Public Key Kryptographie

Vorlesungsskript
WS 2006/2007

Willi Geiselmann
Jens-Matthias Bohli
Stefan Röhrich

Version: 2006-11-15

Institut für Algorithmen und Kognitive Systeme / E.I.S.S.
Universität Karlsruhe

Vorabversion[1]

[1]Korrektur- und Ergänzungsvorschläge werden mit Interesse entgegengenommen

Inhaltsverzeichnis

Inhaltsverzeichnis **i**

Einführung, Terminologie **1**

Terminologie . . . 1
Aufgaben der Kryptographie . . . 2
Algorithmen und Schlüssel . . . 2
Algorithmen ohne Schlüssel . . . 2
Algorithmen mit Schlüssel . . . 3
Kryptoanalyse . . . 3
Sicherheit von Algorithemen . . . 4
Aufwand von Angriffen . . . 4

1 Protokolle **5**

1.1 Einführung . . . 5
1.2 Protokoll mit Notar . . . 6
1.3 Typen von Angriffen gegen Protokolle . . . 7
1.4 Eine erste Public-Key-Idee (Merkles Puzzle) . . . 7
1.5 Bausteine für Protokolle . . . 8
1.5.1 Einwegfunktionen . . . 8
1.5.2 Hashfunktionen . . . 9
1.6 Protokolle für Schlüsselaustausch . . . 11
1.6.1 Protokoll mit einem symmetrischen Verfahren und einer Schlüsselzentrale (SZ) . . . 12
1.6.2 Protokoll mit einem Public-Key-Verfahren . . . 12

2 Public-Key-Algorithmen **16**
2.1 Knapsack . . . 16
2.1.1 Verfahren . . . 17
2.1.2 Angriff auf das Knapsackverfahren . . . 18
2.2 RSA-Verfahren . . . 19
2.2.1 Verfahren (Textbook-RSA) . . . 19
2.2.2 Primzahlerzeugung . . . 20
2.2.3 Angriffe auf Protokollebene . . . 21
2.3 ElGamal-Verfahren . . . 22
2.4 McEliece-Kryptosystem . . . 23
2.5 Beweisbar sichere Public-Key-Systeme . . . 25
2.5.1 Public-Key-Kryptosysteme . . . 26

3 Probabilistische Primtests **29**
3.1 Primzahlen . . . 29
3.2 Quadratische Reste . . . 34
3.3 Pseudoprimzahlen . . . 38
3.4 Probabilistischer Primtest nach Solovay, Strassen . . . 40
3.5 Probabilistischer Primtest nach Rabin und Miller . . . 43

4 Faktorisierungsalgorithmen **47**
4.1 Faktorisierungsalgorithmen mit Gruppen . . . 47
4.1.1 Pollard $p-1$-Methode . . . 47
4.1.2 Faktorisierung mit elliptischen Kurven . . . 49
4.2 Faktorisierung mit quadratischen Kongruenzen . . . 53
4.2.1 Grundlagen der Laufzeitanalyse . . . 54
4.2.2 Dixons Algorithmus . . . 55
4.2.3 Quadratischer Siebalgorithmus . . . 57

5 Diskrete Logarithmen **61**
5.1 Einführung . . . 61
5.2 Baby-Step-Giant-Step-Algorithmus . . . 61
5.3 Pohlig-Hellman-Algorithmus . . . 62
5.4 Index Calculus . . . 63

5.4.1 Endliche Körper . . . 64
5.4.2 Index-Calculus für $\mathbb{F}_{p^n}$ . . . 66
5.4.3 Index-Calculus für $\mathbb{F}_p$ . . . 68

6 Digitale Signatur 69
6.1 Einführung . . . 69
6.1.1 Signatur mit Hilfe eines Notars und eines symm. Kryptosystems . . . 69
6.1.2 Signatur mit Public-Key-Verfahren . . . 70
6.1.3 Signatur mit Hashfunktionen . . . 70
6.2 Signieren mit RSA . . . 71
6.2.1 Blinde Signaturen . . . 71
6.3 Public-Key-Signatursysteme . . . 72
6.3.1 Unterschiedliche Sicherheitsstufen . . . 72
6.3.2 Definition . . . 73
6.4 ElGamal-Signaturverfahren . . . 75
6.5 Signieren mit elliptischen Kurven . . . 76
6.6 Ist EUF-CMA sicher genug? . . . 77
6.6.1 Key-Substitution-Angriffe . . . 78
6.6.2 Subliminale Kanäle . . . 80

7 Schlüsselaustausch, Authentifikation, . . . 82
7.1 Authentifikation mit Einwegfunktionen . . . 82
7.2 Authentifikation mit Public-Key-Verfahren . . . 83
7.3 Schlüsselaustausch . . . 84
7.3.1 Wide-mouth frog . . . 84
7.3.2 Needham-Schroeder . . . 85
7.3.3 Kerberos . . . 85
7.3.4 Neuman-Stubblebine . . . 86
7.4 Ein Modell für beweisbare Sicherheit . . . 86
7.5 Public-Key-Verfahren zum Schlüsselaustausch . . . 88
7.5.1 Diffie-Hellman-Verfahren . . . 88
7.5.2 Diffie-Hellman mit Signaturen . . . 88
7.5.3 MTI-Protokoll . . . 89
7.5.4 Transport Layer Security (TLS) . . . 90

7.5.5 Secure Shell (SSH) 91
7.6 Zero-Knowledge-Protokolle 92
7.7 Digital Cash 97
7.7.1 Erste Protokolle 97
7.7.2 Hilfsprotokolle 100
7.7.3 Ein vollständiges Protokoll 101

Literaturverzeichnis **102**

Index **103**

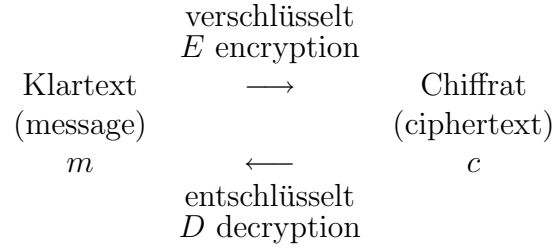
Einführung, Terminologie

Terminologie

Das kleinstmögliche Modell in der Kryptographie hat folgende Form:

- Sender Alice, Empfänger Bob
- Nachricht oder Klartext m, Chiffrat c
- Verschlüsselung E (encryption), Entschlüsselung D (decryption)

Alice will Bob eine Nachricht m schicken. Alice will erreichen, daß ihre Nachricht sicher übertragen wird, d. h. ein Unbefugter darf die Nachricht nicht lesen oder verändern können. Um dies zu erreichen, muß die Nachricht von Alice verschlüsselt werden, so daß sie nur von Bob entschlüsselt werden kann.



Mathematisch kann der Sachverhalt vereinfacht wie folgt beschrieben werden. E ist ein Algorithmus, der zu Eingaben aus der Menge der Klartexte Ausgaben aus der Menge der Chiffrate liefert, D entsprechend ein Algorithmus in der entgegengesetzten Richtung. Dabei sollen folgende Eigenschaften gelten:

$$\begin{aligned} E(m) &= c \\ D(c) &= m \\ D(E(m)) &= m \\ (\ E(D(c)) &= c\ ). \end{aligned}$$

Die letzte Gleichung ist nicht zwingend gefordert, oft aber auch erfüllt. Bei einigen Verschlüsselungen geht aber außer dem Klartext auch noch ein Zufallsstring mit ein, so daß derselbe Klartext auf verschiedene Chiffrate führen kann; in diesem Fall handelt es sich bei E um keine Funktion von den Klartexten in die Chiffrate und die Gleichung $E(D(c)) = c$ gilt hier nicht.

Aufgaben der Kryptographie

Kryptographie oder auch *Kryptologie* (aus dem Griechischen κρυπτός (kryptós, „versteckt“) und λόγος (lógos, „Wort“) bzw. γράφειν (gráphein, „schreiben“)) ist die Wissenschaft, die sich mit der sicheren (geheimen) Kommunikation befaßt. Die Vertraulichkeit ist nur eine der Aufgaben, die die Kryptographie heute erfüllen muß. Üblicherweise werden die folgenden vier Eigenschaften betrachtet:

- **Vertraulichkeit:**
  Geheimhaltung der Nachricht.
- **Integrität:**
  Empfänger kann verifizieren, daß die Nachricht nicht verändert wurde.
- **Authentizität:**
  eindeutige Herkunft, Nachricht ist wirklich vom Sender.
- **Verbindlichkeit:**
  Unabstreitbarkeit, Sender kann die Herkunft nicht ableugnen.

Algorithmen und Schlüssel

Ein *kryptographischer Algorithmus* ist ein Verfahren zur Ver- und Entschlüsselung. (Im allgemeinen gibt es zwei verschiedene Algorithmen zur Ver- und Entschlüsselung.)

Algorithmen ohne Schlüssel

- Sicherheit beruht auf der Geheimhaltung des Algorithmus.
- Der Algorithmus kann schlecht auf Güte überprüft werden, falls kein kompetentes Mitglied in der Gruppe vorhanden ist, das den Algorithmus benutzt.
- Personen in mehreren Gruppen benötigen mehrere Algorithmen.
- Nach einem erfolgreichen Angriff oder nachdem ein Mitglied die Gruppe verlassen hat, muß der Algorithmus vollständig geändert werden.

Algorithmen mit Schlüssel

Es wird ein Algorithmus verwendet, der von Kryptographen geprüft oder standardisiert werden kann. Die Sicherheit wird durch die Wahl eines geheimen Schlüssels erreicht. Es gibt zwei Klassen von Algorithmen mit Schlüssel.

1. **Symmetrische Algorithmen**
   Die Schlüssel zur Ver- und Entschlüsselung sind einfach auseinander berechenbar, meistens sind sie identisch. Sender und Empfänger einigen sich vor dem Senden von Nachrichten auf den (die) Schlüssel.

   - **Stromchiffren:** Nachrichten werden z. B. bit- oder byteweise verarbeitet. Das Chiffrat eines Zeichens hängt von den vorigen Zeichen bzw. von der Stelle im Text ab.
   - **Blockchiffren:** Nachrichten werden blockweise (64 bit, 128 bit) verarbeitet. Ein Klartextblock wird unabhängig von der Stelle seines Auftretens auf ein Chiffrat abgebildet.

2. **Asymmetrische Algorithmen** (oder Public-Key-Algorithmen)
   Es werden unterschiedliche Schlüssel für Ver- und Entschlüsselung verwendet. Der Schlüssel zur Entschlüsselung läßt sich nicht mit vertretbarem Aufwand aus dem Schlüssel zum Verschlüsseln berechnen. Der Schlüssel zum Verschlüsseln heißt *öffentlicher Schlüssel*, der andere *geheimer Schlüssel*.

Kryptoanalyse

Das Ziel der *Kryptoanalyse* ist es, aus einem Chiffrat c die zugehörige Nachricht m ohne Kenntnis des geheimen Schlüssels zu gewinnen oder den geheimen Schlüssel zu finden. Die Grundannahme ist dabei, daß der Algorithmus bekannt ist. Es gibt vier Typen von Angriffen:

1. **Ciphertext-only**
   Geg.: Einige Chiffrate, die mit demselben Schlüssel und Algorithmus verschlüsselt wurden.
   Ziel: Klartexte und/oder Schlüssel zu finden.

2. **Known-plaintext**
   Geg.: Paare von Klartext und Chiffrat.
   Ziel: Finden des Schlüssels (um später unbekannte Klartexte zu finden).

3. **Chosen-plaintext**
   Geg.: Klartexte können frei gewählt werden, deren Chiffrate dann berechnet werden.
   Ziel: Schlüssel zu finden.

4. **Chosen-ciphertext**
   Geg.: Chiffrate können frei gewählt werden, die dann entschlüsselt werden (bei Public-Key-Verfahren sinnvoll).
   Ziel: Schlüssel (geheimen Schlüssel bei PK) zu finden.

Alle Kryptosysteme können durch einen Known-plaintext-Angriff gebrochen werden, indem man alle möglichen Schlüssel ausprobiert. Diese Art, den Schlüssel aus einem Klartext-Chiffrat-Paar zu gewinnen, heißt *Brute-Force-Angriff*. Oft ist dieser Angriff wegen der Redundanz in der Nachricht (Sprache, bekannter Anfang, Anrede usw.) auch möglich, wenn nur Chiffrate gegeben sind. Die Kryptographie sucht Algorithmen, die mit den zur Verfügung stehenden Mitteln (heutige oder zukünftige) nicht gebrochen werden können.

Sicherheit von Algorithmen

Es gibt vier Kategorien, die das Versagen eines kryptographischen Algorithmus beschreiben.

1. **Vollständiger Zusammenbruch**
   Der (geheime) Schlüssel wurde gefunden.

2. **Globale Entschlüsselung** (global deduction)
   Es wurde ein anderer Algorithmus zum Entschlüsseln gefunden, ohne den Schlüssel zu kennen.

3. **Lokale Entschlüsselung**
   Es wurden von einigen Chiffraten die Klartexte gefunden.

4. **Informationsgewinn**
   Teile oder Information über den Klartext und/oder den Schlüssel wurden gefunden.

Aufwand von Angriffen

Der Aufwand der Angriffe kann auf drei verschiedene Weisen gemessen werden:

1. **Rechenkapazität:**
   Zeit bzw. Anzahl der Rechenschritte.

2. **Speicherkomplexität:**
   benötigter Speicher.

3. **Datenkomplexität:**
   benötigte Anzahl an Chiffraten bzw. Chiffrat-Klartext-Paaren.

Kapitel 1

Protokolle

1.1 Einführung

Ein Protokoll besteht aus einer Reihe von Schritten, die nacheinander ausgeführt werden. Durch das Protokoll soll ein gewisser Zweck erfüllt werden. Weiterhin müssen noch folgende Bedingungen erfüllt sein:

- Das Protokoll ist jedem Teilnehmer bekannt.
- Jeder Teilnehmer ist mit dem Protokoll einverstanden.
- Der Ablauf des Protokolls muß eindeutig sein.
- Das Protokoll muß vollständig sein (es ist in allen Fällen genau bestimmt, was zu tun ist).

Das Ziel eines kryptographischen Protokolls ist es, Abhören und Betrügen zu verhindern. Außerdem darf kein Teilnehmer mehr erfahren können als das, was im Protokoll vorgesehen ist.

Beispiel 1.1. Informelle Protokolle aus dem täglichen Leben:

- Telefonische Bestellung von Gütern
- Bezahlen mit Kreditkarte
- Wahlen
- Kartenspiele

Diese Protokolle funktionieren meistens, da man die Teilnehmer kennt, sieht oder hört. Computer brauchen im Gegensatz zu Menschen formale Protokolle.

