ein Spion g, p, und die Zwischenergebnisse A und B abhört, kann er den schließlich bestimmten Sessionkey, wegen der Schwierigkeit, den diskreten Logarithmus zu bestimmen, nicht berechnen.

Das ganze soll an einem Beispiel mit (unrealistisch) kleinen Zahlen gezeigt werden.

Beispiel in Zahlen:

1. Alice und Bob wählen $g = 11$, $p = 347$.
2. Alice wählt $a = 240$, Bob wählt $b = 39$ und behalten a und b geheim.
3. Alice berechnet $A \equiv g^a \equiv 11^{240} \equiv 49 \pmod{347}$.
   Bob berechnet $B \equiv g^b \equiv 11^{39} \equiv 285 \pmod{347}$.
4. Alice sendet Bob: $A = 49$,
   Bob sendet Alice: $B = 285$.
5. Alice berechnet $B^a \equiv 285^{240} \equiv 268 \pmod{347}$,
   Bob berechnet $A^b \equiv 49^{39} \equiv 268 \pmod{347}$.

Nun können Alice und Bob mit Hilfe ihres gemeinsamen Sessionkeys sicher kommunizieren. Auch wenn ein Spion alles, was über die Leitung ging, abhörte: $g = 11, p = 347, A = 49$ und $B = 285$, den geheimen Schlüssel kann er nicht berechnen.

Bemerkung:
In diesem Beispiel mit den kleinen Zahlen ist das Diskrete Logarithmus-Problem lösbar, aber mit großen Zahlen ist es kaum zu lösen[79,80].

Zu berechnen ist hier:
Von Alice: $11^x \equiv 49 \pmod{347}$, also $\log_{11}(49) \pmod{347}$.
Von Bob: $11^y \equiv 285 \pmod{347}$, also $\log_{11}(285) \pmod{347}$.

---

[79] Versucht man mit Mathematica, den diskreten Logarithmus x, der die Gleichung $11^x \equiv 49 \pmod{347}$ löst, mit Hilfe von Solve zu bestimmen, erhält man die *tdep*-Message "The equations appear to involve the variables to be solved for in an essentially non-algebraic way". Mathematica meint also, keine algebraisch direkten Verfahren zu kennen, um die Gleichung zu lösen. Doch mit der verallgemeinerten Funktion für die multiplikative Ordnung kann es Mathematica (hier für Alice): `MultiplicativeOrder[11, 347, 49]` liefert den Wert 67.
In Pari-GP lautet die Syntax: `znlog(Mod(49,347),Mod(11,347))`.
Auch mit anderen Tools wie dem LiDIA- oder BC-Paket (siehe Anhang Web-Links) können solche zahlentheoretischen Aufgaben gelöst werden. Die Funktion `dl` im Userinterface **LC** zu LiDIA liefert mit `dl(11,49,347)` ebenfalls den Wert 67.

[80] Warum haben die Funktionen bei dem DL-Problem für Alice den Wert 67 und nicht den Wert 240 geliefert? Der diskrete Logarithmus ist der kleinste natürliche Exponent, der die Gleichung $11^x \equiv 49 \pmod{347}$ löst. Sowohl $x = 67$ als auch $x = 240$ (die im Beispiel gewählte Zahl) erfüllen die Gleichung und können damit zur Berechnung des Sessionkeys benutzt werden: $285^{240} \equiv 285^{67} \equiv 268 \pmod{347}$. Hätten Alice und Bob als Basis g eine Primitivwurzel modulo p gewählt, dann gibt es für jeden Rest aus der Menge $\{1, 2, \cdots, p-1\}$ genau einen Exponenten aus der Menge $\{0, 1, \cdots, p-2\}$.
Info: Zum Modul 347 gibt es 172 verschiedene Primitivwurzeln, davon sind 32 prim (ist nicht notwendig). Da die im Beispiel für g gewählte Zahl 11 keine Primitivwurzel von 347 ist, nehmen die Reste nicht alle Werte aus der Menge $\{1, 2, \cdots, 346\}$ an. Somit kann es für einen bestimmten Rest mehr als einen oder auch gar keinen Exponenten aus der Menge $\{0, 1, \cdots, 345\}$ geben, der die Gleichung erfüllt.
`PrimeQ[347] = True; EulerPhi[347] = 346; GCD[11, 347] = 1; MultiplicativeOrder[11, 347] = 173`
In Pari-GP lautet die Syntax:
`isprime(347); eulerphi(347); gcd(11,347); znorder(Mod(11,347))`.

3.13 Das RSA-Verfahren mit konkreten Zahlen

Nachdem oben die Funktionsweise des RSA-Verfahrens beschrieben wurde, sollen diese Schritte hier mit konkreten, aber kleinen Zahlen durchgeführt werden.

3.13.1 RSA mit kleinen Primzahlen und mit einer Zahl als Nachricht

Bevor wir RSA auf einen Text anwenden, wollen wir es erst direkt mit einer Zahl zeigen[81].

1. Die gewählten Primzahlen seien $p = 5$ und $q = 11$.
   Also ist $n = 55$ und $J(n) = (p-1) * (q-1) = 40$.
2. $e = 7$ (sollte zwischen 11 und 40 liegen und muss teilerfremd zu 40 sein).
3. $d = 23$ (da $23 * 7 \equiv 161 \equiv 1 \pmod{40}$)

   $\rightarrow$ Public Key des Senders: $(55, 7)$,

   $\rightarrow$ Private Key des Empfängers: $(55, 23)$.
4. Nachricht sei nur die Zahl $M = 2$ (also ist kein Aufbrechen in Blöcke nötig).
5. Verschlüsseln: $C \equiv 2^7 \equiv 18 \pmod{55}$.
6. Chiffrat ist nur die Zahl $C = 18$ (also kein Aufbrechen in Blöcke nötig).
7. Entschlüsseln: $M \equiv 18^{23} \equiv 18^{(1+2+4+16)} \equiv 18 * 49 * 36 * 26 \equiv 2 \pmod{55}$.

Nun wollen wir RSA auf einen Text anwenden: zuerst mit dem Großbuchstabenalphabet (26 Zeichen), dann mit dem gesamten ASCII-Zeichensatz als Bausteine für die Nachrichten.

i	$11^i \bmod 347$	
0	1	
1	11	
2	121	
3	290	
67	49	gesuchter Exponent
172	284	
173	1	= Multiplikative Ordnung von 11 (mod 347)
174	11	
175	121	
176	290	
240	49	gesuchter Exponent

[81] Mit CrypTool v1.3 können Sie dies per **Einzelverfahren \ RSA-Demo \ RSA-Kryptosystem** lösen.

3.13.2 RSA mit etwas größeren Primzahlen und einem Text aus Großbuchstaben

Gegeben ist der Text „ATTACK AT DAWN" und die Zeichen werden auf folgende einfache Weise codiert[82]:

Tabelle 7: Großbuchstabenalphabet

Zeichen	Zahlenwert	Zeichen	Zahlenwert
Blank	0	M	13
A	1	N	14
B	2	O	15
C	3	P	16
D	4	Q	17
E	5	R	18
F	6	S	19
G	7	T	20
H	8	U	21
I	9	V	22
J	10	W	23
K	11	X	24
L	12	Y	25
		Z	26

Schlüsselerzeugung (Schritt 1 bis 3):
1. $p = 47, q = 79$ ($n = 3713$; $J(n) = (p-1) * (q-1) = 3.588$).
2. $e = 37$ (sollte zwischen 79 und 3588 liegen und muss teilerfremd zu 3588 sein).
3. $d = 97$ (denn $e * d = 1 \bmod J(n); 37 * 97 \equiv 3.589 \equiv 1 \pmod{3588}$)[83].

4. Verschlüsselung:

Text:	A	T	T	A	C	K		A	T		D	A	W	N
Zahl:	01	20	20	01	03	11	00	01	20	00	04	01	23	14

Aufteilung dieser 28-stelligen Zahl in 4-stellige Teile (denn 2626 ist noch kleiner als $n = 3713$), d.h. dass die Blocklänge 2 beträgt.
`0120 2001 0311 0001 2000 0401 2314`

Verschlüsselung aller 7 Teile jeweils per: $C \equiv M^{37} \pmod{3713}$[84]:
`1404 2932 3536 0001 3284 2280 2235`

5. Entschlüsselung:

[82] Mit CrypTool v1.3 können Sie dies per **Einzelverfahren \ RSA-Demo \ RSA-Kryptosystem** lösen. Dies ist auch im Tutorial/Szenario der Online-Hilfe zu CrypTool beschrieben [Optionen, Alphabet vorgeben, Basissystem, Blocklänge 2 und Dezimaldarstellung].
[83] Wie man $d = 97$ mit Hilfe des erweiterten ggT berechnet wird im Anhang A zu diesem Kapitel gezeigt.
[84] In Anhang D zu diesem Kapitel finden Sie den Beispiel-Quelltext zur RSA-Verschlüsselung mit Mathematica und Pari-GP.
Mit CrypTool v1.3 können Sie dies per **Einzelverfahren \ RSA-Demo \ RSA-Kryptosystem** lösen.

Chiffrat: `1404 2932 3536 0001 3284 2280 2235`

Aufteilung dieser 28-stelligen Zahl in 4-stellige Teile.
Entschlüsselung aller 7 Teile jeweils per: $M \equiv C^{97} \pmod{3.713}$:
`0120 2001 0311 0001 2000 0401 2314`

Umwandeln von 2-stelligen Zahlen in Großbuchstaben und Blanks.
Bei den gewählten Werten ist es für einen Kryptoanalytiker einfach, aus den öffentlichen Parametern $n = 3713$ und $e = 37$ die geheimen Werte zu finden, indem er offenlegt, dass $3713 = 47 * 79$.

Wenn n eine 768-Bit-Zahl ist, bestehen dafür – nach heutigen Kenntnissen – wenig Chancen.

3.13.3 RSA mit noch etwas größeren Primzahlen und mit einem Text aus ASCII-Zeichen

Real wird das ASCII-Alphabet benutzt, um die Einzelzeichen der Nachricht in 8-Bit lange Zahlen zu codieren.

Diese Aufgabe[85] ist angeregt durch das Beispiel aus [Eckert2003, S. 271].

Der Text „RSA works!" bedeutet in Dezimalschreibweise codiert:

Text:	R	S	A		w	o	r	k	s	!
Zahl:	82	83	65	32	119	111	114	107	115	33

Das Beispiel wird in 2 Varianten durchgespielt. Gemeinsam für beide sind die Schritte 1 bis 3.

Schlüsselerzeugung (Schritt 1 bis 3):
1. $p = 509, q = 503 \quad (n = 256.027; \quad J(n) = (p-1) * (q-1) = 255.016 = 2^3 * 127 * 251)$[86].
2. $e = 65.537$ (soll zwischen 509 und 255.016 liegen u. muss teilerfremd zu 255.016 sein)[87].
3. $d = 231.953$ (denn $e \equiv d^{-1} \bmod J(n)$; $65.537 * 231.953 \equiv 15.201.503.761 \equiv 1 \pmod{255.016}$)[88].

Variante 1:
Alle ASCII-Zeichen werden einzeln ver- und entschlüsselt (keine Blockbildung).

4. Verschlüsselung:

Text:	R	S	A		w	o	r	k	s	!
Zahl:	82	83	65	32	119	111	114	107	115	33

Keine Zusammenfassung der Buchstaben[89]!

---

[85] Mit CrypTool v1.3 können Sie dies per **Einzelverfahren \ RSA-Demo \ RSA-Kryptosystem** lösen.

[86] In Anhang D zu diesem Kapitel finden Sie den Quelltext zur Faktorisierung von $J(n)$ mit Mathematica und Pari-GP.
Mit CrypTool v1.3 können Sie dies per **Einzelverfahren \ RSA-Demo \ Faktorisieren einer Zahl** lösen.

[87] e soll also nicht $2, 127$ oder 251 sein ($65537 = 2^{16} + 1$).
Real wird $J(n)$ nicht faktorisiert, sondern für das gewählte e wird mit dem Euklidschen Algorithmus sichergestellt, dass $\mathrm{ggT}(d, J(n)) = 1$.

[88] Andere mögliche Kombinationen von (e,d) sind z.B.: (3, 170.011), (5, 204.013), (7, 36.431).

[89] Für sichere Verfahren braucht man große Zahlen, die möglichst alle Werte bis $n-1$ annehmen. Wenn die mögliche Wertemenge der Zahlen in der Nachricht zu klein ist, nutzen auch große Primzahlen nichts für die Sicherheit. Ein

Verschlüsselung pro Zeichen per: $C \equiv M^{65.537} \pmod{256.027}$[90]:

```
212984  025546  104529  031692  248407
100412  054196  100184  058179  227433
```

5. Entschlüsselung:
Chiffrat:

```
212984  025546  104529  031692  248407
100412  054196  100184  058179  227433
```

Entschlüsselung pro Zeichen per: $M \equiv C^{231.953} \bmod 256.027$:

```
82 83 65 32 119 111 114 107 115 33
```

Variante 2: Jeweils zwei ASCII-Zeichen werden als Block ver- und entschlüsselt.

Bei der Variante 2 wird die Blockbildung in zwei verschiedenen Untervarianten 4./5. und 4'./5'. dargestellt.

Text:	R	S	A		w	o	r	k	s	!
Zahl:	82	83	65	32	119	111	114	107	115	33

4. Verschlüsselung:
Blockbildung[91] (die ASCII-Zeichen werden als 8-stellige Binärzahlen hintereinander geschrieben):
`21075 16672 30575 29291 29473`[92]

Verschlüsselung pro Block per: $C \equiv M^{65.537} \pmod{256027}$[93]:

```
158721 137346 37358 240130 112898
```

5. Entschlüsselung:
Chiffrat:

```
158721 137346 37358 240130 112898
```

ASCII-Zeichen ist durch 8 Bits repräsentiert. Will man größere Werte, muss man mehrere Zeichen zusammenfassen. Zwei Zeichen benötigen 16 Bit, womit maximal der Wert 65.536 darstellbar ist; dann muss der Modul n größer sein als $2^{16} = 65.536$. Dies wird in Variante 2 angewandt. Beim Zusammenfassen bleiben in der Binär-Schreibweise die führenden Nullen erhalten (genauso wie wenn man oben in der Dezimalschreibweise alle Zahlen 3-stellig schreiben würde und dann die Folge `082 083, 065 032, 119 111, 114 107, 115 033` hätte).

[90] In Anhang D zu diesem Kapitel finden Sie den Quelltext zur RSA-Exponentiation mit Mathematica und Pari-GP.

[91]

```
                Binärdarstellung   Dezimaldarstellung
01010010, 82    01010010 01010011  =21075
01010011, 83
01000001, 65    01000001 00100000  =16672
00100000, 32
01110111, 119   01110111 01101111  =30575
01101111, 111
01110010, 114   01110010 01101011  =29291
01101011, 107
01110011, 115   01110011 00100001  =29473
00100001, 33:
```

[92] Mit CrypTool v1.3 können Sie dies per **Einzelverfahren \ RSA-Demo \ RSA-Kryptosystem** mit den folgenden Optionen lösen: alle 256 Zeichen, b-adisch, Blocklänge 2, dezimale Darstellung.

[93] In Anhang D zu diesem Kapitel finden Sie den Quelltext zur RSA-Exponentiation mit Mathematica und Pari-GP.

Entschlüsselung pro Block per: $M \equiv C^{231.953} \pmod{256.027}$:
`21075 16672 30575 29291 29473`

4‘. Verschlüsselung:

Blockbildung (die ASCII-Zeichen werden als 3-stellige Dezimalzahlen hintereinander geschrieben):
`82083 65032 119111 114107 115033`[94]

Verschlüsselung pro Block per: $C \equiv M^{65537} \pmod{256.027}$[95]:
`198967 051405 254571 115318 014251`

5’. Entschlüsselung:
Chiffrat:
`198967 051405 254571 115318 014251`

Entschlüsselung pro Block per: $M \equiv C^{231.953} \pmod{256.027}$:
`82083 65032 119111 114107 115033`

3.13.4 Eine kleine RSA-Cipher-Challenge (1)

Die Aufgabe stammt aus [Stinson1995, Exercise 4.6]: Die pure Lösung hat Prof. Stinson unter

`http://www.cacr.math.uwaterloo.ca/~dstinson/solns.html`[96]

veröffentlicht. Es geht aber nicht nur um das Ergebnis, sondern vor allem um die Einzelschritte der Lösung, also um die Darlegung der Kryptoanalyse[97].

Two samples of RSA ciphertext are presented in Tables 4.1 and 4.2. Your task is to decrypt them. The public parameters of the system are

$n = 18923$ and $e = 1261$ (for Table 4.1) and
$n = 31313$ and $e = 4913$ (for Table 4.2).

This can be accomplished as follows. First, factor n (which is easy because it is so small). Then compute the exponent d from $J(n)$, and, finally, decrypt the ciphertext. Use the square-and-multiply algorithm to exponentiate modulo n.

In order to translate the plaintext back into ordinary English text, you need to know how alphabetic characters are "encoded" as elements in $\mathbb{Z}_n$. Each element of $\mathbb{Z}_n$ represents three alphabetic characters as in the following examples:

`DOG` $\mapsto$ $3 * 26^2 + 14 * 26 + 6 = 2398$
`CAT` $\mapsto$ $2 * 26^2 + 0 * 26 + 19 = 1371$
`ZZZ` $\mapsto$ $25 * 26^2 + 25 * 26 + 25 = 17575$.

You will have to invert this process as the final step in your program.

[94] Die RSA-Verschlüsselung mit dem Modul $n = 256.027$ ist bei dieser Einstellung korrekt, da die ASCII-Blöcke in Zahlen kleiner oder gleich 255.255 kodiert werden.
[95] In Anhang D zu diesem Kapitel finden Sie den Quelltext zur RSA-Exponentiation mit Mathematica und Pari-GP.
[96] oder `http://bibd.unl/~stinson/solns.html`.
[97] Im Szenario der Online-Hilfe zu CrypTool und in der Präsentation auf der Web-Seite wird der Lösungsweg skizziert. Wenn uns jemand einen gut aufbereiteten konkreten Lösungsweg schickt, nehmen wir ihn gerne in die Dokumentation auf.

The first plaintext was taken from "The Diary of Samuel Marchbanks", by Robertson Davies, 1947, and the second was taken from "Lake Wobegon Days", by Garrison Keillor, 1985.

TABLE 4.1[98]: RSA-Ciphertext

12423	11524	7243	7459	14303	6127	10964	16399
9792	13629	14407	18817	18830	13556	3159	16647
5300	13951	81	8986	8007	13167	10022	17213
2264	961	17459	4101	2999	14569	17183	15827
12693	9553	18194	3830	2664	13998	12501	18873
12161	13071	16900	7233	8270	17086	9792	14266
13236	5300	13951	8850	12129	6091	18110	3332
15061	12347	7817	7946	11675	13924	13892	18031
2620	6276	8500	201	8850	11178	16477	10161
3533	13842	7537	12259	18110	44	2364	15570
3460	9886	8687	4481	11231	7547	11383	17910
12867	13203	5102	4742	5053	15407	2976	9330
12192	56	2471	15334	841	13995	17592	13297
2430	9741	11675	424	6686	738	13874	8168
7913	6246	14301	1144	9056	15967	7328	13203
796	195	9872	16979	15404	14130	9105	2001
9792	14251	1498	11296	1105	4502	16979	1105
56	4118	11302	5988	3363	15827	6928	4191
4277	10617	874	13211	11821	3090	18110	44
2364	15570	3460	9886	9988	3798	1158	9872
16979	15404	6127	9872	3652	14838	7437	2540
1367	2512	14407	5053	1521	297	10935	17137
2186	9433	13293	7555	13618	13000	6490	5310
18676	4782	11374	446	4165	11634	3846	14611
2364	6789	11634	4493	4063	4576	17955	7965
11748	14616	11453	17666	925	56	4118	18031
9522	14838	7437	3880	11476	8305	5102	2999
18628	14326	9175	9061	650	18110	8720	15404
2951	722	15334	841	15610	2443	11056	2186

[98]Die Zahlen dieser Tabelle können Sie mit Copy und Paste weiter bearbeiten.