1.2 Protokoll mit Notar

Ein *Notar* ist eine vertrauenswürdige Instanz, die keinen der Teilnehmer bevorzugt. Mit Hilfe einer solchen Instanz lassen sich recht einfach Protokolle angeben.

Beispiel 1.2 (Verkauf eines Autos mit Hilfe eines Notars). Der Verkauf kann wie folgt abgewickelt werden:

1. Alice will Bob ein Auto verkaufen.
2. Bob will mit einem Scheck bezahlen.
3. Alice gibt den KFZ-Brief dem Notar.
4. Bob gibt den Scheck an Alice, die ihn dann bei der Bank einlöst.
5. Falls der Scheck gedeckt war, gibt der Notar den KFZ-Brief an Bob.
6. Falls aber der Scheck nicht gedeckt war, muß Alice dies dem Notar beweisen und erhält daraufhin den KFZ-Brief vom Notar zurück.

Eine zweite Lösung wäre ein von der Bank zertifizierter Scheck. In diesem Fall könnte man ohne Notar auskommen.

Das obige Beispiel läßt sich auf Rechnernetzwerke übertragen. Ein wesentlicher Nachteil ist es, daß bei jeder Transaktion der Notar involviert ist. Dieser Engpaß kann durch den Einsatz mehrerer Notare behoben werden, was aber die Kosten steigert. Eine andere Möglichkeit, diesen Engpaß zu beheben, ist es, das Protokoll wie folgt in zwei Teile zu zerlegen:

- **Normalfall**
  - Alice und Bob handeln einen Vertrag aus,
  - Alice unterschreibt,
  - Bob unterschreibt.
- **Ausnahmefall** (Vertrag wird nicht eingehalten)
  - Alice und Bob erscheinen vor Gericht,
  - Alice und Bob erläutern den Fall,
  - Richter fällt eine Entscheidung.

Dieses Protokoll verhindert nicht einen Betrug, hat aber einen *Abschreckungseffekt*.

Meist ist es nicht möglich, das Betrügen und eventuelle Fehlfunktionen zu unterscheiden, deshalb ist meistens das Beste, was man erreichen kann, ein *abgesichertes Protokoll*. Dieses Protokoll garantiert, daß keiner der Teilnehmer betrügen kann. Sobald sich ein Partner nicht an das Protokoll hält (Fehler oder Betrug), bemerkt das der andere Teilnehmer und bricht ab.

1.3 Typen von Angriffen gegen Protokolle

- **passiver Angriff:**
  Ein Unbeteiligter hört die Nachrichten ab und versucht, daraus Information zu gewinnen.

- **aktiver Angriff:**
  Ein Unbeteiligter ändert die Nachrichten (fängt beispielsweise Nachrichten ab und schickt andere).

- **passiver Betrug:**
  Ein Teilnehmer des Protokolls versucht zusätzliche Informationen zu gewinnen (beispielsweise wählt er bei einer vom Protokoll verlangten Wahl von Zufallszahlen geschickt und nicht zufällig).

- **aktiver Betrug:**
  Ein Teilnehmer hält sich nicht an das Protokoll (er gibt beispielsweise einen falschen Namen an).

1.4 Eine erste Public-Key-Idee (Merkles Puzzle)

Merkles Puzzel basiert auf der Idee, Rätsel zu erzeugen, die für den Sender Alice und den Empfänger Bob in vertretbarer Zeit zu lösen sind, nicht aber für einen Abhörer Eve. Damit soll das Ziel erreicht werden, daß vor der Kommunikation kein gemeinsamer Schlüssel zwischen Alice und Bob ausgetauscht werden muß.

Vorgehensweise:

1. Bob erzeugt $2^{20} (\approx 10^6)$ Nachrichten der Form **„Rätselnr.:** x_i**, Schlüssel:** y_i**“**, wobei x_i und y_i jeweils paarweise verschiedene Zufallszahlen sind. Die y_i haben eine Länge von mindestens 40 bit. Bob verschlüsselt diese Nachrichten mit einem symmetrischen Verfahren, wobei er verschiedene Schlüssel der Länge 20 bit benutzt, und schickt alles an Alice.

2. Alice wählt eine der verschlüsselten Nachrichten zufällig aus und entschlüsselt sie durch einen Brute-Force-Angriff (Aufwand ungefähr 10^6 Schritte). Alice erhält nach der Entschlüsselung die beiden Zufallszahlen x_0 und y_0.

3. Alice verschlüsselt die Nachricht m mit dem Schlüssel y_0 und schickt das Chiffrat zusammen mit x_0 im Klartext an Bob.

4. Bob wählt das zu x_0 gehörende y_0 aus seiner Datenbank und entschlüsselt das Chiffrat von Alice mit dem Schlüssel y_0.

Aufwandsbetrachtung:

Bob	$\approx$	2^{20} Schritte (Erzeugung der Rätsel)
Senden	$\approx$	2^{20} Rätsel von Bob und Chiffrat von Alice
Alice	$\approx$	2^{20} Schritte (Brute-Force-Angriff) und Verschlüsselung des Klartextes
Eve	$\approx$	2^{40} Schritte (Brute-Force-Angriff)

Der Abhörer Eve muß im schlimmsten Fall 2^{20} Rätsel mit Brute-force entschlüsseln bis er x_0 gefunden hat und damit auch y_0. Dazu braucht er aber ungefähr 2^{40} Schritte, da jedes Rätsel mit einem 20 bit langen Schlüssel verschlüsselt wurde. Es ist für Eve ungünstiger, das Chiffrat von Alice direkt mit einem Brute-Force-Angriff zu entschlüsseln, da er dazu mehr als 2^{40} Schritte bräuchte, weil die Schlüssel y_i eine Länge von mehr als 40 bit haben.

Wenn Alice und Bob 20.000 Operationen pro Sekunde ausführen können, dann benötigen sie jeweils ungefähr eine Minute. Eve muß aber bei vergleichbarer Rechenleistung etwa zwei Jahre investieren. Bei einer größeren Rechenleistung von 20.000.000 Operationen pro Sekunde und 2^{30} Rätseln ist der Aufwand der legalen Benutzer immer noch etwa eine Minute, der Aufwand eines Angreifers allerdings 2.000 Jahre. Man sieht also, daß der Aufwand für den Abhörer Eve deutlich schneller wächst, wenn die Anzahl der Rätsel und die Länge der Schlüssel erhöht wird.

1.5 Bausteine für Protokolle

1.5.1 Einwegfunktionen

Einwegfunktionen bilden einen zentralen Baustein der Public-Key-Protokolle.

Definition 1.3 (Einwegfunktion). Eine *Einwegfunktion* ist eine Funktion f, die einfach zu berechnen ist, deren Umkehrfunktion f^{-1} dagegen sehr schwierig zu berechnen ist.

Eine mathematisch exakte Definition wäre: Die Funktion f ist in polynomialer Zeit zu berechnen, die Umkehrfunktion f^{-1} aber nicht. Es ist bisher von keiner Funktionen bewiesen, daß sie diese Eigenschaft erfüllt.

Beispiel 1.4. Es sei eine große Primzahl $p \in \mathbb{N}$ und ein geeignetes Element $\alpha \in \mathbb{Z}_p$ gegeben. Das Potenzieren von α ist eine sehr einfache Funktion, das Berechnen eines Logarithmus $m = \log_\alpha(\alpha^m)$ ist normalerweise schwierig.

Bereits bei dem sehr kleinen Beispiel $\log_{50}(72)$ in $\mathbb{Z}_{101}$ ist man im Kopfrechnen oder mit einem (nicht programmierbaren) Taschenrechner eine kleine Weile beschäftigt. Umgekehrt zu verifizieren, daß $50^9 = 72$ in $\mathbb{Z}_{101}$ gilt, ist schnell geschehen.

Beispiel 1.5 (Diffie-Hellman Schlüsselaustausch). Ein Primzahlmodulus p und ein Erzeuger α von $\mathbb{Z}_p^*$ seien öffentlich bekannt. Alice und Bob wollen über einen unsicheren Kanal einen geheimen, gemeinsamen Schlüssel austauschen:

1. Alice wählt dazu eine Zufallszahl a, berechnet α^a und sendet α^a an Bob.

2. Bob wählt eine Zufallszahl b, berechnet α^b und sendet α^b an Alice.
3. Alice empfängt α^b und berechnet $(\alpha^b)^a$.
4. Bob empfängt α^a und berechnet $(\alpha^a)^b = \alpha^{a \cdot b} = (\alpha^b)^a$.

Damit haben Alice und Bob einen gemeinsamen Schlüssel, der nicht über das Netz übertragen wurde. Zum Berechnen dieses gemeinsamen Schlüssels müßte ein Angreifer z. B. den diskreten Logarithmus aus α^b berechnen. Dieser Schlüsselaustausch wird in Abschnitt 7.5.1 genauer behandelt.

Definition 1.6 (Einwegfunktion mit Falltür). Eine *Einwegfunktion mit Falltür* ist eine Einwegfunktion f, deren Umkehrfunktion f^{-1} einfach zu berechnen ist, falls man eine zusätzliche Information besitzt.

Beispiel 1.7. Das RSA-Verfahren, das in Abschnitt 2.2 genauer betrachtet wird, ist das Musterbeispiel für eine Falltürfunktion. Es funktioniert folgendermaßen:

1. A wählt zwei große Primzahlen p und q und berechnet das Produkt $n = p \cdot q$.
2. Weiter wählt A einen öffentlichen Schlüssel $e \in \mathbb{N}$ mit $\mathrm{ggT}(e, (p-1)(q-1)) = 1$ und veröffentlicht n, e.

Damit kann B eine Nachricht $m \in \mathbb{N}, m < n$ zu $m^e \bmod n$ verschlüsseln, die nur A entschlüsseln kann. Das Verschlüsseln ist eine einfach zu berechnende Funktion, die Umkehrung dazu, also aus $m^e \bmod n$ die e-te Wurzel zu ziehen, ist aber sehr schwierig.

Kennt man die Faktorisierung von n, so ist die Umkehrung aber auch einfach zu berechnen: Zunächst berechnet man ein $d \in \mathbb{N}$ mit $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$, dies ist eine Inversion in $\mathbb{Z}_{(p-1)(q-1)}$, die man mit dem erweiterten Euklidschen Algorithmus ausrechnen kann. Dann gilt $(m^e)^d = m$ und die schwierige Umkehrfunktion ist zu einem einfachen Potenzieren geworden.

1.5.2 Hashfunktionen

Definition und Funktionsweise

Definition 1.8 (Hashfunktion). Eine *Hashfunktion* ist eine Funktion H, die eine beliebig lange Eingabe auf eine Ausgabe fester Länge (normalerweise kürzer) abbildet.

Beispiel 1.9. Das XOR aller Eingabebytes ist eine Hashfunktion.

In der Kryptographie wird eine Hashfunktion folgendermaßen eingesetzt: Zu einer Nachricht wird der Hashwert berechnet und zusammen mit der Nachricht übertragen. Der Empfänger kann durch Berechnen des Hashwerts der empfangenen Nachricht und Vergleich mit dem empfangenen Hashwert überprüfen, ob die Nachricht verfälscht wurde. Dazu ist es notwendig, daß der *kurze* Hashwert auf eine sichere Weise übertragen wird, er also auf keinen Fall verändert werden kann, bzw. eine Veränderung erkennbar ist.

Definition 1.10 (kryptographische Hashfunktion). Eine Hashfunktion, die zusätzlich noch folgende Bedingungen erfüllt, heißt eine *kryptographische Hashfunktion*:

- sie ist einfach zu berechnen,
- es ist schwierig zu einer vorgegebenen Ausgabe (Hashwert) ein Urbild (eine der höchstwahrscheinlich unendlich vielen Nachrichten mit diesem Hashwert) zu finden, und
- es ist schwierig zwei Nachrichten mit demselben Hashwert (Kollision) zu finden.

Beispiel 1.11 (Blockchiffre als Hashfunktion). Blockchiffren können als Grundbaustein für eine Hashfunktionen verwendet werden. Eine Blockchiffre

$$\begin{array}{rccc} C: & \{0,1\}^n \times \{0,1\}^k & \to & \{0,1\}^n \\ & (M,s) & \mapsto & c \end{array}$$

bilde einen Nachrichtenblock M mit n Bit und einen Schlüssel s mit k Bit auf ein Chiffrat c mit n Bit ab.

Für das Hashen wird der Klartext m in Blöcke $m_1, \ldots, m_r$ mit einer Länge von je k Bit zerlegt. Man wählt einen Initialisierungswert I und berechnet:

$$C(\ldots C(C(I, m_1), m_2) \ldots m_r).$$

Der Hashwert hat also immer eine Länge von n Bit.

Die Nachricht wird in den Schlüsseleingang eingegeben, da der Ausgang mit n Bit zurückgekoppelt werden sollte. Dazu ist der Dateneingang am besten geeignet.

Angriffe gegen Hashfunktionen

Es soll zu einer Nachricht m eine weitere Nachricht m' gefunden werden, so daß $H(m) = H(m')$ (Kollision) gilt. Dies kann verwendet werden, um dem Empfänger eine falsche Nachricht zuzuspielen, ohne daß er dies bemerken kann.

- **Brute-force-Angriff**
  Die Nachricht m und der Hashwert $H(m)$ seien bekannt. Es werden solange verschiedene Nachrichten m' ausprobiert, bis $H(m) = H(m')$ gilt. Durch Abändern der Nachricht m an r Stellen (durch Füllworte, Umstellungen) ergeben sich 2^r verschiedene Texte m'. Hat der Hashwert n bit Länge, so müssen etwa 2^n Nachrichten m' ausprobiert werden, bis $H(m) = H(m')$ auftritt.
- **Meet-in-the-middle-Angriff**
  Man kann Hashfunktionen normalerweise derart modifizieren, daß sie eine Nachricht von hinten, also vom bekannten Hashwert $H(m)$ der Nachricht, rückhashen. Dies ist

z. B. möglich, wenn die Nachricht über den Schlüsseleingang einer Blockchiffre eingegeben wird. Dann kann bei gegebenem Schlüssel der Blockchiffre (man beachte, daß diese Schlüssel gerade durch die zu hashende Nachricht gegeben sind) der Hashwert dechiffriert werden. Diese Tatsache wird ausgenutzt, um eine Kollision schneller als mit dem Brute-Force-Angriff zu finden. Dazu wird die Nachricht m in zwei Teile m_1 und m_2 geteilt. Im ersten Teil werden Änderungen an $n/2$ Stellen vorgenommen. Es entstehen dadurch $2^{n/2}$ verschiedene Nachrichten m_1' mit Vorwärtshashwerten $H(m_1')$. Diese Nachrichten werden nach dem Hashwert sortiert. (Dabei ist n wieder die Länge des Hashwerts in Bit.)

Mit Hilfe des bekannten Hashwerts $H(m)$ wird solange der Rückwärtshashwert $H_{rück}(m_2', H(m))$ vom entsprechend veränderten zweiten Teil m_2' der Nachricht m berechnet, bis eine Übereinstimmung des Rückwärtshashwerts $H_{rück}(m_2', H(m))$ mit einem der Vorwärtshashwerte $H(m_1')$ auftritt. Sie kann effizient mit Binärsuche festgestellt werden. Es sind verhältnismäßig wenige Veränderungen an m_2 notwendig, bei $2^{n/2}$ rückwärts gehashten Nachrichten hat man mit Wahrscheinlichkeit größer $0,6$ einen der Vorwärtshashwerte getroffen. Eine Kollisionsnachricht m' ergibt sich durch Hintereinanderschreiben von m_1' und m_2'.

Vergleich des Aufwands für $n = 64$

- Brute-Force-Angriff

Zeitaufwand	$\approx$	2^{64} Schritte (2^{64} m' hashen)
Speicheraufwand	$\approx$	nur eine Nachricht

- Meet-in-the-middle-Angriff

Vorwärts hashen	$\approx$	2^{32} Schritte (2^{32} m_1' hashen)
Rückwärts hashen	$\approx$	2^{32} Schritte (2^{32} m_2' hashen)
Sortieren der Tabelle	$\approx$	32×2^{32} Schritte
Suchen in der Tabelle	$\approx$	32×2^{32} Schritte
Speicheraufwand	$\approx$	$2^{32} \times 8$ byte = 32 GByte für $H(m_1')$
		$+2^{32} \times 4$ byte = 16 GByte für Textauswahl.

Lösung des Meet-in-the-middle-Problems: Will man k bit Sicherheit, so muß der Hashwert $2k$ bit lang sein.

1.6 Protokolle für Schlüsselaustausch

Es werden Protokolle benötigt, die den sicheren Austausch eines gemeinsamen Sitzungsschlüssels K zwischen zwei Teilnehmern koordinieren, um ihnen später den Einsatz eines schnellen symmetrischen Verfahrens zu ermöglichen.

1.6.1 Protokoll mit einem symmetrischen Verfahren und einer Schlüsselzentrale (SZ)

Die Schlüssel von Alice und Bob sind der SZ bekannt.

1. Alice teilt der SZ mit, daß sie mit Bob kommunizieren will.
2. Die SZ erzeugt einen Sitzungsschlüssel K, berechnet $E_A(K)$ und $E_B(K)$ und schickt dies an Alice.
3. Alice berechnet $D_A(E_A(K)) = K$.
4. Alice schickt $E_B(K)$ an Bob.
5. Bob berechnet $D_B(E_B(K)) = K$.
6. Alice und Bob besitzen jetzt einen gemeinsamen Sitzungschlüssel K, der nur ihnen und der SZ bekannt ist.

Problem: Die SZ kennt die Schlüssel aller Teilnehmer und alle Sitzungsschlüssel und kann somit alle Nachrichten abhören.

1.6.2 Protokoll mit einem Public-Key-Verfahren

Public-Key-Kryptosysteme sind etwa einen Faktor 1000 langsamer als Blockchiffren bei gleichen Sicherheitsanforderungen. Bei den symmetrischen Verfahren muß aber vor der Kommunikation ein geheimer Schlüssel sicher ausgetauscht werden. Daher wird in den meisten praktisch eingesetzten Systemen zunächst ein geheimer Sitzungsschlüssel K mit einem Public-Key-Verfahren übermittelt. Danach wird ein schnelleres symmetrisches Verfahren verwendet, das dann den Sitzungsschlüssel K benutzt. Diese Systeme heißen *Hybridsysteme*, da zwei kryptographische Verfahren eingesetzt werden.

Das grundlegende Protokoll

Die öffentlichen Schlüssel von Alice und Bob sind in einer Datenbank der Schlüsselzentrale gespeichert.

1. Alice holt Bobs öffentlichen Schlüssel B aus der Datenbank.
2. Alice erzeugt einen zufälligen Sitzungsschlüssel K, berechnet $E_B(K)$ und schickt dies mit ihrem Namen versehen an Bob.
3. Bob entschlüsselt $D_{B'}(E_B(K)) = K$ mit seinem geheimen Schlüssel B'.
4. Alice und Bob haben jetzt einen gemeinsamen Sitzungsschlüssel K.

Vorteil: Die SZ kennt nur die öffentlichen Schlüssel der Teilnehmer. Ihr sind die Sitzungsschlüssel nicht bekannt.

Dieses Protokoll bietet noch keine befriedigende Sicherheit, da sich ein aktiver Angreifer in einem Man-in-the-middle-Angriff zwischen die SZ und die Teilnehmer schalten kann. Er kann dann den Teilnehmern falsche Schlüssel schicken, ohne daß es von ihnen bemerkt wird. Hier ist der Einfachheit halber der aktive Angreifer zwischen Alice und Bob dazwischengeschaltet. Bei einem Angriff auf das obige Protokoll muß er noch die öffentlichen Schlüssel, die von der Schlüsselzentrale verschickt werden, fälschen.

Man-in-the-middle-Angriff

Ein passiver Angreifer Eve kann nur das Public-Key-Verfahren oder das symmetrische Verfahren angreifen. Eve sollte keine Chance haben, diese Verfahren zu brechen. Ein aktiver Angreifer Mallory kann aber die Leitung unterbrechen und die Kommunikation, die über ihn läuft, kontrollieren.

1. Alice schickt ihren öffentlichen Schlüssel A an Bob. Mallory fängt A ab und schickt seinen öffentlichen Schlüssel M an Bob mit Absender Alice.
2. Bob schickt seinen öffentlichen Schlüssel B an Alice. Mallory fängt B ab und schickt seinen öffentlichen Schlüssel M an Alice mit Absender Bob.
3. Alice verschlüsselt die Nachricht m und schickt das Chiffrat $c = (E_M(m))$ an Bob.
4. Mallory entschlüsselt $D_{M'}(c) = m$, liest die Nachricht und leitet $E_B(m)$ weiter an Bob.
5. Der Rückweg verläuft analog.

Dabei kann die Nachricht m auch der Sitzungsschlüssel sein, mit dem anschließend ein symmetrisches Verfahren initialisiert wird. Die von Alice und Bob ausgetauschten Nachrichten können von Mallory offensichtlich nicht nur gelesen, sondern auch in beliebiger Weise manipuliert werden. Alice und Bob haben keine Möglichkeit festzustellen, ob ihre Nachrichten abgehört oder manipuliert wurden.

Interlock-Protokoll

Die besten Public-Key-Verfahren sind also wertlos, wenn die Teilnehmer falsche Schlüssel von einem Angreifer Mallory bei einem Man-in-the-middle-Angriff bekommen. Wird das Public-Key-Verfahren für die gesamte Kommunikation und nicht nur für einen Schlüsselaustausch verwendet, so kann dieser Schwachpunkt teilweise durch das folgende Protokoll behoben werden. Hierbei muß es Alice und Bob möglich sein, einander Fragen zu stellen, die nur der jeweils andere beantworten kann.

1. Alice schickt Bob ihren öffentlichen Schlüssel A.
2. Bob schickt Alice seinen öffentlichen Schlüssel B.
3. Alice berechnet $E_B(m_A) = e_1e_2$ und schickt die erste Hälfte e_1 an Bob.
4. Bob berechnet $E_A(m_B) = f_1f_2$ und schickt die erste Hälfte f_1 an Alice.
5. Alice schickt die zweite Hälfte e_2 an Bob.
6. Bob berechnet $D_{B'}(e_1e_2) = m_A$. Bob beantwortet *sofort* diese „Frage“ m_A von Alice und schickt die zweite Hälfte f_2.
7. Alice beantwortet Bobs Frage $D_{A'}(f_1f_2)$.

Ein aktiver Angreifer Mallory kann bis zum 3-ten Schritt alles wie bisher machen. Dann muß aber Mallory eine neue Nachricht erfinden, sie verschlüsseln und die erste Hälfte an Bob schicken. Im 4-ten Schritt muß Mallory nochmals eine neue Nachricht erfinden, sie verschlüsseln und die erste Hälfte an Alice schicken. Im 5-ten Schritt weiß Mallory Alices Frage, muß aber die zweite Hälfte der erfundenen Nachricht an Bob schicken. Im 6-ten Schritt erwartet Alice eine Antwort von Bob, die jetzt aber Mallory beantworten muß. Mallory weiß entweder die Antwort oder muß raten. Der 7-te Schritt verläuft analog. Durch das Interlock-Protokoll wird der Man-in-the-middle-Angriff Mallory erschwert, jedoch nicht unmöglich gemacht.

Das Protokoll versagt, wenn Mallory die beiden Teilnehmer kennt oder wenn Alice und Bob sich nicht direkt kennen und daher keine geeigneten Fragen stellen können, d. h. kein gemeinsames Geheimnis besitzen. Das ist vor allem dann der Fall, wenn Alice und Bob keine Menschen, sondern Computer sind, die Fragen nur automatisch erzeugen können.

Schlüsselaustausch mit Signaturen

Das Interlockprotokoll erschwert einen Man-in-the-middle-Angriff, macht ihn aber nicht unmöglich. Das grundlegende Protokoll kann mit Hilfe einer digitalen Signatur so erweitert werden, daß ein Man-in-the-middle-Angriff für einen Außenstehenden Mallory unmöglich wird.

Die Schlüsselzentrale unterschreibt die öffentlichen Schlüssel von Alice und Bob, wobei sie persönlich anwesend sein müssen. Diese von der Schlüsselzentrale signierten Schlüssel sind in einer Datenbank zugänglich. Jeder Teilnehmer kennt den öffentlichen Schlüssel der Schlüsselzentrale und kann damit ihre Unterschrift prüfen. Mallory kann keinen Man-in-the-middle-Angriff durchführen, da Alice und Bob die Authentizität der Schlüssel überprüfen können. Mallory kann jetzt nur noch zuhören oder die Leitung stören. Die Schlüsselzentrale kann nicht wie bei der Vergabe von Sitzungsschlüsseln im Protokoll 1.6.1 alle Nachrichten abhören, ist also nicht mehr der große Schwachpunkt im System.

Die Schlüsselzentrale kann jetzt nur noch als einzige Instanz einen Man-in-the-middle-Angriff durchführen.

Mit Hilfe einer Schlüsselzentrale lassen sich auch sehr einfach Nachrichten verschicken. Der öffentliche Schlüssel von Bob wird von der Schlüsselzentrale geliefert. Alice erzeugt einen Sitzungsschlüssel K, verschlüsselt die Nachricht m mit einem symmetrischen Verfahren und schickt $(E_B(K), E_K(m))$ an Bob. Hier ist wieder wichtig, daß der öffentliche Schlüssel B signiert ist.

Kapitel 2

Public-Key-Algorithmen

Das Konzept der Public-Key-Kryptographie basiert auf der Tatsache, daß es möglich ist, zwei Schlüssel (einen öffentlichen E zum Verschlüsseln und einen geheimen D zum Entschlüsseln) zu benutzen, wobei es aber praktisch unmöglich ist, den geheimen Schlüssel aus dem öffentlichen zu berechnen. Viele der vorgeschlagenen Public-Key-Algorithmen stellten sich als unsicher heraus, und viele der sicheren Algorithmen sind nicht praktisch einsetzbar, da sie entweder zu große Schlüssel benötigen oder das Chiffrat im Vergleich zu der Nachricht viel zu groß wird. Von den sicheren und praktisch einsetzbaren Algorithmen eignen sich nur einige für eine Schlüsselvergabe. Einige der Algorithmen können nur zur Verschlüsselung und andere nur zur digitalen Signatur eingesetzt werden. Es gibt nur zwei Klassen von Algorithmen, die sich sowohl für Verschlüsselung als auch für digitale Signatur eignen und auch über längere Zeit häufig eingesetzt werden.

Die eine Klasse basiert auf dem Faktorisierungsproblem; das RSA-Verfahren ist der bekannteste Vertreter. Die andere Klasse basiert auf dem diskreten Logarithmusproblem in Gruppen; hier ist das Verfahren von ElGamal das bekannteste Verfahren. Um einen besseren Einblick in die Möglichkeiten der Public-Key-Kryptographie zu geben werden hier noch zwei weitere Ansätze dargestellt: das Knapsack-Verfahren und das Verfahren von McEliece. Diese Public-Key-Algorithmen sind langsam im Vergleich zu symmetrischen Verschlüsselungsalgorithmen. Daher werden in heutigen Systemen sowohl Public-Key-Komponenten, als auch symmetrische Komponenten eingesetzt; man spricht von Hybridkryptosystemen.

2.1 Knapsack

Die Sicherheit der Knapsack-Algorithmen beruht auf der Tatsache, daß das allgemeine Knapsackproblem ein NP-vollständiges Problem ist. Obwohl sich der ursprüngliche Algorithmus später als unsicher heraustellte, lohnt es sich dennoch, ihn zu untersuchen, um zu sehen, wie ein NP-vollständiges Problem für die Entwicklung eines Public-Key-Algorithmus verwendet werden kann.

Definition 2.1 (Das Knapsackproblem). Es seien beliebige Gewichte $a_1, \ldots, a_n \in \mathbb{N}$ und eine beliebige Summe $s \in \mathbb{N}$ gegeben. Es wird eine Teilmenge von $\{a_1, \ldots, a_n\}$ gesucht, so daß sich die in ihr enthaltenen Elemente zu s addieren. Anders formuliert lautet das Problen: Finde ein $(x_1, \ldots, x_n) \in \{0,1\}^n$ mit $\sum_{i=1}^{n} x_i a_i = s$.

Obwohl das allgemeine Knapsackproblem NP-vollständig ist, gibt es spezielle Formen eines Knapsacks, die einfach zu lösen sind. Ein superincreasing Knapsack ist ein Beispiel dafür.

Definition 2.2 (Der superincreasing Knapsack). Es seien $b_1, \ldots, b_n \in \mathbb{N}$ und $s \in \mathbb{N}$ ein Knapsack, wobei für alle i gilt $\sum_{j=1}^{i-1} b_j < b_i$.

Wegen dieser speziellen Struktur läßt sich der Knapsack $(b_1, \ldots, b_n)$ mit einem Greedy-Algorithmus von oben her lösen.

Das nächste Beispiel zeigt einen sehr einfachen superincreasing Knapsack.

Beispiel 2.3 (Ein einfaches Knapsackproblem). Es seien $a_i = 2^{i-1}$ für $i = 1, \ldots, n$. Damit ist es sehr einfach, das Knapsackproblem zu lösen. Der Lösungsvektor $(x_1, \ldots, x_n)$ entspricht der rückwärts gelesenen Binärdarstellung von s.