TABLE 4.2[99]: RSA-Ciphertext

6340	8309	14010	8936	27358	25023	16481	25809
23614	7135	24996	30590	27570	26486	30388	9395
27584	14999	4517	12146	29421	26439	1606	17881
25774	7647	23901	7372	25774	18436	12056	13547
7908	8635	2149	1908	22076	7372	8686	1304
4082	11803	5314	107	7359	22470	7372	22827
15698	30317	4685	14696	30388	8671	29956	15705
1417	26905	25809	28347	26277	7897	20240	21519
12437	1108	27106	18743	24144	10685	25234	30155
23005	8267	9917	7994	9694	2149	10042	27705
15930	29748	8635	23645	11738	24591	20240	27212
27486	9741	2149	29329	2149	5501	14015	30155
18154	22319	27705	20321	23254	13624	3249	5443
2149	16975	16087	14600	27705	19386	7325	26277
19554	23614	7553	4734	8091	23973	14015	107
3183	17347	25234	4595	21498	6360	19837	8463
6000	31280	29413	2066	369	23204	8425	7792
25973	4477	30989					

3.13.5 Eine kleine RSA-Cipher-Challenge (2)

Die folgende Aufgabe ist eine korrigierte Variante aus dem sehr guten Buch von Prof. Yan [Yan2000, Example 3.3.7, S. 318]. Es geht aber nicht nur um das Ergebnis, sondern vor allem um die Einzelschritte der Lösung, also um die Darlegung der Kryptoanalyse[100].

Man kann sich drei völlig unterschiedlich schwierige Aufgaben vorstellen: Gegeben ist jeweils der Ciphertext und der öffentliche Schlüssel (e, n):

- Known-Plaintext: finde den geheimen Schlüssel d unter Benutzung der zusätzlich bekannten Ursprungsnachricht.
- Ciphertext-only: finde d und den Cleartext.
- RSA-Modul knacken, d.h. faktorisieren (ohne Kenntnis der Messages).

[99] Die Zahlen dieser Tabelle befinden sich auch in der Online-Hilfe „Szenario für die RSA-Demonstration“ der CrypTool-Auslieferung .

[100] Im Szenario der Online-Hilfe zu CrypTool und in der CrypTool-Präsentation werden der Lösungsweg skizziert. Wenn uns jemand einen gut aufbereiteten konkreten Lösungsweg schickt, nehmen wir ihn gerne in die Dokumentation auf.

$n = 639784868795271438588314150 41,\ e = 17579$

Message[101]:

```
1401202118011200,
1421130205181900,
0118050013010405,
0002250007150400
```

Cipher:

```
45411667895024938209259253423,
16597091621432020076311552201,
46468979279750354732637631044,
32870167545903741339819671379
```

Bemerkung:
Die ursprüngliche Nachricht bestand aus einem Satz mit 31 Zeichen (codiert mit dem Großbuchstabenalphabet aus Abschnitt 3.12.2). Dann wurden je 16 Dezimalziffern zu einer Zahl zusammengefasst (die letzte Zahl wurde mit Nullen aufgefüllt). Diese Zahlen wurden mit e potenziert.

Beim Entschlüsseln ist darauf zu achten, dass die berechneten Zahlen vorne mit Nullen aufzufüllen sind, um den Klartext zu erhalten.

Wir betonen das, weil in der Implementierung und Standardisierung die Art des Padding sehr wichtig ist für interoperable Algorithmen.

[101] Die Zahlen dieser Tabelle befinden sich auch in der Online-Hilfe „Szenario für die RSA-Demonstration“ der CrypTool-Auslieferung.

Literatur

[Agrawal2002] M. Agrawal, N. Kayal, N. Saxena,
PRIMES in P, August 2002
http://www.cse.iitk.ac.in/news/primality.html.

[Bartholome1996] A. Bartholomé, J. Rung, H. Kern,
Zahlentheorie für Einsteiger, Vieweg 1995, 2. Auflage 1996.

[Bauer1995] Friedrich L. Bauer,
Entzifferte Geheimnisse, Springer, 1995.

[Bauer2000] Friedrich L. Bauer,
Decrypted Secrets, Springer 1997, 2nd edition 2000.

[Bernstein2001] D. J. Bernstein,
Circuits for integer factorization: a proposal,
http://cr.yp.to/papers/nfscircuit.ps
http://cr.yp.to/djb.html.

[Beutelspacher1996] Albrecht Beutelspacher,
Kryptologie, Vieweg 1987, 5. Auflage 1996.

[Bourseau2002] F. Bourseau, D. Fox, C. Thiel,
Vorzüge und Grenzen des RSA-Verfahrens,
in Datenschutz und Datensicherheit (DuD) 26/2002, S. 84-89.

[Brands2002] Gilbert Brands,
Verschlüsselungsalgorithmen – Angewandte Zahlentheorie rund um Sicherheitsprotokolle,
Vieweg, 2002.

[BSI2002] BSI (Bundesamt für Sicherheit in der Informationstechnik)
Empfehlungen zur Wahl der Schlüssellängen, Bonn, 9.9.2002,
(Geeignete Kryptoalgorithmen zur Erfüllung der Anforderungen nach 17 (1) SigG vom 22. Mai 2001 in Verbindung mit Anlage 1, I 2, SigV vom 22. November 2001),
http://www.bsi.bund.de/esig/basics/techbas/krypto/bund02v7.pdf
Eine Stellungnahme zu diesen Empfehlungen:
http://www.secorvo.de/publikat/
stellungnahme-algorithmenempfehlung-020307.pdf

[Buchmann1999] Johannes Buchmann,
Einführung in die Kryptographie, Springer, 1999.

[Buhler1993] J.P. Buhler, H.W. Lenstra, C. Pomerance,
Factoring integers with the number field sieve,
In: A.K. Lenstra, H.W. Lenstra (Hrsg.): The Development of the Number Field Sieve, Lecture Notes in Mathematics, Vol. 1554, Springer, Heidelberg 1993, S. 50−94.

[Eckert2003] Claudia Eckert,
IT-Sicherheit: Konzepte-Verfahren-Protokolle, Oldenbourg 2001, 2. Auflage 2003.

[Ertel2001] Wolfgang Ertel,
Angewandte Kryptographie, Fachbuchverlag Leipzig FV 2001.

[Graham1994] Graham, Knuth, Patashnik,
Concrete Mathematics, a Foundation of Computer Science,
Addison Wesley 1989, 6th printing 1990.

[Kippenhahn1997] Rudolph Kippenhahn,
Verschlüsselte Botschaften – Geheimschrift, Enigma und Chipkarte, Rowohlt, 1997.

[Kippenhahn1999] Rudolph Kippenhahn,
Code Breaking – A History and Exploration, Constable, 1999.

[Knuth1998] Donald E. Knuth,
The Art of Computer Programming, vol 2: Seminumerical Algorithms,
Addison-Wesley, 2nd edition 1998.

[Lenstra1993] A. Lenstra, H. Lenstra:
The development of the Number Field Sieve,
Lecture Notes in Mathematics 1554, Springer, New York 1993

[Lenstra2002] Arjen K. Lenstra, Adi Shamir, Jim Tomlinson, Eran Tromer,
Analysis of Bernsteins Factorization Circuit,
http://www.cryptosavvy.com/mesh.pdf

[Menezes2001] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone
Handbook of Applied Cryptography, CRC Press 1997, 5th printing 2001.

[Pfleeger1997] Charles P. Pfleeger,
Security in Computing, Prentice-Hall, 2nd edition 1997.

[Pomerance1984] C. Pomerance,
The quadratic sieve factoring algorithm,
In: G.R. Blakley, D. Chaum (Hrsg.): Proceedings of Crypto '84, LNCS 196, Springer Berlin 1995, S. 169-182.

[RSA Security 2002] RSA Security,
Has the RSA algorithm been compromised as a result of Bernstein's Paper?,
8. April 2002,
http://www.rsasecurity.com/.

[Schneier1996] Bruce Schneier,
Applied Cryptography, Protocols, Algorithms, and Source Code in C,
Wiley and Sons 1994, 2nd edition 1996.

[Schwenk2002] Jörg Schwenk,
Sicherheit und Kryptographie im Internet, Vieweg 2002.

[Sedgewick1990] Robert Sedgewick,
Algorithms in C, Addison-Wesley, 1990.

[Shamir2003] Adi Shamir, Eran Tromer,
Factoring Large Numbers with the TWIRL Device, Januar 2003,
http://www.wisdom.weizmann.ac.il/~tromer/.

[Silverman2000] Robert D. Silverman:
A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths
In: RSA Laboratories Bulletin, No. 13, April 2000, S. 1-22

[Stinson1995] Douglas R. Stinson,
Cryptography - Theory and Practice, CRC Press, 1995.

[Welschenbach2001] Welschenbach, Michael,
Kryptographie in C und C++, Springer 2001.

[Wiles1994] Wiles, Andrew,
Modular elliptic curves and Fermat's Last Theorem,
in Annals of Mathematics 141 (1995).

[Wolfenstetter1998] Albrecht Beutelspacher, Jörg Schwenk, Klaus-Dieter Wolfenstetter,
Moderne Verfahren in der Kryptographie, Vieweg 1995, 2. Auflage 1998.

[Yan2000] Song Y. Yan,
Number Theory for Computing, Springer, 2000.

Web-Links

1. Fibonacci-Seite von Ron Knott,
   Hier dreht sich alles um Fibonacci-Zahlen.
   http://www.mcs.surrey.ac.uk/personal/R.Knott/Fibonacci/fib.html

2. CrypTool (Version 1.3.04, 2003),
   Freeware zur Veranschaulichung von Kryptographie,
   http://www.cryptool.de,
   http://www.cryptool.org,
   http://www.cryptool.com

3. Mathematica,
   Kommerzielles Mathematik-Paket
   http://www.wolfram.com

4. LiDIA,
   Umfangreiche Bibliothek mit zahlentheoretischen Funktionen und dem Interpreter LC
   http://www.informatik.tu-darmstadt.de/TI/LiDIA

5. BC,
   Interpreter mit zahlentheoretischen Funktionen
   http://www.maths.uq.edu.au/~krm/gnubc.html

6. Pari-GP,
   Hervorragender, schneller und freier Interpreter mit zahlentheoretischen Funktionen
   http://www.parigp-home.de und http://www.parigp-home.com

7. Erst nach Vollendung dieses Artikels wurde mir die Web-Seite von Herrn Münchenbach bekannt, die interaktiv und didaktisch sehr ausgereift die grundlegenden Denkweisen der Mathematik anhand der elementaren Zahlentheorie nahebringt. Sie entstand für ein Unterrichtsprojekt der 11. Klasse des Technischen Gymnasiums (sie ist leider nur in Deutsch verfügbar):
   http://www.hydrargyrum.de/kryptographie

8. Ebenfalls erst nach Vollendung dieses Artikels wurde mir die Seite des Beauftragten für die Lehrplanentwicklung des Fachs Informatik an der gymnasialen Oberstufe des Landes Saarland bekannt. Hier befindet sich eine Sammlung von Texten und Programmen (in Java), die aus didaktischen Überlegungen entstand (alles leider nur in deutsch verfügbar).
   http://www.hom.saar.de/~awa/kryptolo.htm

9. BSI,
   Bundesamt für Sicherheit in der Informationstechnik
   http://www.bsi.bund.de

10. Faktorisierungsrekorde und Challenges ,
    http://www.crypto-world.com/

http://www.uni-bonn.de/Aktuelles/Pressemitteilungen/pm02/pm035-02.html
http://www.ercim.org/publication/Ercim_News/enw49/franke.html, 2002-01
http://www.rsasecurity.com/rsalabs/challenges/factoring/numbers.html

11. Das Cunningham-Projekt,
http://www.cerias.purdue.edu/homes/ssw/cun/

Dank

Ich möchte hier die Gelegenheit nutzen, den folgenden Personen ganz herzlich zu danken:

- Hr. Henrik Koy für das anregende und sehr konstruktive Korrekturlesen, für die vielen Verbesserungen dieses Artikels und für die Hilfen bei TeX, ohne die dieser Artikel nie in dieser Form erschienen wäre.
  Außerdem hat Hr. Koy die komplexe Dialogbox samt Verarbeitung zum RSA-Kryptosystem erstellt, mit der die RSA-Beispiele dieses Artikels nachvollzogen werden können.
- Jörg-Cornelius Schneider für die Unterstützung bei TeX und die mannigfaltigen Hilfen bei allen Arten von Programmier- und Design-Problemen.
- Dr. Illies für den Hinweis auf Pari-GP .

Anhang A: Der größte gemeinsame Teiler (ggT) von ganzen Zahlen und die beiden Algorithmen von Euklid

1. Der größte gemeinsame Teiler zweier natürlicher Zahlen a und b ist eine wichtige und sehr schnell zu berechnende Größe. Wenn eine Zahl c die Zahlen a und b teilt (d.h. es gibt ein a' und ein b', so dass $a = a' * c$ und $b = b' * c$), dann teilt c auch den Rest r der Division von a durch b. Wir schreiben in aller Kürze: Aus c teilt a und b folgt: c teilt $r = a - \lfloor a/b \rfloor * b$[102].

   Weil die obige Aussage für alle gemeinsamen Teiler c von a und b gilt, folgt für den größten gemeinsamen Teiler von a und b ($\mathrm{ggT}(a, b)$) die Aussage

   $$\mathrm{ggT}(a, b) = \mathrm{ggT}(a - \lfloor a/b \rfloor * b, b).$$

   Mit dieser Erkenntnis lässt sich der Algorithmus zum Berechnen des ggT zweier Zahlen wie folgt (in Pseudocode) beschreiben:

```
INPUT: a,b != 0
1. if ( a < b ) then  x = a; a = b; b = x; // Vertausche a und b (a > b)
2. a = a - int(a/b) * b        // a wird kleiner b, der ggT(a, b)
                               // bleibt unveraendert
3. if ( a != 0 ) then goto 1.  // nach jedem Schritt faellt a, und
                               // der Algorithmus endet, wenn a == 0.
OUTPUT "ggT(a,b) = " b         // b ist der ggT vom urspr"unglichen a und b
```

2. Aus dem ggT lassen sich aber noch weitere Zusammenhänge bestimmen: Dazu betrachtet man für a und b das Gleichungssystem:

   $$\begin{aligned} a &= 1 * a + 0 * b \\ b &= 0 * a + 1 * b, \end{aligned}$$

   bzw. in Matrix-Schreibweise

   $$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} a \\ b \end{pmatrix}.$$

   Wir fassen diese Informationen in der erweiterten Matrix

   $$\left(\begin{array}{c|cc} a & 1 & 0 \\ b & 0 & 1 \end{array}\right)$$

   zusammen. Wendet man den ggT-Algorithmus auf diese Matrix an, so erhält man den *erweiterten ggT Algorithmus*, mit dem die multiplikative Inverse bestimmt wird.

[102] Die Gaussklammer $\lfloor x \rfloor$ der reellwertigen Zahl x ist definiert als: $\lfloor x \rfloor$ ist die größte ganze Zahl kleiner oder gleich x.

`INPUT:` $a, b \neq 0$

`0.` $x_{1,1} := 1, x_{1,2} := 0, x_{2,1} := 0, x_{2,2} := 1.$

`1.` $\left(\begin{array}{c|cc} a & x_{1,1} & x_{1,2} \\ b & x_{2,1} & x_{2,2} \end{array}\right) := \begin{pmatrix} 0 & 1 \\ 1 & -\lfloor a/b \rfloor * b \end{pmatrix} * \left(\begin{array}{c|cc} a & x_{1,1} & x_{1,2} \\ b & x_{2,1} & x_{2,2} \end{array}\right).$

`2. if (b != 0) then goto 1.`

`OUTPUT:` "ggT$(a, b) = a * x + b * y$: ", "ggT$(a, b) =$ " b, "$x =$" $x_{2,1}$, "$y =$" $x_{2,2}$.

Da dieser Algorithmus nur lineare Transformationen durchführt, gelten immer die Gleichungen

$$\begin{aligned} a &= x_{1,1} * a + x_{1,2} * b, \\ b &= x_{2,1} * a + x_{2,2} * b, \end{aligned}$$

und es folgt am Ende des Algorithmus[103] die erweiterte ggT-Gleichung:

$$\text{ggT}(a, b) = a * x_{2,1} + b * x_{2,2}.$$

Beispiel:
Mit dem erweiterten ggT lässt sich für $e = 37$ die modulo 3588 multiplikativ inverse Zahl d bestimmen (d.h $37 * d \equiv 1 \pmod{3588}$):

`0.` $\left(\begin{array}{c|cc} 3588 & 1 & 0 \\ 37 & 0 & 1 \end{array}\right)$

`1.` $\left(\begin{array}{c|cc} 37 & 1 & 0 \\ 36 & 0 & -96 \end{array}\right) = \begin{pmatrix} 0 & 1 \\ 1 & -(\lfloor 3588/36 \rfloor = 96) * 37 \end{pmatrix} * \left(\begin{array}{c|cc} 3588 & 1 & 0 \\ 37 & 0 & 1 \end{array}\right).$

`2.` $\left(\begin{array}{c|cc} 36 & 1 & -96 \\ 1 & -1 & 97 \end{array}\right) = \begin{pmatrix} 0 & 1 \\ 1 & -(\lfloor 37/36 \rfloor = 1) * 36 \end{pmatrix} * \left(\begin{array}{c|cc} 37 & 1 & 0 \\ 36 & 0 & -96 \end{array}\right).$

`3.` $\left(\begin{array}{c|cc} \mathbf{1} & \mathbf{-1} & \mathbf{97} \\ 0 & 37 & -3588 \end{array}\right) = \begin{pmatrix} 0 & 1 \\ 1 & -(\lfloor 36/1 \rfloor = 36) * 1 \end{pmatrix} * \left(\begin{array}{c|cc} 36 & 1 & -96 \\ 1 & -1 & 97 \end{array}\right).$

`OUTPUT:`
ggT$(37, 3588) = a * x + b * y$: ggT$(37, 3588) = 1$, $x = -1$, $y = 97$.

Es folgt

1. 37 und 3588 sind teilerfremd (37 ist invertierbar modulo 3588).
2. $37 * 97 = (1 * 3588) + 1$ mit anderen Worten $37 * 97 \equiv 1 \pmod{3588}$
   und damit ist modulo 3588 die Zahl 97 multiplikativ invers zu 37.

[103] Wenn der ggT-Algorithmus endet, steht in den Programmvariablen a und b: $a = 0$ und $b = ggT(a, b)$. Bitte beachten Sie, dass die Programmvariablen zu den Zahlen a und b verschieden sind und ihre Gültigkeit nur im Rahmen des Algorithmus haben.

Anhang B: Abschlussbildung

Die Eigenschaft der Abgeschlossenheit wird innerhalb einer Menge immer bezüglich einer Operation definiert. Im folgenden wird gezeigt, wie man für eine gegebene Ausgangsmenge G_0 die abgeschlossene Menge G bezüglich der Operation + (mod 8) konstruiert („Abschlussbildung"):

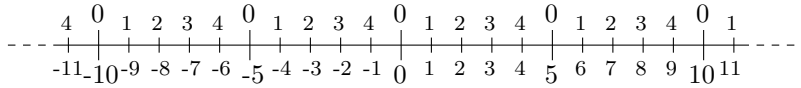
$$
\begin{aligned}
G_0 &= \{2,3\} \text{ die Addition der Zahlen in } G_0 \text{ bestimmt weitere Zahlen :} \\
&\quad 2+3 \equiv 5 \pmod 8 = 5 \\
&\quad 2+2 \equiv 4 \pmod 8 = 4 \\
&\quad 3+3 \equiv 6 \pmod 8 = 6 \\
G_1 &= \{2,3,4,5,6\} \text{ die Addition der Zahlen in } G_1 \text{ bestimmt :} \\
&\quad 3+4 \equiv 7 \pmod 8 = 7 \\
&\quad 3+5 \equiv 8 \pmod 8 = 0 \\
&\quad 3+6 \equiv 9 \pmod 8 = 1 \\
G_2 &= \{0,1,2,3,4,5,6,7\} \text{ die Addition der Zahlen in } G_2 \textit{ erweitert die Menge nicht}! \\
G_3 &= G_2 \text{ man sagt : } G_2 \text{ ist abgeschlossen bezüglich der Addition } \pmod 8.
\end{aligned}
$$

Ende der Abschlussbildung.

Anhang C: Bemerkungen zur modulo Subtraktion

Beispielweise gilt für die Subtraktion modulo 5: $2 - 4 \equiv -2 \equiv 3 \bmod 2$. Es gilt modulo 5 also nicht, dass $-2 = 2$! Dies gleichzusetzen ist ein häufig gemachter Fehler. Dies kann man sich gut verdeutlichen, wenn man die Permutation $(0, 1, 2, 3, 4)$ aus $\mathbb{Z}_5$ von z.B. -11 bis $+11$ wiederholt über den Zahlenstrahl aus $\mathbb{Z}$ legt.

Zahlengerade modulo 5



Zahlengerade der ganzen Zahlen

Anhang D: Beispiele mit Mathematica und Pari-GP

In diesem Anhang finden Sie den Quellcode, um die Tabellen und Beispiele mit Hilfe von Mathematica und dem freien Programm Pari-GP zu berechnen.

Multiplikationstabellen modulo m

Die auf Seite 56 aufgeführten Multiplikationstabellen modulo $m = 17$ für $a = 5$ und $a = 6$ werden mit Mathematica durch die folgenden Befehle bestimmt:

```
m = 17; iBreite = 18; iFaktor1 = 5; iFaktor2 = 6;
Print[ ''i '', Table[ i, {i, 1, iBreite} ] ];
Print[ iFaktor1, ''*i '', Table[ iFaktor1*i, {i, 1, iBreite } ] ];
Print[ ''Rest '', Table[ Mod[iFaktor1*i, m], {i, 1, iBreite } ] ];
Print[ iFaktor2, ''*i '', Table[ iFaktor2*i, {i, 1, iBreite } ] ];
Print[ ''Rest '', Table[ Mod[iFaktor2*i, m], {i, 1, iBreite } ] ];
```

Mit Pari-GP können Sie die Tabellenwerte folgendermaßen berechnen:

```
m=17; iBreite=18; iFaktor1=5; iFaktor2=6;
```

`matrix(1,iBreite, x,y, iFaktor1*y)` ergibt

```
[5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90]
```

`matrix(1,iBreite, x,y, (iFaktor1*y)%m )` ergibt

```
[5 10 15 3 8 13 1 6 11 16 4 9 14 2 7 12 0 5]
```

Beachte: Pari-GP erzeugt bei Verwendung von `Mod` das Objekt `Mod` und gibt es auch aus:

```
matrix(1,iBreite, x,y, Mod(iFaktor1*y, m))
[Mod(5, 17) Mod(10, 17) Mod(15, 17) Mod(3, 17) Mod(8, 17) Mod(13, 17) Mod(1, 17)
 Mod(6, 17) Mod(11, 17) Mod(16, 17) Mod(4, 17) Mod(9, 17) Mod(14, 17) Mod(2, 17)
 Mod(7, 17) Mod(12, 17) Mod(0, 17) Mod(5, 17)]
```

Aus dem `Mod`-Objekt kann man die einzelnen Komponenten entweder mit der `component`- oder mit der `lift`-Funktion zurückgewinnen:

```
component(Mod(5,17), 1) → 17
component(Mod(5,17), 2) → 5
component(Mod(17,5), 1) → 5
component(Mod(17,5), 2) → 2
lift(Mod(17,5)) → 2
```

Die weiteren Beispiele der Multiplikationstabelle modulo 13 und modulo 12 auf Seite 56 bestimmen Sie, indem Sie im Quelltext jeweils `m=17` durch den entsprechenden Zahlenwert (`m=13` bzw. `m=12`) ersetzen.

Schnelles Berechnen hoher Potenzen

Das schnelle Potenzieren modulo m gehört zu den Grundfunktionen von Mathematica und Pari-GP. Sie können mit Hilfe dieser Programme natürlich auch die Idee der Square-and-Multiply-Methode nachvollziehen. In Mathematica bestimmen Sie die Einzelexponentiation des Beispiels auf Seite 59 mit

```
Mod[{87^43, 87^2, 87^4, 87^8, 87^16, 87^32}, 103] = {85, 50, 28, 63, 55, 38}.
```

und in Pari-GP lautet die Syntax:

```
Mod([87^43,87^2,87^4,87^8,87^16,87^32],103)
```

Multiplikative Ordnung und Primitivwurzel

Die Ordnung $ord_m(a)$ einer Zahl a in der multiplikativen Gruppe Z_m^* ist die kleinste ganze Zahl $i \geq 1$ für die gilt $a^i \equiv 1 \bmod m$. Für das Beispiel auf Seite 67 können Sie sich mit Mathematica alle Exponentiationen $a^i \bmod 11$ durch die folgende Programmzeile ausgeben lassen.

```
m=11;  Table[ Mod[a^i, m], {a, 1, m-1}, {i, 1, m-1} ]
```

In Pari-GP lautet die Syntax:

```
m=11; matrix(10,10, x,y, (x^y)%m )
```

Im Beispiel auf Seite 68 wird die Ordnung $ord_{45}(a)$ sowie die Eulersche Zahl $J(45)$ ausgegeben. Mit Mathematica bestimmen Sie diese Tabelle durch die folgende Programmschleife (innerhalb von Print kann man keine Do-Schleife verwenden und jedes Print gibt am Ende ein Newline aus).

```
m = 45;
Do[ Print[ Table[ Mod[a^i, m], {i, 1, 12} ],
'', '', MultiplicativeOrder[a, m, 1],
'', '', EulerPhi[m] ],
{a, 1, 12} ];
```

In Pari-GP lautet die Syntax:

```
m=45;
matrix(12,14, x,y,
       if( y<=12, (x^y)%m,
       if( y==13, if( gcd(x,m)==1, znorder(Mod(x,m)), "--"),
       eulerphi(m))))
```

`znorder(Mod(x,m))` kann nur berechnet werden, wenn x teilerfremd zu m ist, was mit `gcd(x,m)` überprüft wird. Bessere Performance erreicht man mit `Mod(x,m)^y` statt mit `(x^y)%m`.

Auch in Pari-GP kann man Schleifen erzeugen. Verzichtet man auf die Tabellenform, sieht das so aus:

```
for( x=1,12,
     for(y=1,12, print(Mod(x^y,m)));
     if(gcd(x,m)==1, print(znorder(Mod(x,m))), print("--"));
     print(eulerphi(m)))
```

Im dritten Beispiel auf Seite 69 werden die Exponentiationen a^i mod 46 sowie die Ordnungen $ord_{46}(a)$ ausgegeben.

Mit Mathematica bestimmen Sie diese Tabelle durch die folgende Programmschleife.

```
m = 46;
Do[ Print[ Table[ Mod[a^i, m], {i, 1, 23} ],
'', '', MultiplicativeOrder[a, m, 1]
{a, 1, 23} ];
```

In Pari-GP lautet die Syntax:

```
m=46;
matrix(23,24, x,y,
       if( y<=23, (x^y)%m,
       if( y==24, if( gcd(x,m)==1, znorder(Mod(x,m)), "--"))))
```

RSA-Beispiele

In diesem Abschnitt sind die Quelltexte für die RSA-Beispiele im Abschnitt Das RSA-Verfahren mit konkreten Zahlen in Mathematica und Pari-GP angegeben.

Beispiel auf Seite 87.
Die RSA-Exponentiation M^{37} mod 3713 für die Nachricht $M = 120$ kann mit Mathematica per `PowerMod[120, 37, 3713]` berechnet werden.
In Pari-GP lautet die Syntax
`Mod(120,3713)^37 oder Mod(120^37,3713)`.

Beispiel auf Seite 88.
Die Faktorisierung von $J(256.027) = 255.016 = 2^3 * 127 * 251$ bestimmt Mathematica mit `FactorInteger[255016]= {{2,3}, {127,1}, {251,1}}`.
In Pari-GP lautet die Syntax:
`factor(255016)`.

Beispiel auf Seite 88.
Mit Mathematica kann die RSA-Verschlüsselung durch den Befehl
`PowerMod[{82, 83, 65, 32, 119, 111, 114, 107, 115, 33}, 65537, 256027]}`
berechnet werden.
In Pari-GP lautet die Syntax:

```
vecextract( [Mod(82,256027)^65537, Mod(83,256027)^65537, Mod(65,256027)^65537,
Mod(32,256027)^65537, Mod(119,256027)^65537, ...] )
```

Bemerkung zur Verwendung von `Mod` in Pari-GP:
`Mod(82,256027)^65537` ist wesentlich schneller als
– `Mod(82^65537, 256027)` oder
– `(82^65537) % 256027`.

Beispiel auf Seite 89.
Die RSA-Verschlüsselung kann mit Mathematica per
`PowerMod[{21075, 16672, 30575, 29291, 29473}, 65537, 256027]`
berechnet werden.
In Pari-GP lautet die Syntax:

```
vecextract( [Mod(21075,256027)^65537, Mod(16672,256027)^65537,
Mod(30575,256027)^65537, Mod(29291,256027)^65537,
Mod(29473,256027)^65537], 31)
```

Beispiel auf Seite 90.
Die RSA-Verschlüsselung kann mit Mathematica per
`PowerMod[{82083, 65032, 119111, 114107, 115033}, 65537, 256027]`
berechnet werden.
In Pari-GP lautet die Syntax:

```
vecextract( [Mod(82083,256027)^65537, Mod(65032,256027)^65537,
Mod(119111,256027)^65537, Mod(114107,256027)^65537,
Mod(115033,256027)^65537], 31)
```

Anhang E: Liste der hier formulierten Definitionen und Sätze

4 Die mathematischen Ideen hinter der modernen Kryptographie

(Oyono R./ Esslinger B., Sep. 2000, Updates Nov. 2000, Feb. 2003)

4.1 Einwegfunktionen mit Falltür und Komplexitätsklassen

Eine **Einwegfunktion** ist eine effizient zu berechnende (injektive) Funktion, deren Umkehrung jedoch nur mit extrem hohem Rechenaufwand – jedoch praktisch unmöglich – zu berechnen ist.

Etwas genauer formuliert: Eine Einwegfunktion ist eine Abbildung f einer Menge X in eine Menge Y, so dass $f(x)$ für jedes Element x von X leicht zu berechnen ist, während es für (fast) jedes y aus Y praktisch unmöglich ist, ein Urbild x (d.h. ein x mit $f(x) = y$) zu finden.

Ein alltägliches Beispiel für eine Einwegfunktion ist ein Telefonbuch: die auszuführende Funktion ist die, einem Namen die entsprechende Telefonnummer zuzuordnen. Da die Namen alphabetisch geordnet sind, ist diese Zuordnung einfach auszuführen. Aber ihre Invertierung, also die Zuordnung eines Namens zu einer gegebenen Nummer, ist offensichtlich schwierig, wenn man nur ein Telefonbuch zur Verfügung hat.

Einwegfunktionen spielen in der Kryptographie eine entscheidende Rolle. Fast alle kryptographischen Begriffe kann man durch Verwendung des Begriffs Einwegfunktion umformulieren. Als Beispiel betrachten wir die Public Key-Verschlüsselung (asymmetrische Kryptographie):

Jedem Teilnehmer T des Systems wird ein privater Schlüssel d_T und ein sogenannter öffentlicher Schlüssel e_T zugeordnet. Dabei muss die folgende Eigenschaft (Public Key-Eigenschaft) gelten: Für einen Gegner, der den öffentlichen Schlüssel e_T kennt, ist es praktisch unmöglich, den privaten Schlüssel d_T zu bestimmen.

Zur Konstruktion nützlicher Public Key-Verfahren sucht man also eine Einwegfunktion, die in einer Richtung „einfach“ zu berechnen, die in der anderen Richtung jedoch „schwer“ (praktisch unmöglich) zu berechnen ist, solange eine bestimmte zusätzliche Information (Falltür) nicht zur Verfügung steht. Mit der zusätzlichen Information kann die Umkehrung effizient gelöst werden. Solche Funktionen nennt man **Einwegfunktionen mit Falltür** (trapdoor one-way function). Im obigen Fall ist d_T die Falltür-Information.

Dabei bezeichnet man ein Problem als „einfach“, wenn es in polynomialer Zeit als Funktion der Länge der Eingabe lösbar ist, d.h. wenn es so gelöst werden kann, dass der Zeitaufwand sich als polynomiale Funktion in Abhängigkeit der Länge der Eingabe darstellen lässt. Wenn die Länge der Eingabe n Bits beträgt, so ist die Zeit der Berechnung der Funktion proportional zu n^a, wobei a eine Konstante ist. Man sagt, dass die Komplexität solcher Probleme $O(n^a)$ beträgt (Landau- oder Big-O-Notation).

Vergleicht man 2 Funktionen 2^n und n^a, wobei a eine Konstante ist, dann gibt es immer einen Wert für n, ab dem für alle weiteren n gilt: $n^a < 2^n$. Die Funktion n^a hat eine geringere Komplexität. Z.B. für $a = 5$ gilt: ab der Länge $n = 23$ ist $2^n > n^5$ und danach wächst 2^n auch deutlich schneller [($2^{22} = 4.194.304$, $22^5 = 5.153.632$), ($2^{23} = 8.388.608$, $23^5 = 6.436.343$), ($2^{24} = 16.777.216$, $24^5 = 7.962.624$)].

Der Begriff „praktisch unmöglich“ ist etwas schwammiger. Allgemein kann man sagen, ein Problem ist nicht effizient lösbar, wenn der zu seiner Lösung benötigte Aufwand schneller wächst als die polynomiale Zeit als Funktion der Größe der Eingabe. Wenn beispielsweise die Länge der Eingabe n Bits beträgt und die Zeit zur Berechnung der Funktion proportional zu 2^n ist, so gilt gegenwärtig: die Funktion ist für $n > 80$ praktisch nicht zu berechnen.

Die Entwicklung eines praktisch einsetzbaren Public Key-Verfahrens hängt daher von der Entdeckung einer geeigneten Einwegfunktion mit Falltür ab.

Um Ordnung in die verwirrende Vielfalt von möglichen Problemen und ihre Komplexitäten zu bringen, fasst man Probleme mit ähnlicher Komplexität zu Klassen zusammen.

Die wichtigsten Komplexitätsklassen sind die Klassen **P** und **NP**:

- Die Klasse **P**: Zu dieser Klasse gehören diejenigen Probleme, die mit polynomialem Zeitaufwand lösbar sind.
- Die Klasse **NP**: Bei der Definition dieser Klasse betrachten wir nicht den Aufwand zur Lösung eines Problems, sondern den Aufwand zur Verifizierung einer gegebenen Lösung. Die Klasse **NP** besteht aus denjenigen Problemen, bei denen die Verifizierung einer gegebenen Lösung mit polynomialem Zeitaufwand möglich ist. Dabei bedeutet der Begriff **NP** „nichtdeterministisch“ polynomial und bezieht sich auf ein Berechnungsmodell, d.h. auf einen nur in der Theorie existierenden Computer, der richtige Lösungen nichtdeterministisch „raten“ und dies dann in polynomialer Zeit verifizieren kann.

Die Klasse **P** ist in der Klasse **NP** enthalten. Ein berühmtes offenes Problem ist die Frage, ob $\mathbf{P} \neq \mathbf{NP}$ gilt oder nicht, d.h. ob **P** eine echte Teilmenge ist oder nicht. Eine wichtige Eigenschaft der Klasse **NP** ist, dass sie auch sogenannte „**NP**-vollständige“ Probleme enthält. Dies sind Probleme, welche die Klasse **NP** im folgenden Sinne vollständig repräsentieren: Wenn es einen „guten“ Algorithmus für ein solches Problem gibt, dann existieren für alle Probleme aus **NP** „gute“ Algorithmen. Insbesondere gilt: wenn auch nur ein vollständiges Problem in **P** läge, d.h. wenn es einen polynomialen Lösungsalgorithmus für dieses Problem gäbe, so wäre **P**=**NP**. In diesem Sinn sind die **NP**-vollständigen Probleme die schwierigsten Probleme in **NP**.

Viele kryptographische Protokolle sind so gemacht, dass die „guten“ Teilnehmer nur Probleme aus **P** lösen müssen, während sich ein Angreifer vor Probleme aus **NP** gestellt sieht.

Man weiß leider bis heute nicht, ob es Einwegfunktionen überhaupt gibt. Man kann aber zeigen, dass Einwegfunktionen genau dann existieren, wenn $\mathbf{P} \neq \mathbf{NP}$ gilt [Balcazar1988, S.63].

Es gab immer wieder die Aussage, jemand habe die Äquivalenz bewiesen, z.B.

http://www.geocities.com/st_busygin/clipat.html,

aber bisher erwiesen sich diese Aussagen stets als falsch.

Es wurden eine Reihe von Algorithmen für Public Key-Verfahren vorgeschlagen. Einige davon erwiesen sich, obwohl sie zunächst vielversprechend erschienen, als polynomial lösbar. Der berühmteste durchgefallene Bewerber ist der Knapsack mit Falltür, der von Ralph Merkle [Merkle1978] vorgeschlagen wurde.

4.2 Knapsackproblem als Basis für Public Key-Verfahren

4.2.1 Knapsackproblem

Gegeben n Gegenstände $G_1, \ldots, G_n$ mit den Gewichten $g_1, \ldots g_n$ und den Werten $w_1, \cdots, w_n$. Man soll wertmäßig so viel wie möglich unter Beachtung einer oberen Gewichtsschranke g davontragen. Gesucht ist also eine Teilmenge von $\{G_1, \cdots, G_n\}$, etwa $\{G_{i_1}, \ldots, G_{i_k}\}$, so dass $w_{i_1} + \cdots + w_{i_k}$ maximal wird unter der Bedingung $g_{i_1} + \cdots + g_{i_k} \leq g$.

Derartige Fragen sind sogenannte **NP**-vollständige Probleme (nicht deterministisch polynomial), die aufwendig zu berechnen sind.

Ein Spezialfall des Knapsackproblems ist:
Gegeben sind die natürlichen Zahlen $a_1, \ldots, a_n$ und g. Gesucht sind $x_1, \ldots, x_n \in \{0, 1\}$ mit $g = \sum_{i=1}^{n} x_i a_i$ (wo also $g_i = a_i = w_i$ gewählt ist). Dieses Problem heißt auch **0-1-Knapsackproblem** und wird mit $K(a_1, \ldots, a_n; g)$ bezeichnet.

Zwei 0-1-Knapsackprobleme $K(a_1, \ldots, a_n; g)$ und $K(a'_1, \ldots, a'_n; g')$ heißen kongruent, falls es zwei teilerfremde Zahlen w und m gibt, so dass

1. $m > \max\{\sum_{i=1}^{n} a_i, \sum_{i=1}^{n} a'_i\}$,
2. $g \equiv wg' \mod m$,
3. $a_i \equiv wa'_i \mod m$ für alle $i = 1, \ldots, n$.

Bemerkung: Kongruente 0-1-Knapsackprobleme haben dieselben Lösungen. Ein schneller Algorithmus zur Klärung der Frage, ob zwei 0-1-Knapsackprobleme kongruent sind, ist nicht bekannt.

Das Lösen eines 0-1-Knapsackproblems kann durch Probieren der 2^n Möglichkeiten für $x_1, \ldots, x_n$ erfolgen. Die beste Methode erfordert $O(2^{n/2})$ Operationen, was für $n = 100$ mit $2^{100} \approx 1,27 \cdot 10^{30}$ und $2^{n/2} \approx 1,13 \cdot 10^{15}$ für Computer eine unüberwindbare Hürde darstellt. Allerdings ist die Lösung für spezielle $a_1, \ldots, a_n$ recht einfach zu finden, etwa für $a_i = 2^{i-1}$. Die binäre Darstellung von g liefert unmittelbar $x_1, \ldots, x_n$. Allgemein ist die Lösung des 0-1-Knapsackproblems leicht zu finden, falls eine Permutation[104] π von $1, \ldots, n$ mit $a_{\pi(j)} > \sum_{i=1}^{j-1} a_{\pi(i)}$ existiert. Ist zusätzlich π die Identität, d.h. $\pi(i) = i$ für $i = 1, 2, \ldots, n$, so heißt die Folge $a_1, \ldots, a_n$ superwachsend. Der folgende Algorithmus löst das Knapsackproblem mit superwachsender Folge im Zeitraum von $O(n)$.

[104] Eine Permutation π der Zahlen $1, \ldots, n$ ist die Vertauschung der Reihenfolge, in der diese Zahlen aufgezählt werden. Beispielsweise ist eine Permutation π von $(1, 2, 3)$ gleich $(3, 1, 2)$, also $\pi(1) = 3$, $\pi(2) = 1$ und $\pi(3) = 2$.

for $i = n$ **to** 1 **do**
    if $T \geq a_i$ **then**
        $T := T - s_i$
        $x_i := 1$
    else
        $x_i := 0$
if $T = 0$ **then**
    $X := (x_1, \ldots, x_n)$ ist die Lösung.
else
    Es gibt keine Lösung.