Das Knapsackproblem kann wie folgt in einem Public-Key-Algorithmus verwendet werden:

- öffentlicher Schlüssel: $(a_1, \ldots, a_n) \in \mathbb{N}^n$;
- geheimer Schlüssel: Ein Algorithmus, der die Werte $(a_1, \ldots, a_n)$ in einen einfach lösbaren Knapsack transformiert;
- Verschlüsselung: $(x_1, \ldots, x_n) \mapsto s = \sum_{i=1}^{n} x_i a_i$;
- Entschlüsselung: Löse das Knapsackproblem $(a_1, \ldots, a_n)$ für die Summe s durch Transformation in den einfach lösbaren Knapsack.

Merkle und Hellman stellten 1976 ein Verfahren vor, das die oben beschriebene Methode benutzt. Sie verwendeten einen superincreasing Knapsack und verschleierten ihn mit einer oder mehreren modularen Multiplikationen.

2.1.1 Verfahren

- Wähle einen superincreasing Knapsack $(b_1, \ldots, b_n)$ und verschleiere ihn durch eine lineare Transformation. Wähle dazu $M \in \mathbb{N}$ mit $M > \sum_{i=1}^{n} b_i$ und $W \in \mathbb{N}$ mit $\mathrm{ggT}(M, W) = 1$. Berechne die lineare Transformation

$$a_i := W b_i \pmod{M} \tag{2.1}$$

  und veröffentliche das Tupel $(a_1, \ldots, a_n)$ als öffentlichen Schlüssel. Eine binäre Nachricht $(x_1, \ldots, x_n)$ wird verschlüsselt zu $s := \sum_{i=1}^{n} x_i a_i$.

- Entschlüsselt wird durch Zurückführung auf das ursprüngliche Problem. Berechne $s' := W^{-1}s \pmod M$ (W^{-1} existiert, da $\text{ggT}(W, M) = 1$) und löse das Knapsackproblem $(b_1, \ldots, b_n)$ für die Summe s'.

2.1.2 Angriff auf das Knapsackverfahren

Das Knapsackverfahren weist folgende Schwachstellen auf:

- Es ist linear, d. h. es gilt $\sum_{i=1}^{n}(x_i + y_i)a_i = \sum_{i=1}^{n} x_i a_i + \sum_{i=1}^{n} y_i a_i$ und
- Wenn einige a_i ungerade sind, so ist ein Bit Information über den Klartext direkt zu sehen (s gerade oder ungerade).

Um einen echten Angriff ausführen zu können, benötigen wir das folgende

Lemma 2.4. *Für einen superincreasing Knapsack $b_1, \ldots, b_n \in \mathbb{N}$ und $M \in \mathbb{N}$ mit $M > \sum_{i=1}^{n} b_i$ gilt:*

$$b_l < 2^{l-n} \cdot M \text{ für } l = 1, \ldots, n.$$

Beweis: Die verschärfte Behauptung $\sum_{i=1}^{l} b_i < 2^{l-n} \cdot M$ wird über Induktion gezeigt: Für $l = n$ ergibt sich die vorausgesetzte Bedingung $\sum_{i=1}^{n} b_i < M$.

Für $1 < l \leq n$ gilt nach Induktionsvoraussetzung

$$\sum_{i=1}^{l-1} b_i + b_l < 2^{l-n} \cdot M$$

und da $(b_1, \ldots, b_n)$ superincreasing ist auch

$$\begin{aligned} & \sum_{i=1}^{l-1} b_i < b_l. \\ \Rightarrow \quad & 2 \cdot \sum_{i=1}^{l-1} b_i < 2^{l-n} \cdot M \\ \Rightarrow \quad & \sum_{i=1}^{l-1} b_i < 2^{l-1-n} \cdot M, \end{aligned}$$

was die obige Ungleichung beweist.□

Der Angriff basiert auf folgender Überlegung: Aufgrund der Formel 2.1 existieren Zahlen $k_1, \ldots, k_n \in \mathbb{Z}$ mit

$$a_i U - k_i M = b_i, \tag{2.2}$$

wobei $U \equiv W^{-1} \pmod M$. Dividiert man beide Seiten der Formel 2.2 durch $a_i M$, so erhält man

$$\frac{U}{M} - \frac{k_i}{a_i} = \frac{b_i}{a_i M}. \tag{2.3}$$

Substrahiert man Gleichung 2.3 für $i := 1$ von der allgemeinen Gleichung 2.3, so ergibt sich

$$-\frac{k_i}{a_i} + \frac{k_1}{a_1} = \frac{b_i}{a_i M} - \frac{b_1}{a_1 M}$$

Multipliziert man diese Gleichung mit $a_1 a_i$, so erhält man

$$-a_1 k_i + a_i k_1 = \frac{a_1 b_i}{M} - \frac{a_i b_1}{M}. \tag{2.4}$$

Es gilt nach Lemma 2.4

$$\begin{aligned} |a_i k_1 - a_1 k_i| &\leq \max\left\{\frac{a_1 b_i}{M}, \frac{a_i b_1}{M}\right\} && (2.5)\\ &\leq 2^{i-n} M. && (2.6) \end{aligned}$$

Für sehr kleine i lassen sich damit Ungleichungen in $\mathbb{Z}$ angeben die einfach lösbar sind. Dabei wird ausgenutzt, daß a_i sehr groß und $2^{i-n}\, M$ sehr klein sind. Es reichen nur wenige Ungleichungen, um die k_i eindeutig festzulegen. Mit dem Algorithmus von Lenstra zum linearen Programmieren lassen sich solche Ungleichungen in polynomialer Zeit lösen. Im Algorithmus „Faktorisieren von ganzzahligen Polynomen“ von Lenstra ist ein Teilalgorithmus „Finden von fast orthogonalen Basen in ganzzahligen Gittern“ enthalten. Dieser Algorithmus wurde 1983 implementiert und konnte auch zum Brechen des Knapsackverfahrens ausgenutzt werden. Eine zweite günstige Eigenschaft, die das Verfahren vollständig zusammenbrechen ließ, ist, daß nicht die ursprünglichen U, M gefunden werden müssen. Wenn ein $\frac{u}{m} \approx \frac{U}{M}$ verwendet wird, so entsteht ein äquivalenter Knapsack, der dieselbe Lösung $(x_1, \ldots, x_n)$ besitzt.

2.2 RSA-Verfahren

Das RSA-Verfahren (benannt nach den Entdeckern Rivest, Shamir, Adleman) ist das bekannteste Public-Key-Verfahren. Die Sicherheit von RSA beruht auf der enormen Schwierigkeit, große Zahlen zu faktorisieren.

2.2.1 Verfahren (Textbook-RSA)

1. Wähle zwei große Primzahlen p und q (200–300 Dezimalstellen oder mehr).
2. Berechne das Produkt $n = p \cdot q$.
3. Wähle einen öffentlichen Schlüssel e mit $\mathrm{ggT}(e, (p-1)(q-1)) = 1$.
4. Berechne den geheimen Schlüssel d mit $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$ und veröffentliche das Tupel (n, e) als öffentlichen Schlüssel.

Satz 2.5 (Chinesischer Restsatz). *Gegeben seien r paarweise teilerfremde Zahlen $n_1, \ldots, n_r \in \mathbb{N}$ mit $n_i > 1$ $(i = 1, \ldots, r)$. Dann hat jedes System*

$$\begin{aligned} m &\equiv m_1 \bmod n_1 \\ &\vdots \quad\quad \vdots \\ m &\equiv m_r \bmod n_r \end{aligned}$$

eine eindeutige *Lösung $m \in \mathbb{Z}$ mit $0 \leq m < \prod_{i=1}^{r} n_i$; diese Lösung kann effizient berechnet werden.*

Mit Hilfe des Chinesischen Restsatzes sieht man, daß $m^{ed} \equiv m \pmod{n}$ für alle $m \in \mathbb{Z}_n$ gilt.

- Verschlüsseln:
  Es sind der Modulus n und der Schlüssel e öffentlich bekannt. Zur Verschlüsselung wird eine Nachricht m in $m_1, \ldots, m_l$ zerlegt mit $m_i < n$, und es werden die Chiffrate $c_i \equiv m_i^e \pmod{n}$ berechnet.
- Entschlüsseln:
  Berechne $m_i \equiv c_i^d \equiv (m_i^e)^d \equiv m_i^{ed} \equiv m_i^1$ für $i = 1, \ldots, l$ und setze daraus die Nachricht m zusammen.

Das RSA-Verfahren ist in Hardware etwa um einen Faktor 1000 und in Software etwa 100 mal langsamer als symmetrische Blockchiffren. Zur Beschleunigung wählt man geeignete einfache Exponenten (es wurden zum Beispiel $e = 3, 17$ und $2^{16} + 1$ in verschiedenen Standards vorgeschlagen). Zumindest bei $e = 3$ treten Probleme auf, die unten ausgeführt werden.

Die oben vorgestelle Version ist als sogenanntes „Textbook-RSA“ die reine Form der Verschlüsselungsprimitive, in der Praxis darf sie so nicht eingesetzt werden, da darauf diverse Angriffe existieren. Stattdessen werden Verfahren wie RSAES-OAEP benutzt, die beweisbare Sicherheitseigenschaften besitzen, siehe dazu den weiteren Verlauf dieses Kapitels.

2.2.2 Primzahlerzeugung

Die Sicherheit des RSA-Verfahrens hängt in ganz entscheidender Weise davon ab, daß die verwendeten Primzahlen geheim bleiben. Deshalb müssen beim Erzeugen der Primzahlen nicht vorhersehbare Zahlen generiert werden; sie dürfen beispielsweise nicht durch einen Pseudozufallsgenerator erzeugt werden, wenn der Anfangszustand erraten werden kann.

Fast alle Verfahren, große Primzahlen zu erzeugen, wählen sich zunächst eine große Zufallszahl und testen diese auf Primalität. Bei den Primzahltests unterscheidet man zwischen zwei Klassen von Algorithmen:

- Probabilistische Primzahltests (relativ schnell und einfach);

- deterministische Primzahltests (deutlich langsamer und aufwendig).

Die wichtigsten Vertreter der probabilistischen Primzahltests werden im nächsten Kapitel vorgestellt.

2.2.3 Angriffe auf Protokollebene

Angriff aufgrund eines gemeinsamen Modulus:

Jeder Teilnehmer erhält von der Zentrale sein Schlüsselpaar (e_i, d_i), wobei für alle Teilnehmer der Modulus n gleich ist. Soll eine Nachricht m an zwei Personen geschickt werden, so entsteht dadurch das folgende Problem:

Es werden die beiden Chiffrate m^{e_1} und m^{e_2} berechnet und an die entsprechenden Personen geschickt. Es gelte dabei $\mathrm{ggT}(e_1, e_2) = 1$ für die öffentlichen Schlüssel der Personen. Wenn Eve die beiden Chiffrate m^{e_1} und m^{e_2} belauscht, kann sie

$$re_1 + se_2 = 1$$

und damit auch

$$(m^{e_1})^r (m^{e_2})^s = m^{re_1+se_2} = m$$

berechnen. Es besteht allerdings ein noch gravierenderes Problem:

Alice kennt ihr Schlüsselpaar (e_1, d_1) und den öffentlichen Schlüssel e_2 von Bob. Ein Vielfaches von $(p-1)(q-1)$ kann Alice durch $e_1 \cdot d_1 - 1 = k \cdot (p-1)(q-1)$ berechnen. Ein Inverses d_2' von e_2 modulo $k \cdot (p-1)(q-1)$ ist mit dem erweiterten Euklidschen Algorithmus auch einfach zu bestimmen (hat e_2 und k einen gemeinsamen Teiler g, so wird ein Inverses von e_2 modulo $(k/g) \cdot (p-1)(q-1)$ bestimmt). Es ist $d_2' \equiv d_2 (\bmod\ (p-1)(q-1))$ und mit d_2' können beliebige Nachrichten an Bob entschlüsselt werden.

Deswegen muß für jeden Teilnehmer ein eigenes Paar von Primzahlen p und q gewählt werden.

Öffentlicher Schlüssel $e = 3$

Eine Nachricht m wird an drei verschiedene Benutzer mit dem öffentlichen Schlüssel $e = 3$ und den verschiedenen Moduli n_1, n_2, n_3 verschlüsselt. Eine Angreiferin Eve fängt die drei Nachrichten:

- $m^3 \bmod n_1$,
- $m^3 \bmod n_2$,
- $m^3 \bmod n_3$

ab. Damit kann sie mit dem chinesischen Restsatz m^3 mod $(n_1 \cdot n_2 \cdot n_3)$ (Satz 2.5) berechnen.

Da $m < n_i$ für $i = 1, 2, 3$ besitzt Eve damit m^3 als ganze Zahl und kann dort problemlos die dritte Wurzel ziehen und damit m erhalten.

Bei $e = 17$ funktioniert dieser Angriff selbstverständlich auch, es ist jedoch eher selten, daß eine (geheime) Nachricht an 17 verschiedene Benutzer verschickt wird, weshalb dort dieser Angriff an Bedeutung verliert.

Probleme bei kleinem Nachrichtenraum

Da die RSA-Verschlüsselung deterministisch ist (ein Chiffrat ist also *eindeutig* durch den Klartext und den verwendeten öffentlichen Schlüssel bestimmt), ist sie unsicher, falls klar ist, daß die verschlüsselte Nachricht aus einer kleinen Menge von Nachrichten gewählt wurde. Wird beispielsweise garantiert, daß entweder $m_1 =$ `ja` oder $m_2 =$ `nein` verschlüsselt wurde, kann durch einfaches Vergleichen des abgefangenen Chiffrats mit den (eindeutig bestimmten) Chiffraten $c_1 = m_1^e \bmod n$ bzw. $c_2 = m_2^e \bmod n$ erkannt werden, ob der verschlüsselte Klartext `ja` oder `nein` ist.

Eine offensichtliche Lösung dieses Problems ist nun, den zu verschlüsselnden Klartext m mit einem bei jeder Verschlüsselung neu zu wählenden Zufallsstring r zu „padden“, also aufzufüllen. Es wird dann $c = (m||r)^e \bmod n$ anstatt $c = m^e \bmod n$ berechnet.

Probleme bei Auktionen

Angenommen, es findet eine Auktion statt, bei der der Auktionator seinen RSA-Public-Key (n, e) öffentlich macht. Ein Bieter B_i gibt ein Gebot $g_i \in \{0, \ldots, n-1\}$ ab, indem er es einfach mit dem Public Key des Auktionators verschlüsselt, also das Chiffrat $c_i := g_i^e \bmod n$ zum Auktionator sendet. (Dabei soll angenommen werden, daß diese *Chiffrate* c_i prinzipiell abhörbar sind.)

Ein „böser“ Bieter B_j kann nun mühelos – nachdem er dessen chiffriertes Gebot c_i abgehört hat – einen anderen Bieter B_i überbieten, indem er $c_j := 2^e \cdot c_i \bmod n = (2 \cdot g_i)^e \bmod n$ setzt. Schlimmer noch: Weiß B_j, daß $100|g_i$ (etwa, weil oft nur „runde“ Gebote abgegeben werden), kann er B_i mit $c_j := (101/100)^e \cdot c_i \bmod n$ (Berechnung des Bruches in $\mathbb{Z}_n$!) gerade um ein Prozent überbieten.

Dieses Problem kann zwar auch mit einem geeigneten Padding der Klartexte (bzw. Gebote) behoben werden; es ist aber zu beachten, daß es nicht reicht, nur den unteren (niederwertigsten) Teil des Klartextes mit Zufallsbits aufzufüllen (warum nicht?).

2.3 ElGamal-Verfahren

Die Sicherheit des Verfahrens von ElGamal beruht auf der Schwierigkeit, den diskreten Logarithmus in endlichen zyklischen Gruppen zu bestimmen. Es kann sowohl zum Chiffrieren als auch zum Signieren von Nachrichten verwendet werden.

Typischerweise werden die Gruppen $\mathbb{F}_{2^m}^*$, $\mathbb{F}_p^*$ (p Primzahl) oder die Gruppe $E(\mathbb{F})$ der Punkte auf einer elliptischen Kurve über einem endlichen Körper benutzt, da sie sich besonders einfach implementieren lassen. Die Gruppe $E(\mathbb{F})$ ist entweder eine zyklische Gruppe oder das direkte Produkt zweier zyklischer Gruppen. Normalerweise ist die eine Komponente eine zyklische Gruppe kleiner Ordnung; die andere dann eine Gruppe großer Ordnung, die dann zum Verschlüsseln benutzt wird.

Chiffrierverfahren:

- Es sei g ein Erzeuger der Gruppe G. Die Gruppe G und der Erzeuger g seien öffentlich bekannt. Die Nachrichten werden durch Gruppenelemente dargestellt. Bob wählt eine Zufallszahl x (seinen geheimen Schlüssel), berechnet $y = g^x$ und veröffentlicht das Gruppenelement y (seinen öffentlichen Schlüssel).
- Alice will Bob eine Nachricht m senden, sie verschlüsselt diese, indem sie k zufällig mit $\mathrm{ggT}(\#G, k) = 1$ wählt und $a = g^k$ und $b = y^k m$ berechnet. Das Chiffrat ist das Tupel (a, b). Bob kann das Chiffrat (a, b) entschlüsseln, indem er

  $$m = b(y^k)^{-1} = b(g^{xk})^{-1} = b((g^k)^x)^{-1} = b(a^x)^{-1}$$

  mit seinem geheimen Schlüssel x ausrechnet.

Die Bedingung $\mathrm{ggT}(\#G, k) = 1$ ist für ein „korrektes“ Funktionieren der Ver- bzw. Entschlüsselung nicht notwendig, wird aber ein k gewählt, das mit $\#G$ einen großen gemeinsamen Teiler hat, so läßt sich der diskrete Logarithmus von g^k deutlich einfacher berechnen, da g^k in einer kleinen Untergruppe liegt. Diese Unsicherheit wird durch die zusätzliche Bedingung ausgeschlossen.

2.4 McEliece-Kryptosystem

Das McEliece-Kryptosystem benutzt fehlerkorrigierende Goppa-Codes, die einfach zu decodieren sind, und transformiert sie auf einen „allgemeinen“ linearen Code. Die Decodierung von allgemeinen linearen Codes ist ein NP-vollständiges Problem. Das McEliece-System ist folgendermaßen aufgebaut:

- Geheimer Schlüssel
  - G ist eine $k \times n$-Matrix, die Generator-Matrix eines Goppa-Codes, der maximal t Fehler korrigieren kann;
  - P eine $n \times n$-Permutationsmatrix;
  - S eine invertierbare $k \times k$-Matrix.
- Öffentlicher Schlüssel

  - $G' := S \cdot G \cdot P$ eine $k \times n$-Matrix;
  - die Nachricht ist ein Vektor der Länge k über $\{0, 1\}$;
  - verschlüsselt wird durch $c = m\,G' + e$, wobei e ein zufälliger Vektor vom Gewicht ein wenig kleiner als t ist.

- Entschlüsseln
  - $c' = cP^{-1}$;
  - decodiere c' zu m' , es gilt dann $c' = m'\,G + e'$;
  - es gilt dann $m = m'S^{-1}$.

Der Vorschlag von McEliece war

- $n = 1024$,
- $t \approx 50$ Fehler,
- maximal mögliches k, für das ein Goppa-Code mit diesen Parametern existiert, ist $k = 524$.

Bewertung

- Der Schlüssel hat etwa 2^{19} bit (64 KByte) Länge.
- Die Nachrichtenlänge wird fast verdoppelt; der Rechenaufwand ist aber wegen der einfachen Verschlüsselung und Entschlüsselung sehr gering (deutlich schneller als RSA).
- Praktisch wird das Verfahren wegen der Schlüsselgröße kaum eingesetzt.

Angriffe

Direktes Decodieren braucht ungefähr 2^{500} Schritte. Dieser Aufwand läßt sich erheblich reduzieren, wenn man k Spalten der Matrix (mit der Hoffnung, daß dort kein Fehler dazuaddiert wurde) auswählt und das lineare Gleichungssystem $mM_k = c_k$ löst. Falls kein Fehler in diesen Spalten aufgetreten ist, kann man m durch Matrixinversion zurückgewinnen. Es müssen allerdings viele Matrizen getestet werden, bis bei 524 Spalten keiner der 50 Fehler enthalten ist. Der Aufwand beträgt etwa 2^{81} Schritte.

Ein weiterer Angriff besteht in sogenannten *Reaction Attacks*, bei denen Rückmeldungen (z. B. durch Seitenkanäle), ob ein Chiffrat erfolgreich decodiert werden konnte, ausgenutzt werden. Hierbei ändert ein Angreifer einzelne Bits eines Chiffrats und beobachtet, ob der so erhaltene String decodiert werden kann. Damit kann dann gefolgert werden, ob das entsprechende Bit mit einem Fehler versehen war oder nicht, da nicht mehr decodiert werden kann, wenn der String mehr als t Fehler enthält. Somit kann dann durch weitere Tests der Fehlervektor bestimmt und damit decodiert werden. Da dieses Testen einzeln bitweise geschieht, kann so ein Angriff in linear vielen Schritten durchgeführt werden.

2.5 Beweisbar sichere Public-Key-Systeme

Nach diesen Betrachtungen soll nun der Begriff eines Public-Key-Kryptosystems auf einer etwas formaleren Ebene fixiert werden, so daß die Sicherheit eines Systems in gewissem Umfang *bewiesen* (bzw. auf bestimmte mathematische Annahmen zurückgeführt) werden kann. Insbesondere können bei als solchermaßen sicher bewiesenen Varianten des RSA-Systems Angriffe der vorgestellten Art ausgeschlossen werden.

Zunächst soll eine Festlegung getroffen werden, welche Arten von Algorithmen überhaupt betrachtet werden sollen. Damit sind sowohl Algorithmen, welche Angriffe modellieren, als auch Algorithmen, welche bei legitimer Benutzung des Systems benutzt werden sollen, gemeint. Es ist klar, daß hier bestimmte, „zu mächtige“ Angriffe sinnvollerweise ausgeschlossen werden sollten (etwa solche, die eine vollständige Suche nach einem passenden Secret Key durchführen).

Die folgende Definition sagt also, was unter einem „effizienten“ Algorithmus verstanden werden soll:

Definition 2.6. Ein Algorithmus A heißt *PPT* (probabilistic polynomial time), wenn er zum einen probabilistisch und zum anderen (strikt) laufzeitbeschränkt durch ein festes Polynom in der Länge seiner Eingaben ist.

Beispielsweise kann die Funktion $f(x) = x^2$ durch einen PPT-Algorithmus berechnet werden. Aber auch die Ausgabe von $|I|$ gleichverteilten Zufallsbits (wobei $|I|$ die Länge der Eingabe des Algorithmus ist) kann durch einen PPT-Algorithmus bewerkstelligt werden.

Weiter soll gefaßt werden, was eine „kleine“ Funktion ist. Zum Beispiel kann man nicht verhindern, daß ein Angreifer einen Secret Key zu einem gegebenen Public Key einfach rät. Ein solcher Angriff wird zwar mit einer geringen (positiven!) Wahrscheinlichkeit Erfolg haben, aber trotzdem ist diese Erfolgswahrscheinlichkeit meistens viel zu klein, um uns Sorgen zu bereiten. Man wird also Angriffe mit „geringer“ Erfolgswahrscheinlichkeit zulassen wollen, ohne ein Kryptosystem als unsicher zu betrachten:

Definition 2.7. Eine Funktion $f : \mathbb{N} \to \mathbb{R}$ heißt *vernachlässigbar*, wenn für jedes $c \in \mathbb{N}$ ein $k_c \in \mathbb{N}$ existiert, dergestalt daß für alle $k > k_c$ schon $|f(k)| < k^{-c}$ ist. Eine nicht vernachlässigbare Funktion heißt *nicht-vernachlässigbar*.

Es wird also gefordert, daß f „schließlich schneller als jedes Polynom“ verschwindet.

Umgekehrt kann es sinnvoll sein, „große“ Wahrscheinlichkeiten zu fassen:

Definition 2.8. Eine Funktion $f : \mathbb{N} \to \mathbb{R}$ heißt *überwältigend*, wenn $1 - f$ vernachlässigbar ist. Eine nicht überwältigende Funktion heißt *nicht-überwältigend*.

Beispielsweise ist $f(k) = k^{-\log k}$ vernachlässigbar, wohingegen $g(k) = 1 - 2^{-k}$ überwältigend ist.

2.5.1 Public-Key-Kryptosysteme

Nun soll definiert werden, was ein Public-Key-Kryptosystem formal ist:

Definition 2.9. Ein *Public-Key-Kryptosystem* P besteht aus drei PPT-Algorithmen KeyGen, $\mathsf{Encrypt}$, $\mathsf{Decrypt}$. Für beliebige $k \in \mathbb{N}$ und $pk, sk, m, c \in \{0,1\}^*$ werden folgende *syntaktische* Bedingungen gestellt:

- $\mathsf{KeyGen}(1^k) \in \{0,1\}^* \times \{0,1\}^*$,
- $\mathsf{Encrypt}(1^k, pk, m) \in \{0,1\}^*$,
- $\mathsf{Decrypt}(1^k, sk, c) \in \{0,1\}^* \cup \{\perp\}$.

Des weiteren wird verlangt, daß jeder mittels $(pk, sk) \leftarrow \mathsf{KeyGen}(1^k)$ generierte öffentliche Schlüssel pk eindeutig eine *Klartextmenge* $\mathcal{M}_{pk} \subseteq \{0,1\}^*$ spezifiziert. Schließlich wird *Korrektheit* in dem Sinne gefordert, daß die Wahrscheinlichkeit, ein Schlüsselpaar $(pk, sk) \leftarrow \mathsf{KeyGen}(1^k)$ zu generieren, so daß eine Nachricht $m \in \mathcal{M}_{pk}$ existiert, für die $\mathsf{Decrypt}(1^k, sk, \mathsf{Encrypt}(1^k, pk, m)) \neq m$ *möglich* ist, dabei als Funktion in k vernachlässigbar sein muß.

Man beachte hier die unäre Codierung eines expliziten *Sicherheitsparameters* k; ein solcher wird auch für die Definition etwa von Signatursystemen zur Anwendung kommen. Insbesondere dient er – oft im Zusammenhang mit dem gerade benutzten Begriff vernachlässigbarer Funktionen – als technisches Hilfsmittel für asymptotische Sicherheitsaussagen. Konkret kann man sich beispielsweise vorstellen, daß k angibt, wie lang ein zu generierender RSA-Schlüssel werden soll.

Definition 2.9 erlaubt zwar seltene „schlechte“ Schlüsselpaare, jedoch wird für „gute“ Schlüsselpaare gefordert, daß Entschlüsselung eines Chiffrates *immer* den zugehörigen Klartext liefert. Hier sind Abschwächungen denkbar; etwa könnte man seltene und klartextunabhängige Fehlentschlüsselungen erlauben. (Die Forderung eines *immer* fehlerfreien Systems würde oft zu stark sein: Beispielsweise ist nicht bekannt, wie man RSA-Schlüssel in strikt polynomialer Zeit zieht – das Problem ist hier die effiziente, *fehlerfreie* Primzahlgenerierung.)

Nun kann eine Sicherheitsdefinition für Public-Key-Kryptosysteme gemäß Definition 2.9 in Angriff genommen werden. Es soll dabei vermieden werden, zu spezifische Forderungen zu stellen; man würde dabei Gefahr laufen, gewisse Klassen von Angriffen einfach zu ignorieren. Weiter sollten die Angriffe auf das RSA-Kryptosystem in seiner ursprünglichen Form als Motivation dafür dienen, daß es *nicht* reicht, zu fordern, daß etwa die Funktion $\mathsf{Decrypt}$ schwer zu invertieren ist.

Der folgende Sicherheitsbegriff fordert, daß Chiffrate zu verschiedenen Klartexten nicht voneinander unterschieden werden können. Man beachte, daß dies von vornherein deterministische Verfahren wie etwa ein ungepaddetes RSA-Kryptosystem ausschließt. Die Intuition hinter diesem Begriff ist, daß, da noch nicht einmal zwei Chiffrate zu verschiedenen

Klartexten auseinandergehalten werden können, ein Chiffrat *überhaupt keine verwertbare Information* über den Klartext enthält.

Definition 2.10. Sei $P = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ ein Public-Key-Kryptosystem, $b \in \{0,1\}$ und $k \in \mathbb{N}$. Sei A ein Angreifer (d. h. ein PPT-Algorithmus) mit Zugriff auf ein Entschlüsselungsorakel $\mathsf{Decrypt}(1^k, sk, \cdot)$.[1] Man betrachte das folgende Experiment:

$$
\begin{array}{l}
\text{Experiment } \mathbf{Exp}^{\text{ind-cca-b}}_{P,A}(k): \\
\quad (pk, sk) \leftarrow \mathsf{KeyGen}(1^k) \\
\quad (m_0, m_1, s) \leftarrow A^{\mathsf{Decrypt}(1^k, sk, \cdot)}(1^k, pk, \texttt{find}) \\
\quad c \leftarrow \mathsf{Encrypt}(1^k, pk, m_b) \\
\quad \tilde{b} \leftarrow A^{\mathsf{Decrypt}(1^k, sk, \cdot)}(1^k, s, c, \texttt{guess}) \\
\quad \textbf{Return } \tilde{b}
\end{array}
$$

Hierbei wird verlangt, daß A das Orakel $\mathsf{Decrypt}(1^k, sk, \cdot)$ im „`guess`“-Schritt nie mit c startet.