Algorithmus 1. Lösen von Knapsackproblemen mit superwachsenden Gewichten

4.2.2 Merkle-Hellman Knapsack-Verschlüsselung

1978 gaben Merkle und Hellman [Merkle1978] ein Public Key-Verschlüsselungs-Verfahren an, das darauf beruht, das leichte 0-1-Knapsackproblem mit einer superwachsenden Folge in ein kongruentes mit einer nicht superwachsenden Folge zu „verfremden“. Es ist eine Blockchiffrierung, die bei jedem Durchgang einen n Bit langen Klartext chiffriert. Genauer:

Es sei $(a_1, \ldots, a_n)$ superwachsend. Seien m und w zwei teilerfremde Zahlen mit $m > \sum_{i=1}^{n} a_i$ und $1 \leq w \leq m-1$. Wähle $\bar{w}$ mit $w\bar{w} \equiv 1 \mod m$ die modulare Inverse von w und setze $b_i := wa_i \mod m$, $0 \leq b_i < m$ für $i = 1, \ldots, n$, und prüfe, ob die Folge $b_1, \ldots b_n$ nicht superwachsend ist. Danach wird eine Permutation $b_{\pi(1)}, \ldots, b_{\pi(n)}$ von $b_1, \ldots, b_n$ publiziert und insgeheim die zu π inverse Permutation μ festgehalten. Ein Sender schreibt seine Nachricht in Blöcke $(x_1^{(j)}, \ldots, x_n^{(j)})$ von Binärzahlen der Länge n und bildet

$$g^{(j)} := \sum_{i=1}^{n} x_i^{(j)} b_{\pi(i)}$$

und sendet $g^{(j)}, (j = 1, 2, \ldots)$.
Der Schlüsselinhaber bildet

$$G^{(j)} := \bar{w} g^{(j)} \mod m, \quad 0 \leq G^{(j)} < m$$

und verschafft sich die $x_{\mu(i)}^{(j)} \in \{0, 1\}$ (und somit auch die $x_i^{(j)}$) aus

$$\begin{aligned} G^{(j)} &\equiv \bar{w} g^{(j)} = \sum_{i=1}^{n} x_i^{(j)} b_{\pi(i)} \bar{w} \equiv \sum_{i=1}^{n} x_i^{(j)} a_{\pi(i)} \mod m \\ &= \sum_{i=1}^{n} x_{\mu(i)}^{(j)} a_{\pi(\mu(i))} = \sum_{i=1}^{n} x_{\mu(i)}^{(j)} a_i \mod m, \end{aligned}$$

indem er die leichten 0-1-Knapsackprobleme $K(a_1, \ldots, a_n; G^{(j)})$ mit superwachsender Folge $a_1, \ldots, a_n$ löst.

Merkle-Hellman Verfahren (auf Knapsackproblemen basierend).

1982 gab Shamir [Shamir1982] einen Algorithmus zum Brechen des Systems in polynomialer Zeit an, ohne das allgemeine Knapsackproblem zu lösen. Len Adleman [Adleman1982] und Jeff Lagarias [Lagarias1983] gaben einen Algorithmus zum Brechen des 2-fachen iterierten Merkle-Hellman Knapsack-Verschlüsselungsverfahrens in polynomialer Zeit an. Ernst Brickell [Brickell1985] gab schließlich einen Algorithmus zum Brechen des mehrfachen iterierten Merkle-Hellman Knapsack-Verschlüsselungsverfahren in polynomialer Zeit an. Damit war dieses Verfahren als Verschlüsselungsverfahren ungeeignet. Dieses Verfahren liefert also eine Einwegfunktion, deren Falltür-Information (Verfremden des 0-1-Knapsackproblems) durch einen Gegner entdeckt werden könnte.

4.3 Primfaktorzerlegung als Basis für Public Key-Verfahren

4.3.1 Das RSA-Verfahren[105]

Bereits 1978 stellten R. Rivest, A. Shamir, L. Adleman [RSA1978] das bis heute wichtigste asymmetrische Kryptographie-Verfahren vor.

Schlüsselgenerierung:

Seien p und q zwei verschiedene Primzahlen und $N = pq$. Sei e eine frei wählbare, zu $\phi(N)$ relative Primzahl, d.h. $\text{ggT}(e, \phi(N)) = 1$. Mit dem Euklidschen Algorithmus berechnet man die natürliche Zahl $d < \phi(N)$, so dass gilt

$$ed \equiv 1 \mod \phi(N).$$

Dabei ist ϕ die **Eulersche Phi-Funktion**.
Der Ausgangstext wird in Blöcke zerlegt und verschlüsselt, wobei jeder Block einen binären Wert $x^{(j)} \leq N$ hat.

Öffentlicher Schlüssel:

$$N, e.$$

Privater Schlüssel:

$$d.$$

Verschlüsselung:

$$y = e_T(x) = x^e \mod N.$$

Entschlüsselung:

$$d_T(y) = y^d \mod N$$

RSA-Verfahren (auf dem Faktorisierungsproblem basierend).

Bemerkung: Die Eulersche Phi-Funktion ist definiert duch: $\phi(N)$ ist die Anzahl der zu N teilerfremden natürlichen Zahlen $x \leq N$. Zwei natürliche Zahlen a und b sind teilerfremd, falls $\text{ggT}(a, b) = 1$.

Für die Eulersche Phi-Funktion gilt: $\phi(1) = 1,\ \phi(2) = 1,\ \phi(3) = 2,\ \phi(4) = 2,\ \phi(6) = 2,\ \phi(10) = 4,\ \phi(15) = 8$.

Zum Beispiel ist $\phi(24) = 8$, weil $|\{x < 24 : \text{ggT}(x, 24) = 1\}| = |\{1, 5, 7, 11, 13, 17, 19, 23\}|$.

Ist p eine Primzahl, so gilt $\phi(p) = p - 1$.

[105] Mit CrypTool v1.3 können Sie praktische Erfahrungen mit dem RSA-Verfahren sammeln: per Menü **Einzelverfahren \ RSA-Demo \ RSA-Kryptosystem**.

Kennt man die verschiedenen Primfaktoren $p_1, \ldots, p_k$ von N, so ist $\phi(N) = N \cdot (1 - \frac{1}{p_1}) \cdots (1 - \frac{1}{p_k})$[106].
Im Spezialfall $N = pq$ ist $\phi(N) = pq(1 - 1/p)(1 - 1/q) = p(1 - 1/p)q(1 - 1/q) = (p - 1)(q - 1)$.

n	$\phi(n)$	Die zu n teilerfremden natürlichen Zahlen kleiner n.
1	1	1
2	1	1
3	2	1, 2
4	2	1, 3
5	4	1, 2, 3, 4
6	2	1, 5
7	6	1, 2, 3, 4, 5, 6
8	4	1, 3, 5, 7
9	6	1, 2, 4, 5, 7, 8
10	4	1, 3, 7, 9
15	8	1, 2, 4, 7, 8, 11, 13, 14

Die Funktion e_T ist eine Einwegfunktion, deren Falltür-Information die Primfaktorzerlegung von N ist.

Zur Zeit ist kein Algorithmus bekannt, der das Produkt zweier Primzahlen bei sehr großen Werten geeignet schnell zerlegen kann (z.B. bei mehreren hundert Dezimalstellen). Die heute schnellsten bekannten Algorithmen [Stinson1995] zerlegen eine zusammengesetzte ganze Zahl N in einem Zeitraum proportional zu $L(N) = e^{\sqrt{\ln(N)\ln(\ln(N))}}$.

N	10^{50}	10^{100}	10^{150}	10^{200}	10^{250}	10^{300}
$L(N)$	$1,42 \cdot 10^{10}$	$2,34 \cdot 10^{15}$	$3,26 \cdot 10^{19}$	$1,20 \cdot 10^{23}$	$1,86 \cdot 10^{26}$	$1,53 \cdot 10^{29}$

Solange kein besserer Algorithmus gefunden wird, heißt dies, dass Werte der Größenordnung 100 bis 200 Dezimalstellen zur Zeit sicher sind. Einschätzungen der aktuellen Computertechnik besagen, dass eine Zahl mit 100 Dezimalstellen bei vertretbaren Kosten in etwa zwei Wochen zerlegt werden könnte. Mit einer teuren Konfiguration (z.B. im Bereich von 10 Millionen US-Dollar) könnte eine Zahl mit 150 Dezimalstellen in etwa einem Jahr zerlegt werden. Eine 200−stellige Zahl dürfte noch für eine sehr lange Zeit unzerlegbar bleiben, falls es zu keinem mathematischen Durchbruch kommt. Zum Beispiel würde es etwa 1000 Jahre dauern, um eine 200-stellige Zahl mit den bestehenden Algorithmen in Primfaktoren zu zerlegen; dies gilt selbst dann, wenn 10^{12} Operationen pro Sekunde ausgeführt werden könnten, was jenseits der Leistung der heutigen Technik liegt und in der Entwicklung Milliarden von Dollar kosten würde. Dass es aber nicht doch schon morgen zu einem mathematischen Durchbruch kommt, kann man nie ausschließen.

[106] Weitere Formeln finden sich in den Artikel „Einführung in die elementare Zahlentheorie mit Beispielen“, Kap. 3.8.

Bewiesen ist bis heute nicht, dass das Problem, RSA zu brechen äquivalent zum Faktorisierungsproblem ist. Es ist aber klar, dass wenn das Faktorisierungsproblem „gelöst“ ist, dass dann das RSA-Verfahren nicht mehr sicher ist.

4.3.2 Rabin-Public Key-Verfahren (1979)

Für dieses Verfahren konnte gezeigt werden, dass es äquivalent zum Brechen des Faktorisierungsproblems ist. Leider ist dieses Verfahren anfällig gegen chosen-ciphertext-Angriffe.

Seien p und q zwei verschiedene Primzahlen mit $p, q \equiv 3 \mod 4$ und $n = pq$. Sei $0 \leq B \leq n-1$.
Öffentlicher Schlüssel:

$$e = (n, B).$$

Privater Schlüssel:

$$d = (p, q).$$

Verschlüsselung:

$$y = e_T(x) = x(x+B) \mod n.$$

Entschlüsselung:

$$d_T(y) = \sqrt{y + B^2/4} - B/2 \mod n.$$

Rabin-Verfahren (auf dem Faktorisierungsproblem basierend).

Vorsicht: Wegen $p, q \equiv 3 \mod 4$ ist die Verschlüsselung (mit Kenntnis des Schlüssels) leicht zu berechnen. Dies ist nicht der Fall für $p \equiv 1 \mod 4$. Außerdem ist die Verschlüsselungsfunktion nicht injektiv: Es gibt genau vier verschiedene Quellcodes, die $e_T(x)$ als Urbild besitzen $x, -x - B, \omega(x + B/2) - B/2, -\omega(x + B/2) - B/2$, dabei ist ω eine der vier Einheitswurzeln. Es muss also eine Redundanz der Quellcodes geben, damit die Entschlüsselung trotzdem eindeutig bleibt!
Hintertür-Information ist die Primfaktorzerlegung von $n = pq$.

4.4 Der diskrete Logarithmus als Basis für Public Key-Verfahren

Diskrete Logarithmen sind die Grundlage für eine große Anzahl von Algorithmen von Public Key-Verfahren.

4.4.1 Der diskrete Logarithmus in $\mathbb{Z}_p^*$

Sei p eine Primzahl, und sei g ein Erzeuger der zyklischen multiplikativen Gruppe $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$. Dann ist die diskrete Exponentialfunktion zur Basis g definiert durch

$$e_g : k \longrightarrow y := g^k \mod p, \quad 1 \leq k \leq p-1.$$

Die Umkehrfunktion wird diskrete Logarithmusfunktion $\log_g$ genannt; es gilt

$$\log_g(g^k) = k.$$

Unter dem Problem des diskreten Logarithmus (in $\mathbb{Z}_p^*$) versteht man das folgende:

Gegeben p, g (ein Erzeuger der Gruppe $\mathbb{Z}_p^*$) und y, bestimme k so, dass $y = g^k \mod p$ gilt.

Die Berechnung des diskreten Logarithmus ist viel schwieriger als die Auswertung der diskreten Exponentialfunktion (siehe Kapitel 3.9). Es gibt viele Verfahren zur Berechung des diskreten Logarithmus [Stinson1995]:

Name	Komplexität
Baby-Step-Giant-Step	$O(\sqrt{p})$
Silver-Pohlig-Hellman	polynomial in q, dem größten Primteiler von $p-1$.
Index-Calculus	$O(e^{(1+o(1))\sqrt{\ln(p)\ln(\ln(p))}})$

4.4.2 Diffie-Hellman-Schlüsselvereinbarung[107]

Die Mechanismen und Algorithmen der klassischen Kryptographie greifen erst dann, wenn die Teilnehmer bereits den geheimen Schlüssel ausgetauscht haben. Im Rahmen der klassischen Kryptographie führt kein Weg daran vorbei, dass Geheimnisse kryptographisch ungesichert ausgetauscht werden müssen. Die Sicherheit der Übertragung muss hier durch nicht-kryptographische Methoden erreicht werden. Man sagt dazu, dass man zum Austausch der Geheimnisse einen geheimen Kanal braucht; dieser kann physikalisch oder organisatorisch realisiert sein.
Das Revolutionäre der modernen Kryptographie ist unter anderem, dass man keine geheimen Kanäle mehr braucht: Man kann geheime Schlüssel über nicht-geheime, also öffentliche Kanäle vereinbaren.
Ein Protokoll, das dieses Problem löst, ist das von Diffie und Hellman.

[107] In CrypTool v1.3.04 ist dieses Austauschprotokoll visualisiert: Sie können die einzelnen Schritte mit konkreten Zahlen nachvollziehen per **Einzelverfahren \ Diffie-Hellman-Demo**.

Zwei Teilnehmer A und B wollen einen gemeinsamen geheimen Schlüssel vereinbaren.
Sei p eine Primzahl und g eine natürliche Zahl. Diese beide Zahlen müssen nicht geheim sein.
Zunächst wählen sich die beiden Teilnehmer je eine geheime Zahl a bzw. b. Daraus bilden sie die Werte $\alpha = g^a \mod p$ und $\beta = g^b \mod p$. Dann werden die Zahlen α und β ausgetauscht. Schließlich potenziert jeder den erhaltenen Wert mit seiner geheimen Zahl und erhält $\beta^a \mod p$ bzw. $\alpha^b \mod p$.
Damit gilt

$$\beta^a \equiv (g^b)^a \equiv g^{ba} \equiv g^{ab} \equiv (g^a)^b \equiv \alpha^b \mod p$$

Diffie-Hellman-Schlüsselvereinbarung.

Die Sicherheit des **Diffie-Hellman-Protokolls** hängt eng mit der Berechnung der diskreten Logarithmus modulo p zusammen. Es wird sogar vermutet, dass diese Probleme äquivalent sind.

4.4.3 ElGamal-Public Key-Verschlüsselungsverfahren in $\mathbb{Z}_p^*$

Indem man das Diffie-Hellman Schlüsselvereinbarungsprotokoll leicht variiert, kann man einen asymmetrischen Verschlüsselungsalgorithmus erhalten. Diese Beobachtung geht auf Taher ElGamal zurück.

Sei p eine Primzahl, so dass der diskrete Logarithmus in $\mathbb{Z}_p$ schwierig zu berechnen ist. Sei $\alpha \in \mathbb{Z}_p^*$ ein primitives Element. Sei $a \in \mathbb{N}$ eine natürliche Zahl und $\beta = \alpha^a \mod p$.

Öffentlicher Schlüssel:

$$p, \alpha, \beta.$$

Privater Schlüssel:

$$a.$$

Sei $k \in \mathbb{Z}_{p-1}$ eine zufällige Zahl und $x \in \mathbb{Z}_p^*$ der Klartext.

Verschlüsselung:

$$e_T(x, k) = (y_1, y_2),$$

wobei

$$y_1 = \alpha^k \mod p,$$

und

$$y_2 = x\beta^k \mod p.$$

Entschlüsselung:

$$d_T(y_1, y_2) = y_2(y_1^a)^{-1} \mod p.$$

ElGamal Verfahren (auf dem diskreten Logarithmusproblem basierend).

4.4.4 Verallgemeinertes ElGamal-Public Key-Verschlüsselungsverfahren

Den diskreten Logarithmus kann man in beliebigen endlichen Gruppen $(G, \circ)$ verallgemeinern. Im folgenden geben wir einige Eigenschaften über die Gruppe G an, damit das diskrete Logarithmusproblem schwierig wird.

Berechnung der diskreten Exponentialfunktion Sei G eine Gruppe mit der Operation $\circ$ und $g \in G$. Die (diskrete) Exponentialfunktion zur Basis g ist definiert durch

$$e_g : k \longmapsto g^k, \quad \text{für alle } k \in \mathbb{N}.$$

Dabei definiert man

$$g^k := \underbrace{g \circ \ldots \circ g}_{k \text{ mal}}.$$

Die Exponentialfunktion ist leicht zu berechnen:

Lemma.

Die Potenz g^k kann in höchstens $2\log_2 k$ Gruppenoperationen berechnet werden.

Beweis
Sei $k = 2^n + k_{n-1}2^{n-1} + \cdots + k_1 2 + k_0$ die Binärdarstellung von k. Dann ist $n \leq \log_2(k)$, denn $2^n \leq k < 2^{n+1}$. k kann in der Form $k = 2k' + k_0$ mit $k' = 2^{n-1} + k_{n-1}2^{n-2} + \cdots + k_1$ geschrieben werden. Es folgt

$$g^k = g^{2k'+k_0} = (g^{k'})^2 g^{k_0}.$$

Man erhält also g^k aus $g^{k'}$ indem man einmal quadriert und eventuell mit g multipliziert. Damit folgt die Behauptung durch Induktion nach n. □

Problem des diskreten Logarithmus'

> Sei G eine endliche Gruppe mit der Operation $\circ$. Sei $\alpha \in G$ und $\beta \in H = \{\alpha^i : i \geq 0\}$.
> Gesucht ist der eindeutige $a \in \mathbb{N}$ mit $0 \leq a \leq |H| - 1$ und $\beta = \alpha^a$.
> Wir bezeichnen a mit $\log_\alpha(\beta)$.

Berechung des diskreten Logarithmus' Ein einfaches Verfahren zur Berechnung des diskreten Logarithmus' eines Gruppenelements, das wesentlich effizienter ist als das bloße Durchprobieren aller möglichen Werte für k, ist der Baby-Step-Giant-Step-Algorithmus.

Satz 4.1 (Baby-Step-Giant-Step-Algorithmus). *Sei G eine Gruppe und $g \in G$. Sei n die kleinste natürliche Zahl mit $|G| \leq n^2$. Dann kann der diskrete Logarithmus eines Elements $h \in G$ zur Basis g berechnet werden, indem man zwei Listen mit jeweils n Elementen erzeugt und diese Listen vergleicht.*
Zur Berechnung dieser Listen braucht man $2n$ Gruppenoperationen.

Beweis
Zuerst bilde man die zwei Listen
Giant-Step-Liste: $\{1, g^n, g^{2n}, \ldots, g^{n \cdot n}\}$,
Baby-Step-Liste: $\{hg^{-1}, hg^{-2}, \ldots, hg^{-n}\}$.

Falls $g^{jn} = hg^{-i}$, also $h = g^{i+jn}$, so ist das Problem gelöst. Falls die Listen disjunkt sind, so ist h nicht als $g^{i+jn}, i, j \leq n$, darstellbar. Da dadurch alle Potenzen von g erfasst werden, hat das Logarithmusproblem keine Lösung. □

Man kann sich mit Hilfe des Baby-Step-Giant-Step-Algorithmus klar machen, dass die Berechnung des diskreten Logarithmus' sehr viel schwieriger ist als die Berechnung der diskreten Exponentialfunktion. Wenn die auftretenden Zahlen etwa 1000 Bit Länge haben, so benötigt man zur Berechnung von g^k nur etwa 2000 Multiplikationen, zur Berechnung des diskreten Logarithmus' mit dem Baby-Step-Giant-Step-Algorithmus aber etwa $2^{500} \approx 10^{150}$ Operationen.
Neben dem Baby-Step-Giant-Step-Algorithmus gibt es noch zahlreiche andere Verfahren zur Berechnung des diskreten Logarithmus' [Stinson1995].

Der Satz von Silver-Pohlig-Hellman In endlichen abelschen Gruppen lässt sich das diskrete Logarithmusproblem in Gruppen kleinerer Ordnung reduzieren.

Satz 4.2 (Silver-Pohlig-Hellman). *Sei G eine endliche abelsche Gruppe mit $|G| = p_1^{a_1} p_2^{a_2} \cdot \ldots \cdot p_s^{a_s}$. Dann lässt sich das diskrete Logarithmusproblem in G auf das Lösen von Logarithmenproblemen in Gruppen der Ordnung $p_1, \ldots, p_s$ zurückführen.*

Enthält $|G|$ eine „dominanten“ Primteiler p, so ist die Komplexität des Logarithmenproblem ungefähr

$$O(\sqrt{p}).$$

Wenn also das Logarithmusproblem schwer sein soll, so muss die Ordnung der verwendeten Gruppe G einen großen Primteiler haben. Insbesondere gilt, wenn die diskrete Exponentialfunktion in der Gruppe $\mathbb{Z}_p^*$ eine Einwegfunktion sein soll, so muss $p-1$ einen großen Primteiler haben.

Sei G eine endliche Gruppe mit Operation $\circ$, und sei $\alpha \in G$, so dass der diskrete Logarithmus in $H = \{\alpha^i : i \geq 0\}$ schwer ist. Sei a mit $0 \leq a \leq |H| - 1$ und sei $\beta = \alpha^a$.
Öffentlicher Schlüssel:

$$\alpha, \beta.$$

Privater Schlüssel:

$$a.$$

Sei $k \in \mathbb{Z}_{|H|}$ eine zufällige Zahl und $x \in G$ ein Klartext.
Verschlüsselung:

$$e_T(x, k) = (y_1, y_2),$$

wobei

$$y_1 = \alpha^k,$$

und

$$y_2 = x \circ \beta^k.$$

Entschlüsselung:

$$d_T(y_1, y_2) = y_2 \circ (y_1^a)^{-1}.$$

Verallgemeinertes ElGamal Verfahren (auf das diskrete Logarithmusproblem basierend).

Elliptische Kurven liefern nützliche Gruppen für Public Key-Verschlüsselungsverfahren.

Literatur

[Adleman1982] Adleman L.: *On breaking the iterated Merkle-Hellman public key Cryptosystem.*
Advances in Cryptologie, Proceedings of Crypto 82, Plenum Press 1983, 303-308.

[Balcazar1988] Balcazar J.L., Daaz J., Gabarr´ J.: *Structural Complexity I.*
Springer Verlag, pp 63.

[Brickell1985] Brickell E.F.: *Breaking Iterated Knapsacks.*
Advances in Cryptology: Proc. CRYPTÓ84, Lecture Notes in Computer Science, vol. 196, Springer-Verlag, New York, 1985, pp. 342-358.

[Lagarias1983] Lagarias J.C.: *Knapsack public key Cryptosystems and diophantine Approximation.*
Advances in Cryptology, Proseedings of Crypto 83, Plenum Press.

[Merkle1978] Merkle R. and Hellman M.: *Hiding information and signatures in trapdoor knapsacks.*
IEEE Trans. Information Theory, IT-24, 1978.

[RSA1978] Rivest R.L., Shamir A. and Adleman L.: *A Method for Obtaining Digital Signatures and Public Key Cryptosystems.*
Commun. ACM, vol 21, April 1978, pp. 120-126.

[Shamir1982] Shamir A.: *A polynomial time algorithm for breaking the basic Merkle-Hellman Cryptosystem.*
Symposium on Foundations of Computer Science (1982), 145-152.

[Stinson1995] Stinson D.R.: *Cryptography.*
CRC Press, Boca Raton, London, Tokyo, 1995.

Web-Links

1. http://www.geocities.com/st_busygin/clipat.html

5 Digitale Signaturen

(Esslinger B. / Filipovics B. / Schneider J. / Koy H., Juni 2002, Update: Feb. 2003)

Ziel der digitalen Signatur ist es, folgende zwei Punkte zu gewährleisten:

- Benutzerauthentizität:
  Es kann überprüft werden, ob eine Nachricht tatsächlich von einer bestimmten Person stammt.
- Nachrichtenintegrität:
  Es kann überprüft werden, ob die Nachricht (unterwegs) verändert wurde.

Zum Einsatz kommt wieder eine asymmetrische Technik (siehe Verschlüsselungsverfahren). Ein Teilnehmer, der eine digitale Signatur für ein Dokument erzeugen will, muss ein Schlüsselpaar besitzen. Er benutzt seinen geheimen Schlüssel, um Signaturen zu erzeugen, und der Empfänger benutzt den öffentlichen Schlüssel des Absenders, um die Richtigkeit der Signatur zu überprüfen. Es darf wiederum nicht möglich sein, aus dem öffentlichen den geheimen Schlüssel abzuleiten[108].

Im Detail sieht ein *Signaturverfahren* folgendermaßen aus:
Der Absender berechnet aus seiner Nachricht und seinem geheimen Schlüssel die digitale Signatur der Nachricht. Im Vergleich zur handschriftlichen Unterschrift hat die digitale Signatur also den Vorteil, dass die Unterschrift auch vom unterschriebenen Dokument abhängt. Die Unterschriften ein und desselben Teilnehmers sind verschieden, sofern die unterzeichneten Dokumente nicht vollkommen übereinstimmen. Selbst das Einfügen eines Leerzeichens in den Text würde zu einer anderen Signatur führen. Eine Verletzung der Nachrichtenintegrität wird also vom Empfänger der Nachricht erkannt, da in diesem Falle die Signatur nicht mehr zum Dokument passt und sich bei der Überprüfung als unkorrekt erweist.

Das Dokument wird samt Signatur an den Empfänger verschickt. Dieser kann mit Hilfe des öffentlichen Schlüssels des Absenders, des Dokuments und der Signatur feststellen, ob die Signatur korrekt ist. In der Praxis hat das gerade beschriebene Verfahren jedoch einen entscheidenden Nachteil. Die Signatur ist ungefähr genauso lang wie das eigentliche Dokument. Um den Datenverkehr nicht unnötig anwachsen zu lassen und aus Performance-Gründen benutzt man eine kryptographische Hashfunktion[109].

---

[108] Mit CrypTool v1.3 können Sie ebenfalls digitale Signaturen erzeugen und prüfen:
in den Untermenüs des Hauptmenüpunktes **Digitale Signaturen** oder
per **Einzelverfahren \ RSA-Demo \ Signaturdemo**.

[109] Hashfunktionen sind in CrypTool v1.3.04 an mehreren Stellen implementiert.
In dem Menü **Einzelverfahren \ Hashwerte** haben Sie die Möglichkeit

- 6 Hashfunktionen auf den Inhalt des aktiven Fensters anzuwenden,
- für eine Datei den Hashwert zu berechnen,
- in der Hash-Demo die Auswirkung von Textänderungen auf den Hashwert zu testen und
- aufgrund von gezielt gesuchten Hashwertkollisionen einen Angriff auf digitale Signaturen zu simulieren.

Eine kryptographische *Hashfunktion* bildet eine Nachricht beliebiger Länge auf eine Zeichenfolge mit konstanter Größe (meistens 128 oder 160 Bits), den Hashwert, ab. Es sollte praktisch unmöglich sein, zu einer gegebenen Zahl eine Nachricht zu finden, die genau diese Zahl als Hashwert hat. Ferner sollte es praktisch unmöglich sein, zwei Nachrichten mit dem selben Hashwert zu finden. In beiden Fällen würde das Signaturverfahren Schwächen aufweisen.

Bisher konnte die Existenz von perfekt sicheren kryptographischen Hashfunktionen nicht formal bewiesen werden. Es gibt jedoch einige gute Kandidaten, die in der Praxis bislang keine Schwächen gezeigt haben (zum Beispiel SHA-1 oder RIPEMD-160).

Das Verfahren mit Hashfunktion sieht folgendermaßen aus:
Anstatt das eigentliche Dokument zu signieren, berechnet der Absender nun zuerst den Hashwert der Nachricht und signiert diesen. Der Empfänger bildet ebenfalls den Hashwert der Nachricht (der benutzte Algorithmus muss bekannt sein). Er überprüft dann, ob die mitgeschickte Signatur eine korrekte Signatur des Hashwertes ist. Ist dies der Fall, so wurde die Signatur korrekt verifiziert. Die Authentizität der Nachricht ist damit gegeben, da wir angenommen hatten, dass aus der Kenntnis des öffentlichen Schlüssels nicht der geheime Schlüssel abgeleitet werden kann. Dieser geheime Schlüssel wäre jedoch notwendig, um Nachrichten in einem fremden Namen zu signieren.

Einige digitale Signaturverfahren basieren auf asymmetrischer Verschlüsselung, das bekannteste Beispiel dieser Gattung ist RSA. Für die RSA-Signatur verwendet man die gleiche mathematische Operation wie zum Entschlüsseln, nur wird sie auf den Hash-Wert des zu unterschreibenden Dokuments angewendet.

Andere Systeme der digitalen Signatur wurden, wie DSA (Digital Signature Algorithm), ausschließlich zu diesem Zweck entwickelt, und stehen in keiner direkten Verbindung zu einem entsprechenden Verschlüsselungsverfahren.

Beide Signaturverfahren, RSA und DSA, werden in den folgenden beiden Abschnitten näher beleuchtet. Anschließend gehen wir einen Schritt weiter und zeigen, wie basierend auf der elektronischen Unterschrift das digitale Pendant zum Personalausweis entwickelt wurde. Dieses Verfahren nennt man Public Key-Zertifizierung.

5.1 RSA-Signatur

Wie im Kommentar am Ende von Abschnitt 3.10.3 bemerkt, ist es möglich, die RSA-Operationen mit dem privaten und öffentlichen Schlüssel in umgekehrter Reihenfolge auszuführen, d. h. M hoch d hoch e (mod N) ergibt wieder M. Wegen dieser simplen Tatsache ist es möglich, RSA als Signaturverfahren zu verwenden.

Eine RSA-Signatur S zur die Nachricht M wird durch folgende Operation mit dem privaten Schlüssel erzeugt:

$$S \equiv M^d \pmod{N}$$

Zur Verifikation wird die korrespondierende Public Key-Operation auf der Signatur S ausgeführt und das Ergebnis mit der Nachricht M verglichen:

$$S^e \equiv (M^d)^e \equiv (M^e)^d \equiv M \pmod{N}$$

Wenn das Ergebnis S^e mit der Nachricht M übereinstimmt, dann akzeptiert der Prüfer die Signatur, andernfalls ist die Nachricht entweder verändert worden, oder sie wurde nicht vom Inhaber von d unterschrieben.

Wie weiter oben erklärt, werden Signaturen in der Praxis nie direkt auf der Nachricht ausführt, sondern auf einem kryptographischen Hashwert davon. Um verschiedene Attacken auf das Signaturverfahren (und seine Kombination mit Verschlüsselung) auszuschließen, ist es nötig, den Hashwert vor der Exponentiation auf spezielle Weise zu formatieren, wie in PKCS#1 (Public Key Cryptography Standard #1 [PKCS1]) beschrieben. Der Tatsache, dass dieser Standard nach mehreren Jahren Einsatz revidiert werden musste, kann als Beispiel dafür dienen, wie schwer es ist, kryptographische Details richtig zu definieren.

5.2 DSA-Signatur

Im August 1991 hat das U.S. National Institute of Standards and Technology (NIST) einen digitalen Signaturalgorithmus (DSA, Digital Signature Algorithm) vorgestellt, der später zum U.S. Federal Information Processing Standard (FIPS 186 [FIPS186]) wurde.

Der Algorithmus ist eine Variante des ElGamal-Verfahrens. Seine Sicherheit beruht auf dem Diskreten Logarithmus Problem. Die Bestandteile des privaten und öffentlichen DSA-Schlüssels, sowie die Verfahren zur Signatur und Verifikation sind im Folgenden zusammengefasst.

Öffentlicher Schlüssel

p prim
q 160bit Primfaktor von $p-1$
$g = h^{(p-1)/q} \bmod p$, wobei $h < p-1$ und $h^{(p-1)/q} > 1 \pmod p$
$y \equiv g^x \bmod p$

Bemerkung: Die Parameter p, q und g können von einer Gruppe von Benutzern gemeinsam genutzt werden.

Privater Schlüssel

$x < q$ (160bit Zahl)

Signatur

m zu signierende Nachricht
k zufällig gewählte Primzahl, kleiner als q
$r = (g^k \bmod p) \bmod q$
$s = (k^{-1}(\text{SHA-1}(m) + xr)) \bmod q$

Bemerkung:

- (s, r) ist die Signatur.
- Die Sicherheit der Signatur hängt nicht nur von der Mathematik ab, sondern auch von der Verfügbarkeit einer guten Zufallsquelle für k.

- SHA-1 ist eine in FIPS186 spezifizierte 160-Bit Hashfunktion.

Verifikation

$w = s^{-1} \bmod q$
$u_1 = (\text{SHA-1}(m)w) \bmod q$
$u_2 = (rw) \bmod q$
$v = (g^{u_1} y^{u_2}) \bmod p) \bmod q$

Bemerkung: Wenn $v = r$, dann ist die Signatur gültig.

Obwohl DSA unabhängig von einem Verschlüsselungsverfahren so spezifiziert wurde, dass es aus Länder exportiert werden kann, die den Export von kryptographischer Hard- und Software einschränken, wie die USA zum Zeitpunkt der Spezifikation, wurde festgestellt [Schneier1996, S. 490], dass die Operationen des DSA dazu geeignet sind, nach RSA bzw. ElGamal zu verschlüsseln.

5.3 Public Key-Zertifizierung

Ziel der Public Key-Zertifizierung ist es, die Bindung eines öffentlichen Schlüssels an einen Benutzers zu garantieren und nach außen nachvollziehbar zu machen. In Fällen, in denen nicht sichergestellt werden kann, dass ein öffentlicher Schlüssel auch wirklich zu einer bestimmten Person gehört, sind viele Protokolle nicht mehr sicher, selbst wenn die einzelnen kryptographischen Bausteine nicht geknackt werden können.

5.3.1 Die Impersonalisierungsattacke

Angenommen Charlie hat zwei Schlüsselpaare (PK1, SK1) und (PK2, SK2). Hierbei bezeichnet SK den geheimen Schlüssel (secret key) und PK den öffentlichen Schlüssel (public key). Weiter angenommen, es gelingt ihm, Alice PK1 als öffentlichen Schlüssel von Bob und Bob PK2 als öffentlichen Schlüssel von Alice „unterzujubeln“(etwa indem er ein öffentliches Schlüsselverzeichnis fälscht).

Dann ist folgender Angriff möglich:

- Alice möchte eine Nachricht an Bob senden. Sie verschlüsselt diese mit PK1, da sie denkt, dies sei Bobs öffentlicher Schlüssel. Anschließend signiert sie die Nachricht mit ihrem geheimen Schlüssel und schickt sie ab.
- Charlie fängt die Nachricht ab, entfernt die Signatur und entschlüsselt die Nachricht mit SK1. Wenn er möchte, kann er die Nachricht anschließend nach Belieben verändern. Dann verschlüsselt er sie wieder, aber diesmal mit dem echten öffentlichen Schlüssel von Bob, den er sich aus einem öffentlichen Schlüsselverzeichnis geholt hat, signiert sie mit SK2 und schickt die Nachricht weiter an Bob.
- Bob überprüft die Signatur mit PK2 und wird zu dem Ergebnis kommen, dass die Signatur in Ordnung ist. Dann entschlüsselt er die Nachricht mit seinem geheimen Schlüssel.

Charlie ist so in der Lage, die Kommunikation zwischen Alice und Bob abzuhören und die ausgetauschten Nachrichten zu verändern, ohne dass dies von den beteiligten Personen bemerkt wird. Der Angriff funktioniert auch, wenn Charlie nur ein Schlüsselpaar hat.

Ein anderer Name für diese Art von Angriffen ist „man-in-the-middle-attack“. Hilfe gegen diese Art von Angriffen verspricht die Public Key-Zertifizierung, die die Authentizität öffentlicher Schlüssel garantieren soll. Die am weitesten verbreitete Zertifizierungsmethode ist der X.509-Standard.

5.3.2 X.509

Jeder Teilnehmer, der sich per X.509-Zertifikat die Zugehörigkeit seines öffentlichen Schlüssels zu seiner realen Person bestätigen lassen möchte, wendet sich an eine sogenannte Certification Authority (CA). Dieser beweist er seine Identität (etwa durch Vorlage seines Personalausweises). Anschließend stellt die CA ihm ein elektronisches Dokument (Zertifikat) aus, in dem im wesentlichen die Namen des Zertifikatnehmers und der CA, der öffentliche Schlüssel des Zertifikatnehmers und der Gültigkeitszeitraum des Zertifikats vermerkt sind. Die CA unterzeichnet das Zertifikat anschließend mit ihrem geheimen Schlüssel.

Jeder kann nun anhand des öffentlichen Schlüssels der CA überprüfen, ob das Zertifikat unverfälscht ist. Die CA garantiert also die Zugehörigkeit von Benutzer und öffentlichem Schlüssel.

Dieses Verfahren ist nur so lange sicher, wie die Richtigkeit des öffentlichen Schlüssels der CA sichergestellt ist. Aus diesem Grund lässt jede CA ihren öffentlichen Schlüssel bei einer anderen CA zertifizieren, die in der Hierarchie über ihr steht. In der obersten Hierarchieebene (Wurzelinstanz) gibt es in der Regel nur eine CA, die dann natürlich keine Möglichkeit mehr hat, sich ihren Schlüssel bei einer anderen CA zertifizieren zu lassen. Sie ist also darauf angewiesen, ihren Schlüssel auf andere Art und Weise gesichert zu übermitteln. Bei vielen Software-Produkten, die mit Zertifikaten arbeiten (zum Beispiel den Webbrowsern von Microsoft und Netscape) sind die Zertifikate dieser Wurzel-CAs schon von Anfang an fest in das Programm eingebettet und können auch vom Benutzer nachträglich nicht mehr geändert werden. Aber auch durch öffentliche Bekanntgabe in Zeitungen können (öffentliche) CA-Schlüssel gesichert übermittelt werden.

Literatur

[Schneier1996] Bruce Schneier,
Applied Cryptography, Protocols, Algorithms, and Source Code in C,
Wiley, 2nd edition, 1996.

[PKCS1] RSA Laboratories
PKCS #1 v2.1 Draft 3: RSA Cryptography Standard, April 19, 2002.

[FIPS186] U.S. Departement of Commerce/N.I.S.T.
Entity authentication using public key cryptography, February 18, 1997.

6 Elliptische Kurven

(Filipovics B. / Büger M. / Esslinger B. / Oyono R., April 2000, Updates: Dez. 2001, Juni 2002, März 2003)

6.1 Elliptische Kurven – ein effizienter Ersatz für RSA?

In vielen Bereichen sind Sicherheit und Effizienz von Datenübertragungen von großer Wichtigkeit. Viele Anwendungen verwenden den RSA-Algorithmus als asymmetrisches Signatur- und Verschlüsselungsverfahren.

Solange die verwendeten Schlüssel hinreichend lang sind, bestehen keinerlei Bedenken gegen die *Sicherheit* des RSA-Verfahrens. Allerdings hat die Entwicklung der Rechnerleistungen in der vergangenen Jahren dazu geführt, dass die benötigten Schlüssellängen mehrfach angehoben werden mussten (vergleiche Kapitel 3.11). Da die meisten Chips auf Smartcards nicht in der Lage sind, längere Schlüssel als 1024 Bit zu verarbeiten, besteht Bedarf für Alternativen zum RSA. Elliptische Kurven können eine solche Alternative bieten.

Die *Effizienz* eines kryptographischen Algorithmus hängt wesentlich von der benötigten Schlüssellänge und vom Rechenaufwand ab, um ein vorgeschriebenes Sicherheitsniveau zu erreichen. Der entscheidende Vorteil Elliptischer Kurven im Vergleich zum RSA-Algorithmus liegt in der Tatsache, dass die sicheren Schlüssellängen erheblich kürzer sind.