Damit sei der *Unterscheidungsvorteil* von A definiert als

$$\mathbf{Adv}^{\text{ind-cca}}_{P,A}(k) := \mathsf{Pr}\left[\mathbf{Exp}^{\text{ind-cca-1}}_{P,A}(k) \to 1\right] - \mathsf{Pr}\left[\mathbf{Exp}^{\text{ind-cca-0}}_{P,A}(k) \to 1\right].$$

Das Schema P wird *IND-CCA-sicher* genannt, wenn die Funktion $\mathbf{Adv}^{\text{ind-cca}}_{P,A}(\cdot)$ für jeden in vorstehendem Sinne erlaubten Angreifer A vernachlässigbar ist.

Es wird also mit einem möglichen Angreifer A ein Spiel gespielt, in dem A zunächst zwei Klartexte m_0 und m_1 wählt. Einer der beiden Klartexte wird dann verschlüsselt, und A muß – unter Kenntnis allein des Chiffrates c! – raten, ob m_0 oder m_1 verschlüsselt wurde. Man sollte sich dabei von der formalen Notation nicht zu sehr beeindrucken lassen. Etwa dient die Variable s nur dazu, Information zwischen den beiden Angreifer-Aufrufen zu transportieren (ein PPT-Algorithmus hat kein „Gedächtnis“, welches zwischen mehreren Aufrufen erhalten bleibt). Beispielsweise könnte A im „`find`“-Schritt in s die Variablen pk, m_0 und m_1 zwischenspeichern.

Weiter wird offenbar ein *Chosen-Ciphertext-Angriff* modelliert, da A ein Entschlüsselungsorakel zur Verfügung steht. Es wurde dabei nur die kleine Einschränkung vorgenommen, daß A sein Entschlüsselungsorakel nie mit dem „Challenge Ciphertext“ c aufruft. Würde man dies zulassen, wäre ein offenbar nicht erfüllbarer Sicherheitsbegriff die Folge.

Klar ist, daß das RSA-Kryptosystem in der eingangs vorgestellten Form nicht IND-CCA-sicher ist (es ist ja determinisch). Um so erstaunlicher ist, daß ein geeignet gepaddetes RSA-Verfahren IND-CCA-sicher ist. Genauer kann für das RSAES-OAEP-Verfahren IND-CCA-Sicherheit gezeigt werden, wenn angenommen wird, daß die RSA-Funktion („hoch e modulo n“) schwer zu invertieren ist, und im Beweis einige Kunstgriffe benutzt werden.[2]

[1] A darf also Anfragen an $\mathsf{Decrypt}$ stellen, Chiffrate zu entschlüsseln, hat dabei jedoch keinen Einblick in die innere Struktur des Orakels.

[2] Genauer findet der Beweis im sogenannten „Random-Oracle-Modell“ statt, welches hier nicht besprochen werden soll.

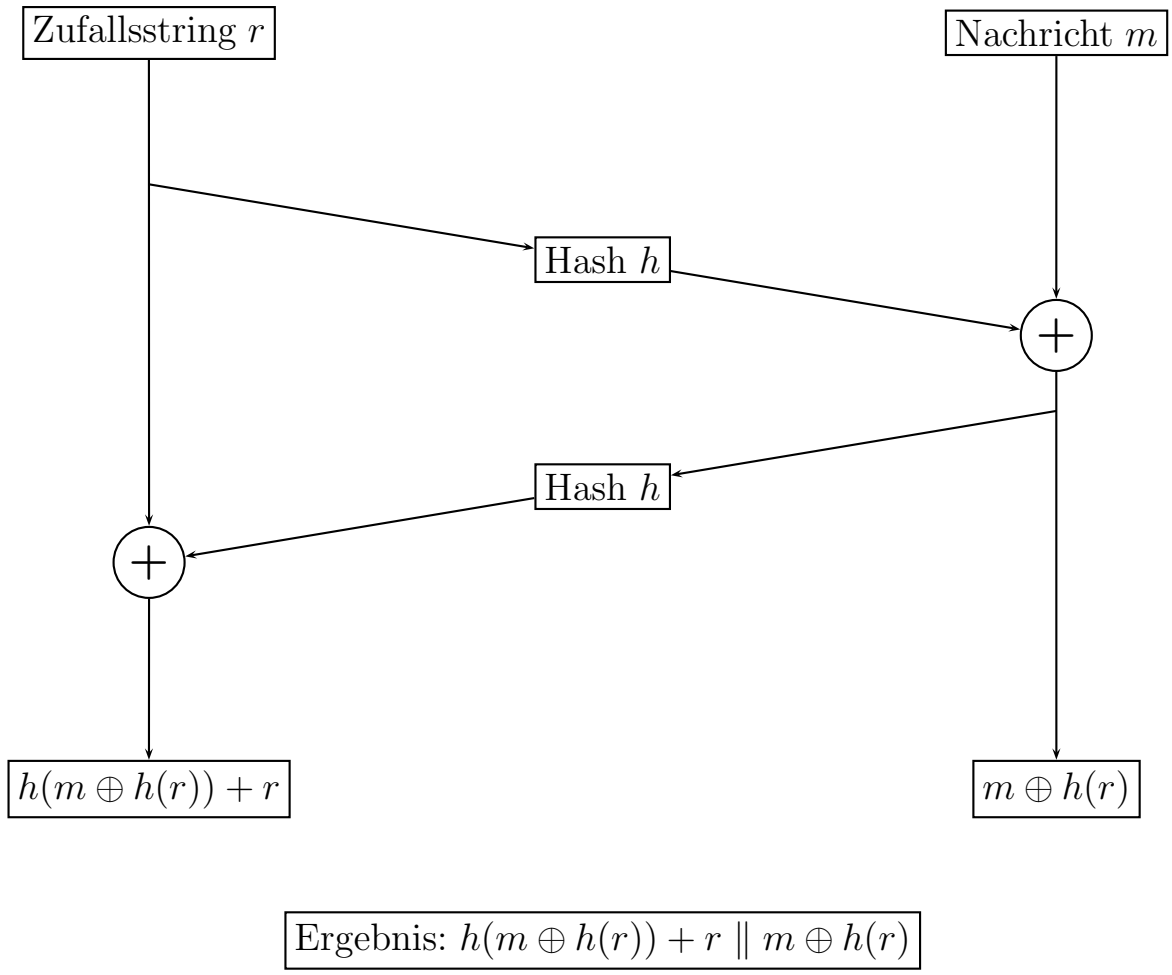


Abbildung 2.1: RSAES-OAEP-Verfahren (vereinfacht)

Der genaue Aufbau dieses Paddings ist in der Spezifikation des von den RSA Laboratories herausgegebenen Public Key Cryptography Standards nachzulesen: http://www.rsasecurity.com/rsalabs/node.asp?id=2125, eine vereinfachte Darstellung findet sich in Abbildung 2.1.

Kapitel 3

Probabilistische Primtests

Dieses Kapitel ist stark an das Skript von Schaefer-Lorinser [8] angelehnt. Hier werden die für das Verständnis der Public-Key-Kryptographie unbedingt notwendigen Teile aus [8] behandelt. Für ein besseres Verständnis der Primtests und Faktorisierungsalgorithmen ist die Lektüre des ausführlichen und weitergehenden Skripts wärmstens empfohlen.

3.1 Primzahlen

In diesem Kapitel werden wir uns mit ein paar grundlegenden Eigenschaften der Primzahlen beschäftigen, insbesondere werden wir ein paar Sätze kennenlernen, die die Verteilung der Primzahlen in den natürlichen Zahlen beschreiben. Beginnen wir jedoch mit der Definition von Primzahlen:

Definition 3.1 (Primzahl). Jede natürliche Zahl $n > 1$, die innerhalb der natürlichen Zahlen nur durch sich selbst und durch 1 teilbar ist, heißt *Primzahl*.

Die Primzahlen bilden die fundamentalen Atome der Arithmetik auf den natürlichen Zahlen, da sich jede natürliche Zahl in eindeutiger Weise als Produkt der Primzahlen darstellen läßt. Dies wird auch als der *Hauptsatz der Arithmetik* bezeichnet.

Um auf eine effiziente Weise eine Liste der Primzahlen kleiner einer vorgegebenen Zahl n zu erstellen, ist das auf den griechischen Mathematiker Eratosthenes (276–196 v. Chr.) zurückgehende Verfahren *„Sieb des Eratosthenes“* geeignet.

Schon aus der Antike ist der von Euklid ca. 325 v. Chr. in seinen Elementen notierte Satz bekannt:

Satz 3.2 (von Euklid). *Es gibt unendlich viele Primzahlen.*

Während uns dieser Satz also zeigt, daß in der Folge der natürlichen Zahlen immer wieder eine Primzahl vorkommt, zeigt der nächste Satz, daß auch beliebig große Lücken auftauchen können, das heißt Abschnitte der natürlichen Zahlen, die keine Primzahl enthalten.

Satz 3.3. *Zu jedem $k \in \mathbb{N}$ gibt es in den natürlichen Zahlen einen Abschnitt der Länge k, der keine Primzahl enthält.*

Beweis: Übung □

Das heißt also, die Primzahlen treten in gewisser Weise unregelmäßig auf. Es gibt keine geschlossene Formel, die beschreibt, welche natürlichen Zahlen Primzahlen sind. Dennoch kann eine recht präzise Aussage über die asymptotische Verteilung der Primzahlen gemacht werden, wie der folgende von Tschebyscheff bewiesene Satz zeigt. Wir definieren zunächst:

Definition 3.4. Es bezeichne für reelle Zahlen $x \in \mathbb{R}$ die Funktion $\pi(x)$ die Anzahl der Primzahlen, die nicht größer als x sind:

$$\pi(x) := \left|\left\{p \leq x \mid p \text{ ist Primzahl}\right\}\right|.$$

Um den Satz von Tschebyscheff über die Verteilung der Primzahlen beweisen zu können zeigen wir zunächst das folgende Lemma.

Lemma 3.5. *Es sei p prim und $V_p(m) := \max\left\{k \in \mathbb{N} \mid p^k \text{ teilt } m\right\}$. Dann gilt:*

$$V_p(n!) = \sum_{i=1}^{\infty} \left\lfloor \frac{n}{p^i} \right\rfloor.$$

Beweis: In $\lfloor \frac{n}{p} \rfloor$ der Zahlen von $1, \ldots, n$ ist der Faktor p enthalten, in $\lfloor \frac{n}{p^2} \rfloor$ der Faktor p^2 usw. $V_p(n!)$ kann also folgendermaßen berechnet werden:

$$\begin{aligned} V_p(n!) = \quad & \#\{m \mid 1 \leq m \leq n; p \text{ teilt } m\} && \left(= \lfloor \tfrac{n}{p} \rfloor\right) \\ + \quad & \#\{m \mid 1 \leq m \leq n; p^2 \text{ teilt } m\} && \left(= \lfloor \tfrac{n}{p^2} \rfloor\right) \\ & \vdots \\ + \quad & \#\{m \mid 1 \leq m \leq n; p^r \text{ teilt } m\} && \left(= \lfloor \tfrac{n}{p^r} \rfloor\right) \end{aligned}$$

mit r so daß $p^r \leq n < p^{r+1}$. □

Satz 3.6 (Tschebyscheff). *Es gibt $a, b \in \mathbb{R}_{>0}$, so daß für $x \geq 2$ gilt*

$$a \frac{x}{\ln x} < \pi(x) < b \frac{x}{\ln x}. \tag{3.1}$$

Beweis: Es sei $n \geq 2$ eine natürliche Zahl. Untersuche die Primfaktorzerlegung von $\binom{2n}{n}$ um Ungleichungen mit $\pi(n)$ und n zu erhalten. Zeige zunächst die folgenden beiden Behauptungen:

(1) $$\prod_{\substack{n < p \leq 2n \\ p \text{ prim}}} p \quad \text{teilt} \quad \binom{2n}{n}$$

(2) $\binom{2\,n}{n}$ teilt $\prod_{\substack{p \le 2\,n \\ p \text{ prim}}} p^{r_p}$,

wobei r_p für p prim definiert ist durch $p^{r_p} \le 2\,n < p^{r_p+1}$.

Zu (1):
Es gilt: $\binom{2\,n}{n} = \frac{2\,n \cdot (2\,n-1) \cdot \ldots \cdot (n+1)}{n \cdot (n-1) \cdot \ldots \cdot 1} \in \mathbb{N}$.
Eine Primzahl p mit $n < p \le 2\,n$ taucht dabei im Zähler, nicht aber im Nenner auf, was Behauptung (1) zeigt.

Zu (2):
Jeder Primteiler von $\binom{2\,n}{n}$ ist kleiner als $2\,n$. Es genügt also zu zeigen:

$$V_p\left(\binom{2\,n}{n}\right) \le r_p \text{ für alle } p < 2\,n.$$

Das Anwenden von Lemma 3.5 ergibt für alle $p < 2\,n$:

$$\begin{aligned} V_p\left(\binom{2\,n}{n}\right) &= V_p\left((2\,n)!\right) - 2 \cdot V_p\left(n!\right) \\ &= \sum_{i=1}^{\infty} \left(\lfloor \frac{2\,n}{p^i} \rfloor - 2 \cdot \lfloor \frac{n}{p^i} \rfloor \right) \\ &= \sum_{i=1}^{r_p} \underbrace{\left(\lfloor \frac{2\,n}{p^i} \rfloor - 2 \cdot \lfloor \frac{n}{p^i} \rfloor \right)}_{\le 1} \le r_p, \end{aligned}$$

was Behauptung (2) zeigt.

Behauptung (2) liefert uns:

$$\begin{aligned} 2^n \le \binom{2\,n}{n} &\le \prod_{\substack{p \le 2\,n \\ p \text{ prim}}} p^{r_p} \le (2\,n)^{\pi(2\,n)} \\ \Rightarrow \quad n \cdot \ln(2) &\le \pi(2\,n) \cdot \ln(2\,n) \end{aligned}$$

Damit ist die eine Ungleichung fast gezeigt; es bedarf noch der Erweiterung von den natürlichen Zahlen auf die reellen Zahlen. Für $x \ge 4$ folgt:

$$\pi(x) \ge \pi\left(2\lfloor \frac{x}{2} \rfloor\right) \ge \frac{\lfloor \frac{x}{2} \rfloor \ln(2)}{\ln\left(2\lfloor \frac{x}{2} \rfloor\right)} \ge \frac{\lfloor \frac{x}{2} \rfloor \ln(2)}{\ln(x)} > \frac{\ln(2)}{4} \cdot \frac{x}{\ln(x)}.$$

Für $x \in [2, 4)$ gilt diese Gleichung auch, dies ist einfach getrennt nachzurechnen. Damit ergibt sich für alle $x \ge 2$:

$$\frac{\ln(2)}{4} \cdot \frac{x}{\ln(x)} < \pi(x).$$

Die andere Ungleichung ergibt sich aus Behauptung (1) von oben:

$$\begin{aligned} n^{\pi(2\,n)-\pi(n)} &\leq \prod_{\substack{n < p \leq 2\,n \\ p \text{ prim}}} p \leq \binom{2\,n}{n} < 2^{2\,n} \\ \Rightarrow \quad & \big(\pi(2\,n) - \pi(n)\big)\log(n) < 2\,n \qquad\qquad | +2\,\pi(2\,n) \\ \Rightarrow \quad & \pi(2\,n)(2+\log(n)) - \pi(n)\log(n) < 2\,n + 2\pi(2\,n) \leq 6\,n \\ \Rightarrow \quad & \pi(2\,n)\log(4\,n) - \pi(n)\log(n) < 6\,n, \end{aligned}$$

wobei $\log(n)$ den Logarithmus zur Basis 2 bezeichnet.