Setzt man den Trend, dass sich die Leistung der verfügbaren Rechner im Schnitt alle 18 Monate verdoppelt (Gesetz von Moore[110]), in die Zukunft fort, so kann man von einer Entwicklung der sicheren Schlüssellängen wie in Abbildung 1 ausgehen [Lenstra1999]

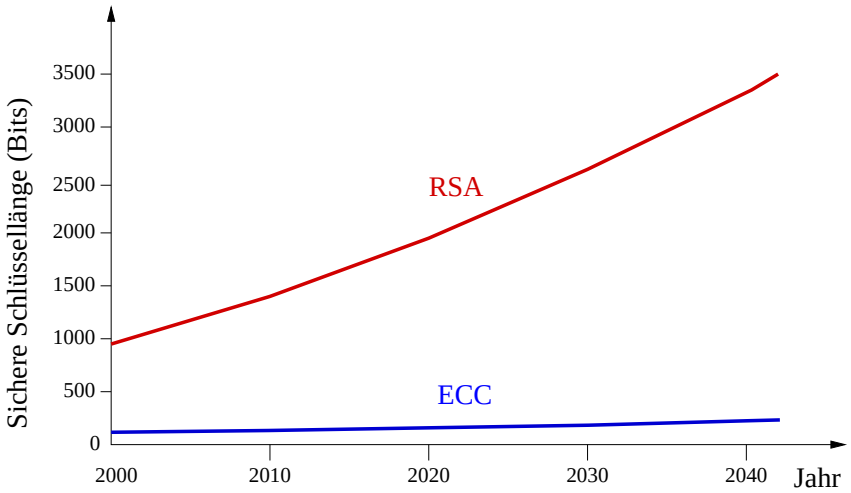


Abbildung 1: Prognose für die Entwicklung der als sicher betrachteten Schlüssellängen für RSA und Elliptische Kurven

[110] empirische Erkenntnis von Gordon Moore, Mitbegründer von Intel, 1965

Bei der digitalen Signatur muss man differenzieren: für die *Erstellung* einer digitalen Signatur benötigen auf Elliptischen Kurven basierende Verfahren im Vergleich zu RSA nur gut ein Zehntel des Rechenaufwandes (66 zu 515 Ganzzahlmultiplikationen). Siehe hierzu Abbildung 2 (Quelle: Dr. J. Merkle, Elliptic Curve Cryptography Workshop, 2001). Betrachtet man die für eine *Verifikation* durchzuführenden Rechenschritte, dreht sich dieses Bild jedoch zu Gunsten von RSA um. Der Grund liegt darin, dass es bei Verwendung des RSA möglich ist, einen sehr kurzen Exponent für den öffentlichen Schlüssel zu wählen, solange der private Exponent nur hinreichend lang ist.

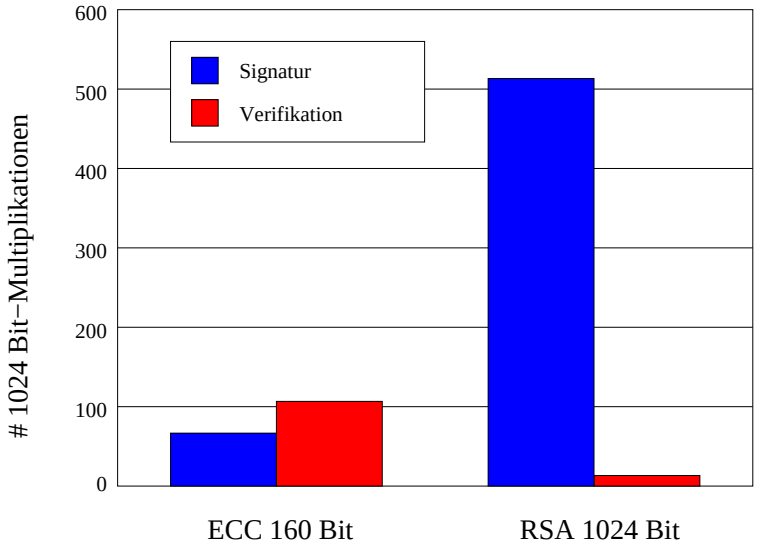


Abbildung 2: Gegenüberstellung des Aufwands der Operationen Signieren und Verifizieren bei RSA und Elliptischen Kurven

Da bei Smartcards, die auf RSA basieren, stets der lange (private) Schlüssel auf der Karte gespeichert werden muss und die Erstellung der digitalen Signatur, nicht aber die Verifikation, auf der Karte stattfindet, treten hier deutlich die Vorteile Elliptischer Kurven zutage.

Das größte Problem bei der Implementierung von Verfahren, die auf Elliptischen Kurven beruhen, ist bislang die mangelnde *Standardisierung*. Es gibt nur eine RSA-Implementierung, aber viele Arten, Elliptische Kurven einzusetzen. So können verschiedene Zahlkörper zugrunde gelegt, eine Vielzahl von (Elliptischen) Kurven — durch 6 Parameter beschrieben — eingesetzt und unterschiedliche Darstellungen der Kurvenpunkte verwendet werden. Jede Wahl hat ihre Vorzüge, so dass für jede Anwendung eine andere Implementierung optimal sein kann. Dies hat jedoch zur Konsequenz, dass Systeme, die auf Elliptischen Kurven beruhen, oftmals nicht interoperabel sind. Um mit einer beliebigen auf Elliptischen Kurven basierenden Anwendung kommunizieren zu können, müsste man eine Vielzahl von Implementierungen vorhalten, was den Effizienzvorteil gegenüber der Verwendung von RSA zunichte macht.

Deshalb bemühen sich internationale Organisationen um Standardisierung: IEEE (P1363), ASC (ANSI X9.62, X9.63), ISO/IEC sowie RSA Laboratories und Certicom. Im Gegensatz zur IEEE,

die bisher nur eine Beschreibung der verschiedenen Implementierungen vorgenommen hat, hat die ASC konkret 10 Kurven ausgewählt und empfiehlt deren Verwendung. Der Vorteil des ASC-Ansatzes ist, dass ein einziges Byte ausreicht, um die verwendete Kurve zu spezifizieren. Zur Zeit ist jedoch nicht absehbar, ob es der ASC gelingen wird, einen de-facto-Standard durchzusetzen.

Obwohl aktuell kein Handlungsbedarf besteht[111], laufende RSA-Anwendungen umzustellen, sollte man bei der Neuimplementierung ernsthaft den Einsatz von Verfahren erwägen, die auf Elliptischen Kurven basieren. Dies gilt insbesondere, wenn es sich um Anwendungen im Finanzsektor handelt, die noch nach 2005[112] operativ sein sollen.

6.2 Elliptische Kurven – Historisches

Auf dem Gebiet der Elliptischen Kurven wird seit über 100 Jahren geforscht. Im Laufe der Zeit hat man viele weitläufige und mathematisch tiefgründige Resultate im Zusammenhang mit Elliptischen Kurven gefunden und veröffentlicht. Ein Mathematiker würde sagen, dass die Elliptischen Kurven (bzw. die dahinterstehende Mathematik) gut verstanden sind. Ursprünglich war diese Forschung reine Mathematik, das heißt Elliptische Kurven wurden zum Beispiel in den mathematischen Teilgebieten Zahlentheorie und algebraische Geometrie untersucht, die allgemein sehr abstrakt sind. Auch in der nahen Vergangenheit spielten Elliptische Kurven eine bedeutende Rolle in der reinen Mathematik. In den Jahren 1993 und 1994 veröffentlichte Andrew Wiles mathematische Arbeiten, die weit über das Fachpublikum hinaus auf große Begeisterung gestoßen sind. In diesen Arbeiten bewies er die Richtigkeit einer — in den sechziger Jahren des 20. Jahrhunderts von zwei Japanern aufgestellten — Vermutung. Dabei geht es kurz und grob gesagt um den Zusammenhang zwischen Elliptischen Kurven und sogenannten Modulformen. Das für die meisten eigentlich Interessante daran ist, dass Wiles mit seinen Arbeiten auch den berühmten zweiten Satz von Fermat bewiesen hat. Dieser Satz hatte sich seit Jahrhunderten (Fermat lebte von 1601 bis 1665) einem umfassenden Beweis durch die Mathematik entzogen. Dementsprechend groß war die Resonanz auf den Beweis durch Wiles. In der Formulierung von Fermat lautet der nach ihm benannte Satz so (Fermat hat folgende Worte an den Rand des 1621 von Bachet herausgegebenen Werks Diophants geschrieben):

> *Cubum autem in duos cubos, aut quadratoquadratum in duos quadratoquadratos, et generaliter nullam in infinitum ultra quadratum potestatem in duos ejusdem nominis fas est dividere: cujus rei demonstrationem mirabilem sane detexi. Hanc marginis exiguitas non caperet.*

Frei übersetzt und mit der Schreibweise der heutigen Mathematik bedeutet dies:
Es gibt keine positiven ganzen Zahlen x, y und z größer als Null, so dass $x^n + y^n = z^n$ für $n > 2$ gilt. Ich habe einen bemerkenswerten Beweis für diese Tatsache gefunden, aber es ist nicht genug Platz am Rand [des Buches], um ihn niederzuschreiben.

Dies ist schon bemerkenswert: Eine relativ einfach zu verstehende Aussage (gemeint ist Fermats zweiter Satz) konnte erst nach so langer Zeit bewiesen werden, obwohl Fermat selber angab, schon

[111] Aktuelle Informationen zur Sicherheit des RSA-Verfahrens finden Sie in Kapitel 3.11.
[112] BSI-Empfehlung (GISA recommendation)

einen Beweis gefunden zu haben. Im übrigen ist der Beweis von Wiles sehr umfangreich (alle im Zusammenhang mit dem Beweis stehenden Veröffentlichungen von Wiles ergeben schon ein eigenes Buch). Man sollte sich daher im klaren sein, dass die Elliptischen Kurven im allgemeinen sehr tiefgreifende Mathematik berühren.

Soweit zur Rolle der Elliptischen Kurven in der reinen Mathematik. Im Jahr 1985 haben Neal Koblitz und Victor Miller unabhängig voneinander vorgeschlagen, Elliptische Kurven in der Kryptographie einzusetzen. Damit haben die Elliptischen Kurven auch eine ganz konkrete praktische Anwendung gefunden. Ein weiteres interessantes Einsatzgebiet für Elliptische Kurven ist die Faktorisierung von ganzen Zahlen (auf der Schwierigkeit/Komplexität, die Primfaktoren einer sehr großen Zahl zu finden, beruht das RSA-Kryptosystem). In diesem Bereich werden seit 1987 (eine Arbeit von H.W. Lenstra) auf Elliptischen Kurven basierende Verfahren untersucht und zum Teil eingesetzt.

Elliptische Kurven werden in den verschiedenen Gebieten unterschiedlich eingesetzt. Kryptographische Verfahren auf Basis von Elliptischen Kurven beruhen auf der Schwierigkeit eines als Elliptische Kurven Diskreter Logarithmus bekannten Problems. Die Faktorisierung von ganzen Zahlen macht sich die Tatsache zunutze, dass man für eine natürliche, aus mehreren Primzahlen zusammengesetzte Zahl n sehr viele verschiedene Elliptische Kurven erzeugen kann; diese Kurven sind dann allerdings für zusammengesetzte n keine Gruppen. Näheres dazu findet sich unter Faktorisieren mit Elliptischen Kurven.

6.3 Elliptische Kurven – Mathematische Grundlagen

In diesem Abschnitt erhalten Sie Informationen über *Gruppen* und *Körper*.

6.3.1 Gruppen

Da der Begriff der *Gruppe* umgangssprachlich anders als in der Mathematik eingesetzt wird, soll der Vollständigkeit halber an dieser Stelle die wesentliche Aussage der formalen Definition einer Gruppe kurz eingeführt werden:

- Eine Gruppe ist eine nichtleere Menge G mit einer Verknüpfung "$\cdot$". Die Menge G ist unter der Verknüpfung $\cdot$ abgeschlossen, d.h, sind a, b Elemente aus G, so ist auch ihre Verknüpfung $ab = a \cdot b$ ein Element aus G.
- Für alle Elemente a, b und c aus G gilt: $(ab)c = a(bc)$ (Assoziativgesetz).
- Es gibt ein Element e in G, das sich bezüglich der Verknüpfung $\cdot$ neutral verhält, d.h., für alle a aus der Menge G : gilt $ae = ea = a$.
- Zu jedem Element a aus G gibt es ein *inverses Element* a^{-1}(in G), so dass gilt: $aa^{-1} = a^{-1}a = e$.

Gilt zusätzlich noch für alle a, b aus G, dass $ab = ba$ (Kommutativgesetz), so nennt man die Gruppe G eine *abelsche* Gruppe.

Da man auf der selben Menge mehrere Verknüpfung erklären kann, unterscheidet man diese durch verschiedene Namensgebungen und Zeichen (z.B. + Addition oder · Multiplikation).

Als einfachstes Beispiel einer (abelschen) Gruppe sei die Gruppe der ganzen Zahlen mit der üblichen Addition genannt. Die Menge der ganzen Zahlen wird mit $\mathbb{Z}$ bezeichnet. $\mathbb{Z}$ hat unendlich viele Elemente, denn $\mathbb{Z} = \{\cdots, -4, -3, -2, -1, 0, 1, 2, 3, 4, \cdots\}$. Die Verknüpfung von zum Beispiel $1 + 2$ liegt in $\mathbb{Z}$, denn $1 + 2 = 3$ und 3 liegt in $\mathbb{Z}$. Das neutrale Element der Gruppe $\mathbb{Z}$ ist 0. Das Inverse Element von 3 ist -3, denn $3 + (-3) = 0$.

Für unsere Zwecke besonders interssant sind sogenannte *endliche* Gruppen, bei denen die zugrundelegte Menge G nur aus einer endlichen Anzahl von Elementen besteht. Beispiele sind die Gruppen $\mathbb{Z}_n = \{0, 1, 2, 3, \cdots, n-1\}, n$ der Teilerreste bei der Division durch n mit der Addition als Verknüpfung.

Zyklische Gruppen Als *zyklische Gruppen*[113] bezeichnet man solche Gruppen G', die ein Element g besitzen, aus dem man mittels der Gruppen-Verknüpfung alle anderen Elemente der Gruppe erzeugen kann. Es gibt also für jedes Element a aus G' eine positive, ganze Zahl i, so dass die i-fache Verknüpfung von g mit sich selbst $g^i = g \cdot g \cdots g = a$. Das Element g ist ein *Generator* der zyklischen Gruppe — jedes Element in G lässt sich mittels g und der Verknüpfung erzeugen.

Ordnung von Elementen einer Gruppe Nun zur Ordnung eines Elements der Gruppe: Sei a aus G. Die kleinste positive ganze Zahl r für die gilt, dass a^r, also r mal a mit sich selbst verknüpft, das neutrale Element der Gruppe G ist (d.h. $a^r = e$), nennt man *Ordnung* von a.

Die *Ordnung der Gruppe* ist die Anzahl der Elemente in der Menge G. Ist die Gruppe G zyklisch und g ein Generator, so stimmt die Ordnung von g mit der Gruppenordnung überein. Man kann leicht zeigen, dass die Ordnung eines Gruppenelements stets die Gruppenordnung teilt. Hieraus folgt insbesondere, dass Gruppen mit Primzahlordnung (d.h. die Ordnung der Gruppe ist eine Primzahl) zyklisch sind.

6.3.2 Körper

In praktischen Anwendungen betrachtet man häufig Mengen, auf denen nicht nur eine (Gruppen-) Verknüpfung, sondern zwei Verknüpfungen definiert sind. Diese nennt man oft Addition und Multiplikation. Die mathematisch interessantesten Mengen dieser Art sind sogenannte Körper, wie z.B. die Menge der reellen Zahlen.

Unter einem Körper versteht man in der Mathematik eine Menge K mit den zwei Verknüpfungen Addition und Multiplikation (mit + und · bezeichnet), so dass die folgenden Bedingungen erfüllt sind:

- Die Menge K ist zusammen mit der Verknüpfung + (Addition) eine abelsche Gruppe. Dabei sei 0 das neutrale Element der Verknüpfung +.

[113] Zyklische Gruppen sind Gruppen, die von einem Element erzeugt werden. Sie können grundsätzlich auch unendlich sein wie z.B. die additive Gruppe der ganzen Zahlen. Wir betrachten hier jedoch nur endliche zyklische Gruppen.

- Die Menge $K \setminus \{0\}$ (d.h. K ohne das Element 0) ist zusammen mit der Verknüpfung $\cdot$ (Multiplikation) ebenfalls eine abelsche Gruppe. Dabei sei 1 das neutrale Element der Verknüpfung $\cdot$.
- Für alle Elemente a, b und c aus K gilt $c \cdot (a + b) = c \cdot a + c \cdot b$ und $(a + b) \cdot c = a \cdot c + b \cdot c$ (Distributivgesetz).

Körper können endlich viele oder unendliche viele Elemente enthalten — je nach dem nennt man den Körper *endlich* oder *unendlich*. So sind die uns vertrauten Körper der rationalen bzw. der reellen Zahlen unendlich. Beispiele für endliche Körper sind die Primkörper $\mathbb{Z}_p = \{0, 1, 2, 3, \cdots, p-1\}$, p eine Primzahl, versehen mit der Addition modulo p und der Multiplikation modulo p (auch Restklassenkörper genannt).

Charakteristik eines Körpers Die Charakteristik eines Körper K ist die Ordnung des neutralen Elements der Multiplikation (1-Element) bezüglich der Addition, d.h. die kleinste natürliche Zahl n, so dass gilt

$$\underbrace{1 + 1 + \cdots + 1}_{n \text{ mal}} = 0\,,$$

wobei 0 das neutrale Element der Addition ist. Gibt es keine solche natürliche Zahl, d.h. ergibt $1 + 1 + \cdots + 1$ unabhängig von der Zahl der Summanden nie das neutrale Element der Addition 0, so sagt man, der Körper habe Charakteristik 0.

Körper mit Charakteristik 0 haben daher stets die (paarweise verschiedenen) Elemente $1, 1 + 1, 1 + 1 + 1, \ldots$ und sind folglich stets unendlich; andererseits können Körper mit endlicher Charakteristik durchaus endlich oder auch unendlich sein. Ist die Charakteristik n endlich, so muss sie eine Primzahl sein, denn wäre sie zusammengesetzt, d.h. $n = pq$, so sind $p, q < n$ und aufgrund der Minimalität der Charakteristik ist keines der Körperelemente $\bar{p} = \underbrace{1 + 1 + \cdots + 1}_{p \text{ mal}}$, $\bar{q} = \underbrace{1 + 1 + \cdots + 1}_{q \text{ mal}}$ gleich 0. Folglich existieren Inverse $\bar{p}^{-1}, \bar{q}^{-1}$ bezüglich der Multiplikation. Dann ist aber $(\bar{p}\bar{q})(\bar{p}^{-1}\bar{q}^{-1}) = 1$, andererseits ist nach Definition der Charakteristik $\bar{p}\bar{q} = n = 0$ und somit $\underbrace{(\bar{p}\bar{q})}_{=0}(\bar{p}^{-1}\bar{q}^{-1}) = 0$, was zu einem Widerspruch führt.

Beispiel: Der Körper $\mathbb{Z}_p$, p prim, hat die Charakteristik p. Ist p nicht prim, so ist $\mathbb{Z}_p$ gar kein Körper.

Der einfachste, denkbare Körper ist $\mathbb{Z}_2 = \{0, 1\}$, der nur Null- und Einselement enthält. Dabei ist $0 + 0 = 0$, $0 + 1 = 1 + 0 = 1$, $1 + 1 = 0$, $1 \cdot 1 = 1$, $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$.

Endliche Körper Wie bereits erwähnt, hat jeder endliche Körper eine Charakteristik $p \neq 0$, wobei p eine Primzahl ist. Zu jeder Primzahl p gibt es einen Körper mit p Elementen, nämlich $\mathbb{Z}_p$.

Die Anzahl der Elemente eines Körpers muss jedoch im allgemeinen keine Primzahl sein. So ist es nicht schwer, einen Körper mit 4 Elementen zu konstruieren[114].

Man kann zeigen, dass die Ordnung jedes Körpers eine Primzahlpotenz (d.h. die Potenz einer Primzahl) ist. Andererseits kann man zu jeder Primzahlpotenz p^n einen Körper konstruieren, der die Ordnung p^n hat. Da zwei endliche Körper mit gleicher Zahl von Elementen nicht unterscheidbar[115] sind, spricht man von **dem Körper mit p^n Elementen** und bezeichnet diesen mit $GF(p^n)$. Dabei steht GF für *Galois Feld* in Erinnerung an den französischen Mathematiker Galois.

Eine besondere Rolle spielen die Körper $GF(p)$, deren Ordnung eine Primzahl ist. Man nennt solche Körper Primkörper zur Primzahl p und bezeichnet ihn meist mit $\mathbb{Z}_p$[116].