Für $x \geq 4$ folgt daraus:

$$\begin{aligned} & \pi\left(2\lfloor\frac{x}{2}\rfloor\right)\log\left(4\lfloor\frac{x}{2}\rfloor\right) - \pi\left(\lfloor\frac{x}{2}\rfloor\right)\log\left(\lfloor\frac{x}{2}\rfloor\right) < 6\lfloor\frac{x}{2}\rfloor \leq 3\,x \\ \Rightarrow \quad & \pi(x)\log(x) - \pi\left(\frac{x}{2}\right)\log\left(\frac{x}{2}\right) < 3\,x + \log(x) < 4\,x. \end{aligned}$$

Für $x \in [2,4)$ läßt sich diese Gleichung wie oben direkt nachzurechnen.

Wähle nun $m \in \mathbb{N}$ mit $\frac{x}{2^{m+1}} < 2 \leq \frac{x}{2^m}$. Es folgt dann mit der obigen Ungleichung:

$$\begin{aligned} \pi(x)\log(x) &= \pi(x)\log(x) - \pi\left(\frac{x}{2^{m+1}}\right)\log\left(\frac{x}{2^{m+1}}\right) \\ &= \sum_{i=0}^{m} \underbrace{\left(\pi\left(\frac{x}{2^i}\right)\log\left(\frac{x}{2^i}\right) - \pi\left(\frac{x}{2^{i+1}}\right)\log\left(\frac{x}{2^{i+1}}\right)\right)}_{< 4\frac{x}{2^i}} \\ &< 4\,x \cdot \sum_{i=0}^{m} \frac{1}{2^i} \;\leq\; 8\,x. \end{aligned}$$

Substituieren des 2-er Logarithmus durch den natürlichen Logarithmus liefert die noch fehlende Ungleichung: $\pi(x)\ln(x) < 8\ln(2)\cdot x$ □

Betrachten wir einen Restklassenring $\mathbb{Z}_n$ modulo einer natürlichen Zahl n, dann ist unmittelbar klar, daß dieser Ring nicht die Körperaxiome erfüllt, falls n keine Primzahl ist. Denn unter dieser Voraussetzung ist jeder echte Teiler von n nicht invertierbar. Die teilerfremden Zahlen a mit $1 \leq a \leq n$, für die $\mathrm{ggT}(a,n) = 1$ gilt, sind in diesem Restklassenring $\mathbb{Z}_n$ invertierbar. Sie bilden die Einheitengruppe $\mathbb{Z}_n^*$, und die Anzahl der Elemente dieser Gruppe wird durch die sogenannte Eulersche φ-Funktion angegeben.

Definition 3.7 (Eulersche φ-Funktion). Die Funktion $\varphi(n)$ bezeichnet die Anzahl der natürlichen Zahlen kleiner n, die relativ prim zu n sind und wird die Eulersche φ-Funktion genannt:

$$\#\mathbb{Z}_n^* = \varphi(n) = \#\{m \mid 1 \leq m < n, \mathrm{ggT}(m,n) = 1\}$$

Wie kann diese Funktion explizit berechnet werden? Betrachten wir zunächst den einfachen Fall, daß n eine Primzahlpotenz p^e ist.

Bemerkung 3.8 (Primzahlpotenz p^e). *Es sei p eine Primzahl und $e \geq 1$. Dann gilt*

$$\varphi\left(p^e\right) = p^e - p^{e-1}.$$

Beweis: Genau für alle Produkte sp mit $s = 1, \ldots, p^{e-1} - 1$ gilt $\mathrm{ggT}(sp, p^e) \neq 1$. Damit sind die $(p^e - 1) - (p^{e-1} - 1)$ restlichen Zahlen teilerfremd zu p^e. □

Es gilt ferner, daß die Eulersche φ-Funktion für teilerfremde Zahlen multiplikativ ist:

Bemerkung 3.9 (Multiplikativität der Eulerschen φ-Funktion). *Es seien $m, n \geq 1$ zwei natürliche Zahlen mit $\mathrm{ggT}(m, n) = 1$. Dann gilt*

$$\varphi\left(m \cdot n\right) = \varphi\left(m\right) \cdot \varphi\left(n\right).$$

Beweis: Es gilt $\mathrm{ggT}(m \cdot n, s) = 1$ genau dann, wenn $\mathrm{ggT}(m, s)$ und $\mathrm{ggT}(n, s) = 1$. Aufgrund der Isomorphie nach dem Chinesischen Restesatz

$$\mathbb{Z}_{mn} \cong \mathbb{Z}_m \times \mathbb{Z}_n \quad \text{für } \mathrm{ggT}(m, n) = 1$$

ergibt sich daraus die Multiplikativität

$$\varphi\left(m \cdot n\right) = \varphi\left(m\right) \cdot \varphi\left(n\right).$$

Die Einheiten von $\mathbb{Z}_{mn}$ werden genau auf die Einheiten von $\mathbb{Z}_m \times \mathbb{Z}_n$ abgebildet. □

Aus diesen Bemerkungen folgt unmittelbar das Korollar, das die φ-Funktion für alle natürlichen Zahlen bestimmt.

Korollar 3.10 (Eulersche φ-Funktion für beliebige Zahlen). *Jedes $n \in \mathbb{N}_{>0}$ läßt sich eindeutig als $n = \prod_{i=1}^{r} p_i^{e_i}$ in seine Primfaktoren zerlegen. Damit gilt*

$$\varphi\left(p_1^{e_1} \cdots p_r^{e_r}\right) = \prod_{i=1}^{r} \left(p_i^{e_i} - p_i^{e_i - 1}\right).$$

Der folgende Satz geht auf Fermat (1601–1665) zurück und wird oft als *der kleine Satz von Fermat* bezeichnet.

Satz 3.11 (der kleine Satz von Fermat). *Es sei p prim, dann gilt für alle $b \in \mathbb{Z}_p^*$*

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

Das folgende weitergehende Resultat geht auf Euler (1707–1783) zurück. Es stellt eine Verallgemeinerung des kleinen Fermats dar, da es auch die Fälle, in denen p keine Primzahl ist, erfaßt.

Satz 3.12. *Es seien $n, b \in \mathbb{N}$ mit $\mathrm{ggT}(b, n) = 1$, das heißt b ist eine Einheit in dem Ring $\mathbb{Z}_n$. Dann gilt*

$$b^{\varphi(n)} \equiv 1 \pmod{n}.$$

Da für n prim gilt: $\varphi\left(n\right) = n - 1$, ist Satz 3.11 ein Spezialfall davon.

3.2 Quadratische Reste

Wir wollen im folgenden die Frage untersuchen, wann sich eine Einheit a des Restklassenrings $\mathbb{Z}_n$ als Quadrat einer anderen Einheit x darstellen läßt.

Definition 3.13 (Quadratische Reste). Es seien $a, n \in \mathbb{Z}$ mit $\mathrm{ggT}(a, n) = 1$. Falls ein x existiert mit

$$x^2 \equiv a \pmod{n},$$

so heißt a *quadratischer Rest* modulo n, andernfalls *quadratischer Nicht-Rest* modulo n.

Der folgende Satz gibt uns schon zwei wichtige Regeln für quadratische Reste bzw. quadratische Nicht-Reste.

Satz 3.14. *Das Produkt von zwei quadratischen Resten ist wieder ein quadratischer Rest, und das Produkt von einem quadratischen Rest und einem quadratischen Nicht-Rest ist ein quadratischer Nicht-Rest.*

Beweis: Der erste Teil des Beweises ist direkt einzusehen. Denn für zwei quadratische Reste a und b exisitieren x und y mit $x^2 \equiv a \pmod{n}$ und $y^2 \equiv b \pmod{n}$. Daraus folgt aber $(x \cdot y)^2 \equiv a \cdot b \pmod{n}$.

Den zweiten Fall beweisen wir durch einen Widerspruch. Angenommen, das Produkt von einem quadratischen Rest x^2 und einem quadratischen Nicht-Rest z wäre ein quadratischer Rest y^2. Daraus würde

$$x^2 \cdot z \equiv y^2 \pmod{n}$$

folgen. Da x als Einheit invertierbar ist, wäre dann z wegen

$$z \equiv (x^{-1} \cdot y)^2 \pmod{n}$$

ein quadratischer Rest, was der Voraussetzung widerspricht. □

Für den Fall, daß es sich bei dem Modulus n um eine Primzahl handelt, definieren wir das *Legendre Symbol*:

Definition 3.15 (Legendre Symbol). Es sei p eine Primzahl und $a \in \mathbb{N}$ teilerfremd zu p. Dann heißt das durch

$$\left(\frac{a}{p}\right) := \begin{cases} 1, & \text{falls } a \text{ ein quadratischer Rest ist} \\ -1, & \text{falls } a \text{ kein quadratischer Rest ist} \end{cases} \tag{3.2}$$

definierte Symbol $\left(\frac{a}{p}\right)$ das Legendre Symbol von a nach p.

Satz 3.16. *Es sei p eine Primzahl und es gelte $\mathrm{ggT}(a, p) = \mathrm{ggT}(b, p) = 1$. Aus $a \equiv b \pmod{p}$ folgt, daß $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$.*

Beweis: Es gilt:

$$\begin{aligned} & \left(\frac{a}{p}\right) = 1 \\ \Leftrightarrow \quad & \exists x \text{ mit } x^2 \equiv a \bmod p \\ \Leftrightarrow \quad & \exists x \text{ mit } x^2 \equiv a + l\,p = b \bmod p \\ \Leftrightarrow \quad & \left(\frac{b}{p}\right) = 1, \end{aligned}$$

was die Behauptung zeigt. □

Das Legendre Symbol ist konstant für alle $b \equiv a \bmod p$, es macht also Sinn vom Legendre Symbol von Restklassen $a \in \mathbb{Z}_p^*$ zu reden. Dies wird im nächsten Satz ausgenutzt.

Satz 3.17. *Im Falle von quadratischen Resten modulo einer Primzahl p gilt:*

$$\left(\frac{a}{p}\right)\left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right). \tag{3.3}$$

Ferner sind für $p \neq 2$ die Hälfte der Elemente der Einheitengruppe $\mathbb{Z}_p^*$ quadratische Reste und die andere Hälfte quadratische Nicht-Reste.

Beweis: Für $p = 2$ ist die Behauptung offensichtlich. Sei nun $p \neq 2$. Die Einheitengruppe $\mathbb{Z}_p^*$ ist eine zyklische Gruppe der Ordnung $p-1$. Es gibt also einen Erzeuger g der Ordung $p-1$ und

$$g, g^2, \ldots, g^{p-1}$$

ist bereits die ganze Gruppe $\mathbb{Z}_p^*$. Zu jedem Element $a \in \mathbb{Z}_p^*$ existiert ein Index i mit

$$g^i \equiv a \pmod p.$$

Der Erzeuger g ist kein quadratischer Rest, da sonst ein x mit $x^2 \equiv g \pmod p$ existieren würde. Nach dem Satz 3.11 würde für dieses x

$$x^{p-1} \equiv 1 \pmod p$$

gelten und somit $g^{(p-1)/2} \equiv 1 \pmod p$. Das ist aber ein Widerspruch dazu, daß g die ganze Einheitengruppe $\mathbb{Z}_p^*$ erzeugt.

Offensichtlich sind alle geraden Potenzen von g quadratische Reste

$$g^2, g^4, g^6, \ldots, g^{p-1}$$

und da g ein quadratischer Nicht-Rest ist, sind nach Satz 3.14 alle ungeraden Potenzen von g quadratische Nicht-Reste

$$g, g^3, g^5, \ldots, g^{p-2}.$$

Damit ist die Formel 3.3 bewiesen, da das Produkt von zwei ungeraden Potenzen eine gerade Potenz ergibt. □

Auf Euler geht das folgende Ergebnis zurück, das uns ein Kriterium liefert, für ein gegebenes Element $a \in \mathbb{Z}_p^*$ direkt zu entscheiden, ob es ein quadratischer Rest ist oder nicht.

Satz 3.18 (Euler-Kriterium). *Es sei $a \in \mathbb{N}$ mit $\mathrm{ggT}(a,p) = 1$, wobei p eine ungerade Primzahl ist. Dann gilt*

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}.$$

Beweis: Es sei g ein Erzeuger der Einheitengruppe $\mathbb{Z}_p^*$. Wir bestimmen zunächst ein $s \in \{0,\ldots,p-2\}$ mit

$$g^s \equiv -1 \pmod{p}.$$

Für ein solches s gilt

$$g^{2s} \equiv (g^s)^2 \equiv (-1)^2 \equiv 1 \pmod{p}.$$

Dies kann nur gelten, wenn $2s$ ein Vielfaches der Gruppenordnung $\#\mathbb{Z}_p^* = p-1$ ist, da $p-1$ der kleinste Exponent e ist, für den $g^e \equiv 1 \pmod{p}$ gilt. Also kann s nur einen der Werte

$$0, \frac{p-1}{2}, p-1, \ldots$$

annehmen. Da wegen Satz 3.11

$$g^0 \equiv g^{p-1} \equiv 1 \pmod{p}$$

und $s \leq p-2$ muß $s = \frac{p-1}{2}$ sein und somit

$$g^{(p-1)/2} \equiv -1 \pmod{p}.$$

Da g ein Erzeuger der Einheitengruppe $\mathbb{Z}_p^*$ ist, kann jedes Element $a \in \mathbb{Z}_p^*$ als $a \equiv g^i \pmod{p}$ dargestellt werden. Damit gilt

$$a^{(p-1)/2} \equiv (g^i)^{(p-1)/2} \equiv \left(g^{(p-1)/2}\right)^i \equiv (-1)^i \pmod{p}.$$

Das heißt aber, daß $a^{(p-1)/2} \equiv 1 \pmod{p}$ genau dann gilt, wenn i ein gerader Exponent ist, und $a^{(p-1)/2} \equiv -1 \pmod{p}$ genau dann, wenn i ein ungerader Exponent ist, was mit Satz 3.17 die Behauptung liefert. □

Ein sehr interessantes Ergebnis der Zahlentheorie, das wir ohne Beweis angeben, ist *das quadratische Reziprozitätsgesetz*, das von Gauß bewiesen wurde. Es stellt eine Relation zwischen dem Legendresymbol von zwei Primzahlen $\left(\frac{p}{q}\right)$ und der „Umkehrung“ $\left(\frac{q}{p}\right)$ dar.