6.4 Elliptische Kurven in der Kryptographie

In der Kryptographie betrachten wir Elliptische Kurven. Solche Kurven ergeben sich als Lösungen einer Gleichung. Es hat sich bei Untersuchungen gezeigt, dass die durch eine Kurve vom Grad drei[117] der Form

$$F(x_1, x_2, x_3) = -x_1^3 + x_2^2 x_3 + a_1 x_1 x_2 x_3 - a_2 x_1^2 x_3 + a_3 x_2 x_3^2 - a_4 x_1 x_3^2 - a_6 x_3^3 = 0,$$

wobei die Variablen x_1, x_2, x_3 sowie die Parameter $a_1, \ldots, a_4, a_6$ Elemente eines gegebenen Körpers K sind, der eine für die Kryptographie interessante Eigenschaften besitzt. Der zugrunde liegende Körper K kann einfach die bekannte Menge der reellen Zahlen oder auch ein endlicher Körper sein (vgl. letzter Abschnitt). Damit sich eine sinnvolle Kurve ergibt, müssen die Parameter so gewählt sein, dass die folgenden Nebenbedingungen gelten

$$\frac{\partial F}{\partial x_1} \neq 0, \quad \frac{\partial F}{\partial x_2} \neq 0, \quad \frac{\partial F}{\partial x_3} \neq 0.$$

Ferner betrachten wir Punkte, die sich nur durch eine Vervielfachung jeder Komponente ergeben, als identisch, denn mit (x_1, x_2, x_3) erfüllt stets auch $\alpha(x, y, z)$ die Ausgangsgleichung. Formal betrachten wir daher Äquivalenzklassen von Punkten (x_1, x_2, x_3), wobei wir zwei Punkte als

---

[114] Die Menge $K = \{0, 1, a, b\}$ ist mit den Verknüpfungen der folgenden Tabellen ein Körper:

$+$	0	1	a	b
0	0	1	a	b
1	1	0	b	a
a	a	b	0	1
b	b	a	1	0

und

$\cdot$	0	1	a	b
0	0	0	0	0
1	0	1	a	b
a	0	a	b	1
b	0	b	1	a

[115] Sind K, K' zwei Körper mit $k = p^n$ Elementen, so gibt es eine eineindeutige Abbildung $\varphi : K \to K'$, die sich mit der Körperarithmetik verträgt. Eine solche Abbildung nennt man Isomorphie. Isomorphe Körper verhalten sich mathematisch gleich, so dass es keinen Sinn macht, zwischen ihnen zu unterscheiden. Z.B. sind $\mathbb{Z}_2$ und $K' = \{NULL, EINS\}$ mit Nullelement $NULL$ und Einselement $EINS$ isomorph. Hierbei sei darauf hingewiesen, dass mathematische Objekte ausschließlich über ihre Eigenschaften definiert sind.

[116] Für Primkörper sind die additive Gruppe sowie die multiplikative Gruppe zyklisch. Ferner enthält jeder Körper $GF(p^n)$ einen zu $\mathbb{Z}_p$ isomorphen Primkörper.

[117] d.h. die Summe der Exponenten jedes einzelnen Summanden ist maximal 3, bei mindestens einem Summanden kommt 3 als Exponentwert auch wirklich vor

gleich ansehen, wenn sie durch Multiplikation mit einer Konstante α auseinander hervorgehen. Setzt man in der Ausgangsgleichung $x_3 = 0$, so wird diese zu $-x_1^3 = 0$, also $x_1 = 0$. Folglich ist die Äquivalenzklasse, die das Element $(0, 1, 0)$ enthält, die einzige Punkt mit $x_3 = 0$. Für alle anderen Lösungspunkte können wir die Transformation

$$K \times K \times (K \setminus \{0\}) \ni (x_1, x_2, x_3) \mapsto (x, y) := \left(\frac{x_1}{x_3}, \frac{x_2}{x_3}\right) \in K \times K$$

vornehmen, die die Anzahl der Variablen von drei auf zwei reduziert. Die Ausgangsgleichung $F(x_1, x_2, x_3) = 0$ war so gewählt, dass sich auf diese Weise die sogenannten Weierstrass-Gleichung[118]

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

ergibt. Da alle bis auf einen Lösungspunkt durch die Gleichung (1) beschrieben werden können, bezeichnet man (1) auch oft als die Elliptische Gleichung, ihre Lösungsmenge folglich mit

$$\mathbf{E} = \left\{(x, y) \in K \times K \,|\, y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\right\} \cup \{\mathcal{O}\}.$$

Dabei soll $\mathcal{O}$ den auf diese Weise nicht beschriebenen Punkt $(0, 1, 0)$ darstellen, der durch die Projektion (Division durch x_3) quasi in den unendlich fernen Punkt abgebildet wird.

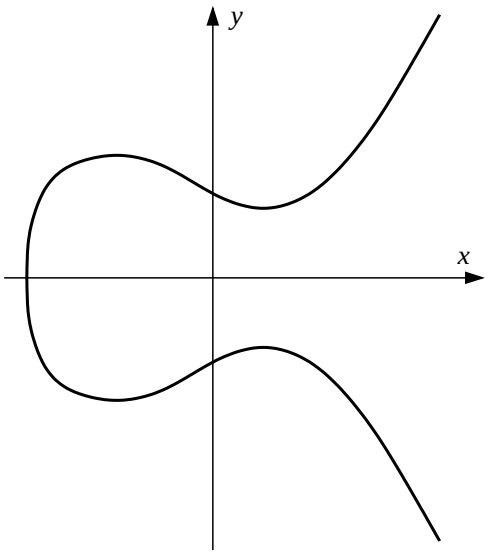


Abbildung 3: Beispiel einer elliptischen Kurve über dem Körper der reellen Zahlen.

Als zugrunde liegenden Körper für eine Elliptische Kurve verwendet man in der Kryptographie stets endliche Körper $K = GF(p^n)$, also nicht wie in Abblidung 3 die zu einer stetigen Kurve führenden reellen Zahlen. Der Grund liegt, einfach gesagt, darin, dass wir bei der Verarbeitung und Übertragung von Nachrichten stets nur endlich viele Zustände zur Verfügung haben (aufgrund der Arbeitsweise moderner Computer), Körper mit unendlich vielen Elementen wie z.B. die reellen Zahlen daher stets nur unvollständig darstellen können.

[118] Karl Weierstrass, 1815-1897, dt. Mathematiker, Verfechter der streng formalen Ausrichtung der Mathematik.

In der Praxis hat es sich als sinnvoll erwiesen, entweder $GF(p)$ mit einer großen Primzahl p oder $GF(2^n)$ mit einer (großen) natürlichen Zahl n zu betrachten. Der Grund für die Verwendung des Primkörpers $GF(p)$ liegt in seiner einfachen Arithmetik; andererseits kommt $GF(2^n)$ der binären Darstellung in Computersystemen entgegen. Andere Körper wie z.B. $GF(7^n)$ bieten keiner dieser beiden Vorteile und werden daher in der Praxis nicht verwendet, ohne dass dies theoretische Gründe hätte.

Durch Koordinatentransformation kann man die Weierstrass-Gleichung in eine einfachere Form schreiben[119] Je nachdem, ob $p > 3$ ist, verwendet man unterschiedliche Transformationen und erhält so

- im Fall $GF(p)$, $p > 3$, Elliptische Gleichungen der Form

$$y^2 = x^3 + ax + b \tag{2}$$

  mit $4a^3 + 27b^2 \neq 0$

- im Fall $GF(2^n)$ Gleichungen der Form

$$y^2 + xy = x^3 + ax^2 + b \tag{3}$$

  mit $b \neq 0$[120].

Durch diese Bedingungen an die Parameter a, b ist gewährleistet, dass die Elliptische Gleichung für kryptographische Anwendungen geeignet ist[121].

Für die Anzahl $|E|$ der Elemente einer Elliptischen Kurve E über einem Körper $GF(k)$ (praktisch $k = p$ prim oder $k = 2^n$) gilt nach dem Satz von Hasse [Silverman1986] die einfache Beziehung $||E|-k-1| \leq 2 \cdot \sqrt{k}$. Diese Gleichung ist gleichbedeutend mit $k+1-2\sqrt{k} < |E| < k+1+2\sqrt{k}$.Dies bedeutet, dass die Anzahl der Elemente der Elliptischen Kurve mit der Größe k gut abgeschätzt werden kann.

6.5 Verknüpfung auf Elliptischen Kurven

Um mit Elliptischen Kurven arbeiten zu können, definiert man eine Verknüpfung (meist additiv als + geschrieben) auf den Punkten der Elliptischen Kurve. Dabei definiert man bei Elliptischen Kurven über $GF(p)$ die kommutative Verknüpfung durch

1. $P + \mathcal{O} = \mathcal{O} + P = P$ für alle $P \in E$,

[119] Anschaulich bedeutet eine solche Koordinatentransformation eine Drehung bzw. Streckung der Koordinatenachsen, ohne dass die zugrunde liegende Kurve selbst verändert wird.

[120] Die Form (2) ist die Standardform der Weierstrass-Gleichung. Ist die Charakteristik des Körpers jedoch 2 oder 3, so ist $4 = 0$ bzw. $27 = 0$, was dazu führt, dass man in der Bedingung an die Parameter a, b wesentliche Informationen verliert. Dies ist ein Hinweis darauf, dass die Transformation auf die Standardform in diesen Fällen nicht zu befriedigenden Ergebnissen führt. Auf dieFür $p = 3$

[121] Formal sagt man, die Kurve ist nicht singulär.

2. für $P = (x, y)$ und $Q = (x, -y)$ ist $P + Q = \mathcal{O}$,

3. für $P_1 = (x_1, x_2), P_2 = (x_2, y_2) \in E$ mit $P_1, P_2 \neq \mathcal{O}$ und $(x_2, y_2) \neq (x_1, -y_1)$ ist $P_3 := P_1 + P_2$, $P_3 = (x_3, y_3)$ definiert durch

$$x_3 := -x_1 - x_2 + \lambda^2\,, \qquad y_3 := -y_1 + \lambda(x_1 - x_3)$$

mit dem Hilfsquotienten

$$\lambda := \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} & \text{falls } P_1 \neq P_2, \\ \frac{3x_1^2 + a}{2y_1} & \text{falls } P_1 = P_2. \end{cases}$$

Hieraus folgt insbesondere für $P = (x, y) \in E$, dass gilt $-P = (x, -y)$.
Über $GF(2^n)$ definiert man analog die Verknüpfung durch

1. $P + \mathcal{O} = \mathcal{O} + P = P$ für alle $P \in E$,

2. für $P = (x, y)$ und $Q = (x, x + y)$ ist $P + Q = \mathcal{O}$,

3. für $P_1 = (x_1, x_2), P_2 = (x_2, y_2) \in E$ mit $P_1, P_2 \neq \mathcal{O}$ und $(x_2, y_2) \neq (x_1, x_1 + y_1)$ ist $P_3 := P_1 + P_2$, $P_3 = (x_3, y_3)$ definiert durch

$$x_3 := -x_1 + x_2 + \lambda + \lambda^2 + a\,, \qquad y_3 := y_1 + x_3 + \lambda(x_1 + x_3)$$

mit

$$\lambda := \begin{cases} \frac{y_1 + y_2}{x_1 + x_2} & \text{falls } P_1 \neq P_2, \\ x_1 + \frac{y_1}{x_1} & \text{falls } P_1 = P_2. \end{cases}$$

Hieraus folgt insbesondere für $P = (x, y) \in E$, dass gilt $-P = (x, x + y)$. (Beachte: $-(-P) = (x, x + (x + y)) = (x, 2x + y) = (x, y)$, da der zugrunde liegende Körper Charakteristik 2 hat.)[122]

[122] Eine Animation der Punktaddition auf Elliptischen Kurven findet man auf der Certicom-Seite unter http://www.certicom.com/resources/ecc_tutorial/ecc_tutorial.html

Addieren von Punkten auf einer Elliptischen Kurve

Die zwei folgenden Abbildungen zeigen, wie bei einer Elliptischen Kurve über den reellen Zahlen in affinen Koordinaten zwei Punkte addiert werden. Der unendlich ferne Punkt $\mathcal{O}$ kann nicht in der affinen Ebene dargestellt werden.

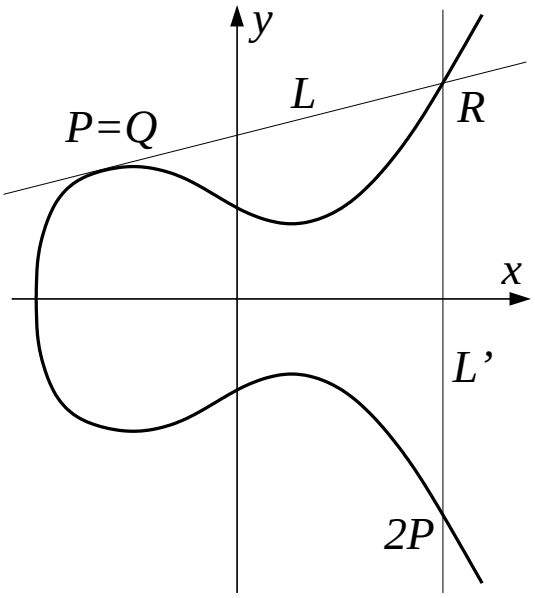


Abbildung 4: Verdoppelung eines Punktes

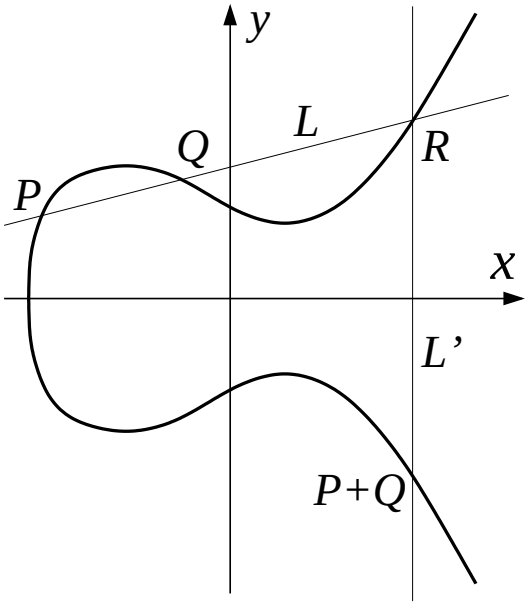


Abbildung 5: Addition zweier verschiedener Punkte im Körper der reellen Zahlen

6.6 Sicherheit der Elliptischen-Kurven-Kryptographie: Das ECDLP

Besonders interessant sind aus kryptographischer Sicht Vielfache eines gegebenen Punktes $P \in E$. So bezeichnet man für eine natürliche Zahl m

$$mP := \underbrace{P + P + \cdots + P}_{m \text{ mal}}.$$

Um mP zu berechnen, muss man im allgemeinen $\log m$ Additionen durchführen — man bildet einfach P, $2P$, 2^2P, 2^3P, ..., schreibt m binär und addiert schließlich entsprechend der Binärdarstellung von m auf. Andererseits erscheint es extrem schwierig zu sein, zu gegebenen Punkten P und $Q = mP$ auf E die Zahl m zu bestimmen. Natürlich kann man $P, 2P, 3P, 4P, 5P, \ldots$ bilden und jeweils mit Q vergleichen. Hierzu benötigt man jedoch m Additionen.

Bisher ist noch kein Algorithmus bekannt, der effizient m aus P und Q berechnet. Die bisher besten Verfahren liegen z.B. im Fall $GF(p)$ in der Größenordnung $\sqrt{q}$, wobei q ein (großer) Primfaktor von $p-1$ ist; m selbst sollte in diesem Fall zwischen 1 und q liegen, so dass man für die Muliplikation mP maximal $\log q$ Schritte benötigt. Der Quotient $\frac{\sqrt{q}}{\log q}$ strebt jedoch (schnell) gegen $+\infty$.

Sind die Parameter hinreichend groß (ist zum Beispiel p prim und mehr als 160 Bit lang) ist der Computer ohne weiteres in der Lage, sehr schnell (in wenigen Bruchteilen einer Sekunden) den Punkt mP zu bestimmen. Das *inverse Problem*, m aus mP und P zu erhalten, ist jedoch nicht in akzeptabler Zeit möglich.

Dies wird als das „Diskrete Logarithmus Problem über Elliptischen Kurven“ bezeichnet (auch ECDLP - Elliptic Curve Discrete Logarithm Problem - abgekürzt).

Es ist zu beachten, dass nicht alle Elliptischen Kurven gleich sicher sind. Das bedeutet, dass man bei der Definition einer Kurve auf die Wahl der Parameter achten muss. Denn für bestimmte Klassen von Elliptischen Kurven ist es möglich, das ECDLP leichter zu lösen als im allgemeinen Fall. Kryptographisch ungeeignete Elliptische Kurven sind die sogenannten *anormalen* Kurven (das sind Kurven über $\mathbb{Z}_p$, für die die Menge $\mathbf{E}$ genau p Elemente hat) und die *supersingulären* Kurven (das sind Kurven, für die man das Berechnen des ECDLP auf das Berechnen des „normalen“ Diskreten Logarithmus in anderen endlichen Körper reduzieren, d.h. vereinfachen, kann). Daher gibt es kryptographisch gute und schlechte Kurven. Allerdings kann man für gegebene Parameter a und b mit etwas Aufwand feststellen, ob die resultierende Elliptische Kurve kryptographisch brauchbar ist oder nicht. Die in der Kryptographie eingesetzten Kurven werden meist von Fachleuten zur Verfügung gestellt. Sie gewährleisten, dass die von ihnen als sicher eingestuften Elliptischen Kurven den aktuellen Sicherheitsanforderungen genügen.

Bei sicheren Kurven wird hauptsächlich durch den Parameter p im Fall des zugrunde liegenden Körpers $GF(p)$ bzw. n im Fall des zugrunde liegenden Körpers $GF(2^n)$ bestimmt, wie lange es dauert, das ECDLP auf dieser Kurve zu lösen. Je größer diese Parameter sind, desto länger nimmt das Lösen des Problems in Anspruch. Von Fachleuten wird z.B. eine Bitlänge von über 200 Bit für den Parameter p empfohlen. Hier wird deutlich, warum die Elliptischen Kurven so interessant für die Kryptographie sind. Denn die Parameter bestimmen auch den Signatur-/Verschlüsselungsaufwand, wenn mit Elliptischen Kurven Kryptographie betrieben wird. Die Dauer einer

Schlüsselpaar-Erzeugung ist ebenfalls von den Parametern abhängig. Daher sind kleine Werte (wenige Bits) wünschenswert (möglichst schnelle Laufzeiten der Verfahren); allerdings muss die geforderte Sicherheit dabei eingehalten werden. Mit einer Länge von zum Beispiel 200 Bit für p ist eine *gute* Elliptische Kurve genau so sicher wie ein RSA-Modulus von über 1024 Bit Länge (zumindest nach dem heutigen Forschungstand). Der Grund dafür ist, dass die schnellsten Algorithmen zum Lösen des *Elliptische Kurven Diskreter Logarithmus* Problems eine exponentielle Laufzeit haben — im Gegensatz zu den subexponentiellen Laufzeiten, die die zur Zeit besten Faktorisierungsalgorithmen haben (Zahlkörpersieb, Quadratisches Sieb oder Faktorisieren mit Elliptischen Kurven). Dies erklärt, warum die Parameter von Kryptoverfahren, die auf dem Problem *Faktorisieren von ganzen Zahlen* beruhen, größer sind als die Parameter von Kryptoverfahren, die auf dem ECDL-Problem basieren.

6.7 Verschlüsseln und Signieren mit Hilfe Elliptischer Kurven

Das *Elliptische Kurven Diskreter Logarithmus Problem* (ECDLP) ist die Grundlage für die Elliptische-Kurven-Kryptographie. Darauf basierend gibt es verschiedene Signaturverfahren. Um ein solches Signaturverfahren anzuwenden, benötigt man:

- Eine Elliptischen Kurve $\mathbf{E}$, beschrieben durch den zugrunde liegenden Körper $GF(p^n)$.
- Eine Primzahl $q \neq p$ sowie einen Punkt G auf der Elliptischen Kurve $\mathbf{E}$ mit Ordnung q. D.h., es gilt $qG = \mathcal{O}$ und $rG \neq \mathcal{O}$ für alle $r \in \{1, 2, \ldots, q-1\}$. Die Zahl q muss dann ein Teiler der Gruppenordnung (entspricht der Anzahl der Elemente) $\#\mathbf{E}$ sein. Aufgrund der Primordnung, erzeugt G eine zyklischen Untergruppe von $\mathbf{E}$ mit Ordnung q.

Die genannten Parameter bezeichnet man als *Domain*-Parameter. Durch sie wird festgelegt, auf welcher Elliptischen Kurve $\mathbf{E}$ und in welcher zyklischen Untergruppe von $\mathbf{E}$ ein Signaturverfahren eingesetzt wurde.

Verschlüsselung:

Mit Hilfe Elliptischer Kurven kann ein sicherer Schlüsselaustausch nach dem DH-KeyExch erfolgen (siehe Kapitel 4.4.2). Dieser Schlüssel kann dann für eine anschließende symmetrische Verschlüsselung verwendet werden.

In der Schreibweise der Elliptischen Kurven ließt sich das Diffie-Hellman Verfahren wie folgt: Zunächst einigen sich beide Partner (A und B) öffentlich auf eine Gruppe G und ein Element q. Danach wählen sie zufällig $r_A, r_B \in \{1, 2, \ldots, q-1\}$, bilden die Punkte $R_A = r_A G$, $R_B = r_B G$ auf der Elliptischen Kurve und tauschen diese aus. Danach berechnet A leicht $R = r_A R_B$. Denselben Punkt $r_A r_B G$ erhält auch B, indem er $r_B R_A = r_B r_A G = r_A r_B G = R$ bildet.
Für einen Dritten ist es nach heutigen Kenntnisstand nicht möglich, R zu berechnen, wenn er nicht mindestens einen der Werte r_A oder r_B ermitteln kann, d.h. das ECDLP löst.

Um einen "Man-in-the-middleÄngriff zu verhindern, kann man auch hier wie schon in Kapitel 5.3.1 beschrieben, die übertragenen Werte G, q, R_A, R_B digital signieren.

Signatur-Erstellung:

Überträgt man den DSA auf Elliptische Kurve, so kann man wie folgt eine digitale Signatur erzeugen: Man wählt vorab eine (nicht-triviale) Zahl $s \in \mathbb{Z}_q$. Diese bildet den privaten Schlüssel. Hingegen werden q, G und $R = sG$ veröffentlicht. Aus G und R lässt sich jedoch s nicht ermitteln, worauf die Sicherheit des Signaturverfahrens beruht.

Eine Nachricht m wird zunächst mit Hilfe eines Hash-Verfahrens h ein digitaler Fingerabdruck erstellt, wobei $h(m)$ im Wertebereich $\{0, 1, 2, \ldots, q-1\}$ liegt und $h(m)$ somit als Element von $\mathbb{Z}_q$ interpretiert werden kann. Dann wird ein zufälliges $r \in \mathbb{Z}_q$ gewählt und $R = (r_1, r_2) = rG$ berechnet. Die erste Komponente r_1 von R ist ein Element von $GF(p^n)$. Diese wird auf $\mathbb{Z}_q$ abgebildet, z.B. im Fall $n = 1$ als Element von $\{0, 1, \ldots, p-1\}$ interpretiert und dann der Teilerrest modulo q gebildet. Das so erhaltene Element von $\mathbb{Z}_q$ bezeichnen wir mit $\bar{r}_1$. Nun bestimmt man $x \in \mathbb{Z}_q$ mit

$$rx - s\bar{r}_1 - h(m) = 0.$$

Das Tripel (m, r_1, x) bildet nun die digitale Signatur.

Signatur-Verifikation:

Zur Verifikation muss zunächst $u_1 = h(m)/x$, $u_2 = \bar{r}_1/x$ (in $\mathbb{Z}_q$ gebildet werden). Dann bestimmt man

$$V = u_1 G + u_2 Q.$$

Wegen $Q = sG$ ist $V = (v_1, v_2)$ mit $v_1 = u_1 + u_2 s$. Diese Addition findet formal im Raum $GF(p^n)$ statt. Die Projektion von $GF(p^n)$ auf $\mathbb{Z}_q$ sollte jedoch so gewählt sein, dass $\bar{v}_1 = u_1 + u_2 s$ in $\mathbb{Z}_q$ ist. Dann gilt nämlich

$$\bar{v}_1 = u_1 + u_2 s = h(m)/x + \bar{r}_1 s/x = (h(m) + \bar{r}_1 s)/x = rx/x = r.$$

Nun ist $R = rG$. Also folgt hier $\bar{v}_1 = \bar{r}_1$, d.h. R und V stimmen modulo der Projektion auf $\mathbb{Z}_q$ überein.

6.8 Faktorisieren mit Elliptischen Kurven

Es gibt Faktorisierungsalgorithmen, die auf Elliptischen Kurven basieren[123]. Genauer gesagt, machen sich diese Verfahren zunutze, dass man auch über $\mathbb{Z}_n$ (n zusammengesetzte Zahl) Elliptische Kurven definieren kann. Elliptische Kurven über $\mathbb{Z}_n$ bilden keine Gruppe, da es nicht zu jedem Punkt auf solchen Elliptischen Kurven einen inversen Punkt geben muss. Dies hängt damit zusammen, dass es – falls n eine zusammengesetzte Zahl ist – in $\mathbb{Z}_n$ Elemente gibt, die kein Inverses bezüglich der Multiplikation modulo n haben. Um zwei Punkte auf einer Elliptischen Kurve über $\mathbb{Z}_n$ zu addieren, kann prinzipiell genauso gerechnet werden wie auf Elliptischen Kurven über $\mathbb{Z}_p$. Eine Addition von zwei Punkten (auf einer Elliptischen Kurve über $\mathbb{Z}_n$) scheitert aber genau dann, wenn man einen Teiler von n gefunden hat. Der Grund dafür ist, dass das Verfahren zum Addieren von Punkten auf Elliptischen Kurven Elemente in $\mathbb{Z}_n$ ermittelt und zu diesen Elementen die inversen Elemente (bezüglich der Multiplikation modulo n) in $\mathbb{Z}_n$ berechnet. Dazu wird

[123] Im Jahr 1987 stellte H.W. Lenstra einen Faktorisierungsalgorithmus vor, der auf Elliptischen Kurven basiert (siehe [Lenstra1987]). Die aktuell größte mit Elliptischen Kurven faktorisierte Zahl ist die 55 Dezimalstellen lange Zahl $629^{59} - 1$, sie wurde am 6. Oktober 2001 von M. Izumi gefunden (siehe ECMNET).

der erweiterte Euklidsche Algorithmus benutzt. Ergibt sich nun bei der Addition zweier Punkte (die auf einer Elliptischen Kurve über $\mathbb{Z}_n$ liegen) ein Element aus $\mathbb{Z}_n$, das kein inverses Element in $\mathbb{Z}_n$ hat, so gibt der erweiterte Euklidsche Algorithmus einen echten Teiler von n aus.

Das Faktorisieren mit Elliptischen Kurven funktioniert somit prinzipiell so: Man wählt zufällige Kurven über $\mathbb{Z}_n$, sowie zufällig irgendwelche Punkte (die auf diesen Kurve liegen) und addiert diese; dabei bekommt man wieder Punkte, die auf der Kurve liegen oder findet einen Teiler von n. Die Faktorisierungsalgorithmen auf Basis von Elliptischen Kurven arbeiten also probabilistisch. Durch die Möglichkeit, sehr viele Elliptische Kurven über $\mathbb{Z}_n$ zu definieren, kann man die Wahrscheinlichkeit erhöhen, zwei Punkte zu finden, bei deren Addition ein Teiler von n gefunden wird. Daher eignen sich diese Verfahren auch sehr gut für eine Parallelisierung.

6.9 Implementierung Elliptischer Kurven

CrypTool benutzt Elliptische Kurven bei der Funktion für digitale Signaturen.

Implementiert sind die Basisalgorithmen für Gruppenoperationen, für das Erzeugen von Elliptischen Kurven und für das Ein- und Auslesen von Parametern für Elliptische Kurven über endlichen Körpern $GF(p)$ mit p (p prim) Elementen. Die Implementierung erfolgte in ANSI C und richtete sich nach dem Entwurf Nr.8 der Arbeitsgruppe IEEE P1363 *Standard Specifications for Public Key Cryptography*

http://grouper.ieee.org/groups/1363.

Implementiert sind die kryptographischen Primitive zur Signaturerzeugung und Signaturverifikation für die auf Elliptischen Kurven basierenden Varianten von Nyberg-Rueppel-Signaturen und DSA-Signaturen (nach Entwurf Nr.8 der Arbeitsgruppe IEEE P1363). Dies erfolgte in Zusammenarbeit mit der Secude GmbH — und unter Benutzung der obigen Bibliothek und des Secude SDK.

Wenn man statt des Primkörpers $GF(p)$ einen Körper der Form $GF(2^n)$ zugrunde legt, ergeben sich wesentliche Unterschiede in der Implementierung. Der Vorteil einer Implementierung unter Verwendung von $GF(2^n)$ liegt darin, dass Rechnungen aufgrund der binären Darstellung effizienter durchgeführt werden können. Dies betrifft insbesondere die Division, die in $GF(p)$ vergleichsweise aufwändig ist (was z.B. bei dem oben beschriebenen Signaturverfahren sowohl die Erstellung der Signatur als auch ihre spätere Verifikation betrifft, da beide eine Divisionsoperation enthalten).

Um das Potenzial der Effizienzsteigerung möglichst optimal zu nutzen, kann man z.B. Körper wählen, die besondere Basen besitzen, wie Polynomialbasen (besonders geeignet für Software-Implementierungen) oder Normalbasen (bevorzugt bei Hardware-Implementierungen). Für bestimmte Werte von n (wie z.B. $n = 163, 179, 181$) lassen sich sogar beide Vorteile kombinieren. Allerdings sind spezielle Darstellungen oft nicht Standard.

Um Speicherplatz zu sparen, wird zuweilen nur die erste Komponente sowie ein weiteres Bit für jeden Punkt auf der Elliptischen Kurve gespeichert. Hieraus kann jeweils der gesamte Punkt errechnet werden. Dies ist besonders bei Verwendung von Normalbasen effizient. Selbst bei der Durchführung der kryptographischen Protokolle kann eine signifikante Beschleunigung erreicht

werden. Diese sogenannte *Punkt-Kompression*, die bei der Hälfte aller Kurven einsetzbar ist, ist jedoch patentiert (US Patent 6141420, Certicon) und daher nicht ohne weiteres frei einsetzbar.

Im allgemeinen Fall $GF(p^n)$ (aber auch für $n = 1$) werden oft sogenannte affine oder projektive Koordinaten eingesetzt, was je nach Anwendung zu Effizienzgewinnen führt.

Eine vollständige Beschreibung aller Implementierungen unter Abwägung ihrer Vor- und Nachteile würde an dieser Stelle zu weit führen. Abschließend kann festgehalten werden, dass die Vielfalt möglicher Implementierungen bei Elliptischen Kurven z.B. im Vergleich zu RSA-Implementierungen sehr groß ist. Aus diesem Grund gibt es Bestrebungen, sich auf wenige Standardimplementierungen, ja sogar auf eine kleine Schar fest vorgegebener Kurven zu beschränken (ASC-Ansatz).

Der Erfolg dieser Standardisierungsbemühungen ist heute noch nicht absehbar. Dies wäre aber eine Voraussetzung dafür, dass sich ECC dauerhaft als Alternative zu RSA etabliert.

Literatur

[Lenstra1987] H.W. Lenstra
Factoring integers with elliptic curves, Annals of Mathematics 126, pp. 649-673, 1987.

[Lenstra1999] Arjen K. Lenstra, Eric R. Verheul
Selecting Cryptographic Key Sizes (1999), Journal of Cryptology: the journal of the International Association for Cryptologic Research
cryptosavvy.com/cryptosizes.pdf

[Silverman1986] J. Silverman
The Arithmetic of Elliptic Curves, Springer-Verlag, 1986.

Bücher über Elliptische Kurven (Auswahl)

[Cas] J. W. S. Cassels, *Lectures on elliptic curves*, Cambridge University Press, 1991, 143 Seiten.

[Kob] N. Koblitz, *Introduction to elliptic curves and modular forms*, Graduate Texts in Mathemathics, Springer-Verlag, 1984.

[Kob] N. Koblitz, *Algebraic aspects of Cryptography. With an appendix on Hyperelleptic curves by Alfred J. Menezes, Yi Hong Wu and Robert J. Zuccherato*, Springer, 1998, 206 Seiten.

[Men] A. J, Menezes, *Elliptic curve public key cryptosystems*, Kluwer Academic Publishers, 1993.

[Sil] J. Silverman, *The arithmetic of elliptc curves*, Graduate Texts in Mathemathics, Springer-Verlag, 1992.

[ST] J. Silverman, J. Tate, *Rational points on elliptic curves*, Springer-Verlag, 1992.

Web-Links

1. Online-Tutorial über Elliptische Kurven der Firma Certicom
   http://www.certicom.com/resources/ecc_tutorial/ecc_tutorial.html
2. Arbeitsgruppe IEEE P1363
   http://grouper.ieee.org/groups/1363
3. Eine informative Seite zum Faktorisieren mit Elliptischen Kurven.
   http://www.loria.fr/~zimmerma/records/ecmnet.html
   Dort findet man Literatur zum Thema Faktorisieren mit Elliptischen Kurven sowie Links zu anderen ECC-Seiten.
4. Schlüssellängen-Vergleich von Arjen Lenstra und Eric Verheul
   http://cryptosavvy.com/table.htm

A CrypTool-Menüs

Dieser Anhang enthält den kompletten Menü-Baum von CrypTool. Bitte beachten Sie, dass nicht immer alle Menüeinträge gleichzeitig verfügbar sind, weil der Menübaum dynamisch aufgebaut wird. Welche Menüeinträge gerade angezeigt werden, bestimmt das aktive Dokumentenfenster. So ist z. B. die Brute-Force Analyse für DES nur verfügbar, wenn das aktive Fenster in Hexadezimal-Darstellung geöffnet ist, während der Menüeintrag „Zufallsdaten erzeugen. . . “ immer verfügbar ist. In den Abbildungen 6 und 7 ist dieser Zusammenhang hinter den Menüeinträgen in eckigen Klammern dargestellt, z. B. bedeutet „Suchen. . . [ATH]“, dass die Funktion „Suchen. . . “ nur für ASC-, Text- und Hexadezimal-Fenster verfügbar ist.

Codebuchstabe	**Dokumententyp**
M	(kein Dokument geöffnet)
A	ASC
T	Text
H	Hexadezimal
P	Plot

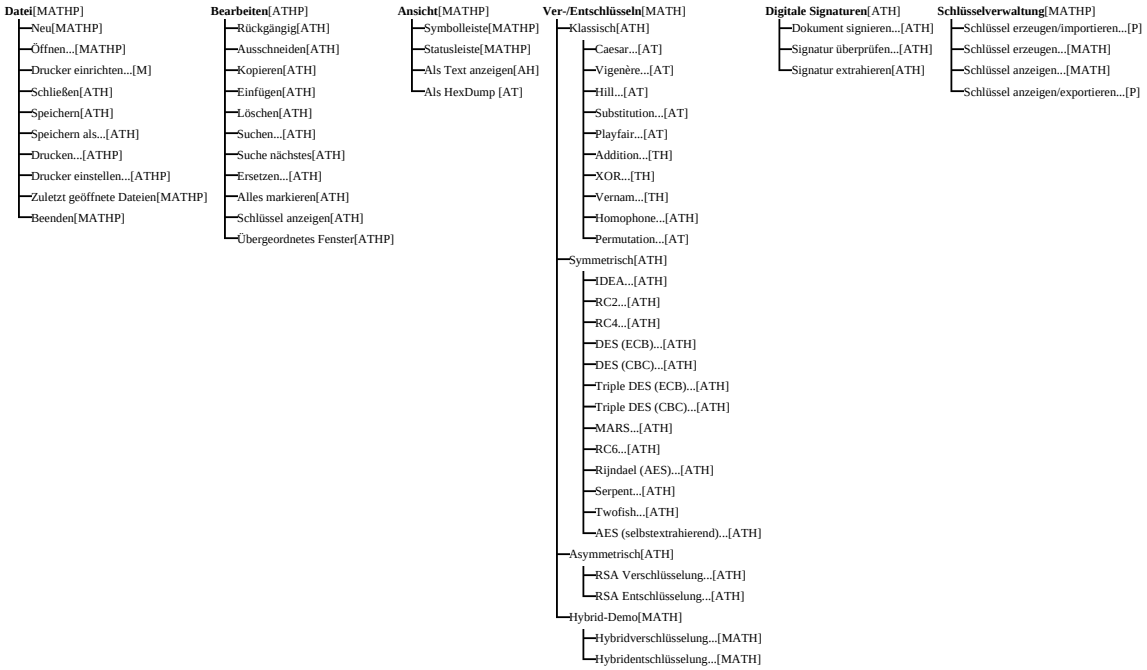


Abbildung 6: Detaillierte Ansicht des CrypTool-Menü-Baums – Teil 1

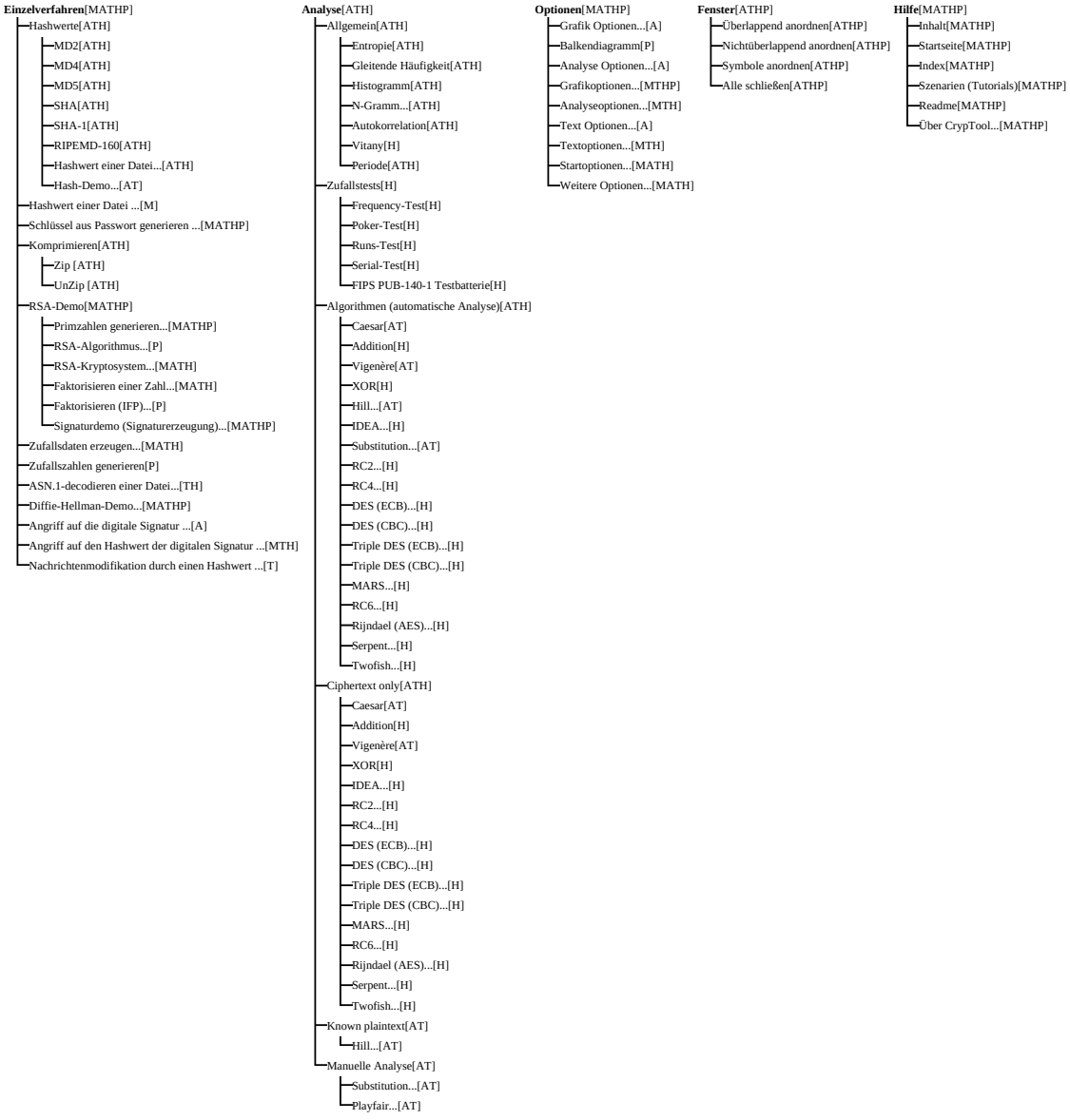


Abbildung 7: Detaillierte Ansicht des CrypTool-Menü-Baums – Teil 2

Abbildung 8: Komplette Übersicht des CrypTool-Menü-Baums

Index