Satz 3.19 (Quadratisches Reziprozitätsgesetz). *Seien p und q zwei ungerade Primzahlen. Dann gilt:*

$$\left(\frac{p}{q}\right) = \left(\frac{q}{p}\right)(-1)^{\frac{1}{2}(p-1)\frac{1}{2}(q-1)}. \tag{3.4}$$

Mit Hilfe des quadratischen Reziprozitätssatzes und von Satz 3.16 läßt sich manchmal effizient entscheiden, ob eine gegebene Zahl a modulo einer Primzahl p ein Quadratischer Rest ist. Eine vollständige Lösung gibt Satz 3.23.

Beispiel 3.20. Ist 19 ein Quadratischer Rest modulo 61? Es gilt

$$\left(\frac{19}{61}\right) = \left(\frac{61}{19}\right)(-1)^{30\cdot 9} = \left(\frac{61}{19}\right) = \left(\frac{4}{19}\right) = 1,$$

da $4 \equiv 2^2 \pmod{19}$.

Beim Legendre-Symbol sind Quadratische Reste modulo einer Primzahl p zugelassen. Im folgenden wird eine Verallgemeinerung des Legendre-Symbols, das Jacobi-Symbol, das auch für zusammengesetzte Zahlen definiert ist, eingeführt.

Definition 3.21 (Jacobi Symbol). Es seien $a, n \in \mathbb{Z}$ mit n ungerade und $\mathrm{ggT}(a, n) = 1$. Die Primfaktorzerlegung von n sei gegeben durch $n = \prod_{i=1}^{r} p_i^{e_i}$. Dann ist das Jacobi-Symbol definiert durch

$$\left(\frac{a}{n}\right) := \prod_{i=1}^{r} \left(\frac{a}{p_i}\right)^{e_i}.$$

Das Jacobi-Symbol ist so definiert, daß es den gleichen Rechenregeln wie das Legendre-Symbol genügt. Es ist aber zu beachten, daß das Jacobi-Symbol $\left(\frac{a}{n}\right)$ *nicht* angibt, ob a ein quadratischer Rest modulo n ist.

Es ist leicht einzusehen, daß für ein a, das ein Quadratischer Rest modulo einer zusammengesetzten Zahl $n = \prod_{i=1}^{r} p_i^{e_i}$ ist, das Jacobi-Sybmol $\left(\frac{a}{n}\right)$ den Wert 1 annimmt. Da a ein Quadratischer Rest modulo n ist, existiert ein $x \in \mathbb{N}$ mit $x^2 \equiv a \pmod{n}$. Betrachten wir diese Gleichung modulo p_i für $i = 1, \ldots, r$, so erhalten wir $x^2 \equiv a \pmod{p_i}$. Es ist also a auch Quadratischer Rest modulo aller Primfaktoren p_i für $i = 1, \ldots, r$ und damit ist das Jacobi-Symbol $\left(\frac{a}{n}\right)$ gleich 1.

Die Umkehrung dieser Aussage gilt nicht, wie das folgende Beispiel zeigt.

Beispiel 3.22. Das Jacobi-Symbol kann auch für Quadratische Nicht-Reste den Wert 1 annehmen:

$$\left(\frac{2}{15}\right) = \left(\frac{2}{3}\right)\left(\frac{2}{5}\right) = (-1)(-1) = 1,$$

obwohl 2 kein Quadratischer Rest modulo 15 ist.

Satz 3.23. *Es seien $m, n \in \mathbb{Z}$ ungerade. Für das Jacobi-Symbol gelten folgende Rechenregeln:*

(i)

$$\left(\frac{-1}{m}\right) = (-1)^{(m-1)/2},$$

(ii)

$$\left(\frac{2}{m}\right) = (-1)^{(m^2-1)/8},$$

(iii) Für m, n teilerfremd gilt:

$$\left(\frac{m}{n}\right) = \left(\frac{n}{m}\right)(-1)^{\frac{m-1}{2}\frac{n-1}{2}}.$$

(iv) Für $m_1 \cdot m_2, n$ teilerfremd und $m_1 \equiv m_2 \bmod n$ gilt:

$$\left(\frac{m_1}{n}\right) = \left(\frac{m_2}{n}\right).$$

(v) Für $m_1 \cdot m_2, n$ teilerfremd gilt:

$$\left(\frac{m_1}{n}\right) \cdot \left(\frac{m_2}{n}\right) = \left(\frac{m_1 \cdot m_2}{n}\right).$$

3.3 Pseudoprimzahlen

Betrachten wir zunächst einen einfachen Satz, der ein Kriterium liefert, mit dem sich entscheiden läßt, ob eine Zahl n prim oder zusammengesetzt ist.

Satz 3.24. *Eine Zahl $n > 1$ ist prim genau dann, wenn*

$$a^{n-1} \equiv 1 \pmod{n}, \tag{3.5}$$

für alle $a \in \mathbb{N}$ mit $2 \leq a < n$.

Beweis: Die Hinrichtung gilt aufgrund des Satzes 3.11.

Für die Rückrichtung nehmen wir an, die Fermat-Bedingung 3.5 sei erfüllt und n sei zusammengesetzt. Dann hat n nichttriviale Teiler r, s mit $1 < r, s < n$ und $n = r \cdot s$, und es gilt

$$r^{n-1} \equiv 1 \pmod{n} \quad \text{und} \quad s^{n-1} \equiv 1 \pmod{n}.$$

Damit gilt auch

$$1 \equiv r^{n-1} \cdot s^{n-1} \equiv (r \cdot s)^{n-1} \equiv 0 \pmod{n},$$

was einen Widerspruch liefert. □

Mit Hilfe dieses Satzes können wir bereits einen einfachen Primtest realisieren. Haben wir eine Zahl n gegeben, von der wir entscheiden möchten, ob sie prim oder zusammengesetzt ist, so wählen wir eine Basis a mit $2 \leq a < n$ und testen die Fermat-Bedingung 3.5. Erfüllt die Zahl n die Bedingung nicht, so ist n sicher eine zusammengesetzte Zahl. Andernfalls wählen wir eine neue Basis und führen den Test erneut durch.

Dies würde, falls n eine Primzahl ist, einen extrem aufwendigen Primtest ergeben. Wir können aber nach einer größeren Anzahl von Durchläufen des Primtests schon mit einer gewissen Wahrscheinlichkeit schließen, daß die Zahl n prim ist.

Definition 3.25. Eine zusammengesetzte Zahl n heißt *Pseudoprimzahl zur Basis* a, falls

$$a^{n-1} \equiv 1 \pmod{n}$$

erfüllt ist.

Beispiel 3.26. Die Zahl $9 = 3 \cdot 3$ ist eine Pseudoprimzahl zur Basis 8, denn

$$8^{9-1} \equiv 1 \pmod{9}.$$

Die Zahl $341 = 11 \cdot 31$ ist die kleinste Pseudoprimzahl zur Basis 2, es gilt:

$$2^{341-1} \equiv 1 \pmod{341}.$$

Satz 3.27. *Es sei $n \in \mathbb{N}$ eine Pseudoprimzahl zur Basis a_1 und zur Basis a_2. Dann ist n auch eine Pseudoprimzahl zur Basis $a_1 \cdot a_2$ und zur Basis $a_1 \cdot a_2^{-1} \bmod n$.*
Falls ein $a \in \mathbb{Z}_n^*$ existiert, das die Bedingung

$$a^{n-1} \equiv 1 \pmod{n}$$

nicht erfüllt, so wird sie auch von mindestens der Hälfte der Elemente aus $\mathbb{Z}_n^*$ nicht erfüllt.

Beweis: Der erste Teil ist durch direktes Nachrechnen zu beweisen. Es gilt

$$(a_1 \cdot a_2)^{n-1} \equiv a_1^{n-1} \cdot a_2^{n-1} \equiv 1 \pmod{n},$$

und damit ist n eine Pseudoprimzahl zur Basis $a_1 a_2$. Weiterhin gilt

$$(a_1 \cdot a_2^{-1})^{n-1} \equiv a_1^{n-1} \cdot (a_2^{n-1})^{-1} \equiv 1 \cdot 1^{-1} \equiv 1 \pmod{n},$$

und damit ist n eine Pseudoprimzahl zur Basis $a_1 a_2^{-1}$.

Es sei $B := \{a_1, \ldots, a_l\}$ die Menge aller $a \in \mathbb{Z}_n^*$ mit $a^{n-1} \equiv 1 \pmod{n}$. Die zusammengesetzte Zahl n ist also eine Pseudoprimzahl zur Basis $a_i \in B$ für $i = 1, \ldots, l$. Wir wählen $b \in \mathbb{Z}_n^* \setminus B$ beliebig aus. Wenn n eine Pseudoprimzahl zur Basis $b \cdot a_i$ wäre, dann wäre nach dem ersten Teil n auch eine Pseudoprimzahl zur Basis $(b \cdot a_i)a_i^{-1} \equiv b \pmod{n}$. Dies kann aufgrund der Wahl von b aber nicht sein, und damit sind alle Elemente aus $\{b \cdot a_1, \ldots, b \cdot a_l\}$ Basen, zu denen n keine Pseudoprimzahl ist. Damit sieht man, daß mindestens die Hälfte der Elemente aus $\mathbb{Z}_n^*$ Basen sind, zu denen n keine Pseudoprimzahl ist. □

Falls also zu einer zusammengesetzten Zahl n überhaupt eine Zahl $a \in \mathbb{Z}_n^*$ existiert, so daß n keine Pseudoprimzahl zur Basis a ist, so finden wir mit einer Wahrscheinlichkeit größer als $1/2$ eine solche Basis. Eine solche Basis a wird auch *Zeuge* für n genannt, weil die Angabe von a genügt, um zu beweisen, daß n eine zusammengesetzte Zahl ist.

Wählen wir nun k verschiedene Kandidaten für die Basis a auf zufällige Weise, so ist a für eine zusammengesetzte Zahl n nur mit Wahrscheinlichkeit kleiner als $1/2^k$ unter diesen Elementen kein Zeuge. Diese Aussage gilt wieder nur unter der Voraussetzung, daß es für n überhaupt einen Zeugen gibt. Es gibt Zahlen, für die das nicht erfüllt ist:

Eine zusammengesetzte Zahl n heißt *Carmichael Zahl*, wenn

$$a^{n-1} \equiv 1 \pmod{n}$$

für alle $a \in \mathbb{Z}_n^*$.

Für kleine Zahlen sind die Carmichael Zahlen recht selten [7]: Die Zahl $561 = 3 \cdot 11 \cdot 17$ ist beispielsweise die kleinste, und es gibt nur 2 163 unter $2,5 \cdot 10^{10}$ und 246 683 unter 10^{15}. Es wurde 1994 allerdings bewiesen, daß es unendlich viele Carmichael Zahlen gibt [1].

3.4 Probabilistischer Primtest nach Solovay, Strassen

Im vorigen Abschnitt haben wir aus dem kleinen Satz von Fermat eine Art heuristischen Primtest abgeleitet, indem wir für die auf Primalität zu testende Zahl n überprüften, ob sie die Eigenschaft 3.5, die der kleine Fermat für alle Primzahlen liefert, erfüllt. Dieser einfache Primtest versagt mit sehr hoher Wahrscheinlichkeit bei den Carmichael-Zahlen. Wir wählen in dem Primtest eine Zufallszahl a mit $1 < a < n$. Ist $\mathrm{ggT}(a, n) \neq 1$, so hat man einen Nullteiler gefunden, und weiß sofort, daß die Zahl n zusammengesetzt ist. Wie man aber leicht aus der Berechnungsvorschrift für die Eulersche φ-Funktion sieht, ist die Menge der Nullteiler im Vergleich zu der Menge der Einheiten sehr klein. Ist n eine Carmichael-Zahl, so ist n eine Pseudoprimzahl zu allen Einheiten aus $\mathbb{Z}_n^*$.

Die Grundidee des folgenden Primtests ist es, einen analogen Test bezüglich des Euler-Kriteriums vorzunehmen.

Definition 3.28. Analog zur Definition von Pseudoprimzahlen nennen wir zusammengesetzte, ungerade Zahlen n, für die

$$\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}$$

gilt, *Euler-Pseudoprimzahlen* zur Basis a.

Das folgende einfache Lemma zeigt schon, daß dieser Primtest den auf dem kleinen Fermat basierenden Primtest umfaßt.

Lemma 3.29. *Wenn n eine Euler-Pseudoprimzahl zur Basis a ist, dann ist n auch eine Pseudoprimzahl zur Basis a.*

Beweis: Quadrieren der Gleichung oben ergibt direkt Definition 3.25 und liefert die Behauptung. □

Der auf dieser Überlegung basierende Primtest wurde 1974 von Solovay und Strassen vorgeschlagen. Man nennt Algorithmen dieser Art probabilistisch, da es zusammengesetzte Zahlen gibt, für die der Test behauptet, die Zahl n sei mit einer gewissen Wahrscheinlichkeit prim. Durch Wiederholen des Primtests kann die Irrtumswahrscheinlichkeit exponentiell verringert werden. In der Praxis treten Zahlen, für die sich der Test auch nach mehrmaligem Anwenden irrt, äußerst selten auf.

Der Primzahltest nach Solovay und Strassen war das erste probabilistische Verfahren überhaupt. Heute werden in vielen Bereichen der Algorithmentechnik probabilistische Methoden eingesetzt.

Algorithmus:

1. Bestimme eine zufällig gewählte Zahl a mit $1 < a < n$ und $\mathrm{ggT}(a, n) = 1$.
2. Teste, ob $\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}$ gilt.
3. Falls ja, dann ist n wahrscheinlich prim, sonst ist n mit Sicherheit zusammengesetzt.

Das Jacobi-Symbol kann hierbei offensichtlich nicht über die Definiton berechnet werden, da dies die Primfaktorzerlegung von n voraussetzt. Es läßt sich aber effizient mit den Rechenregeln von Satz 3.23 ausrechnen.

Der große Vorteil dieses Primtests liegt nun darin, daß es für Euler-Pseudoprimzahlen nichts Äquivalentes zu den Carmichael-Zahlen gibt. Ferner läßt sich zeigen, daß die Irrtumswahrscheinlichkeit dieses Test kleiner als $1/2$ ist.

Satz 3.30 (Solovay, Strassen 1974). *Es sei* $P := \left\{a \;\middle|\; a \in \mathbb{Z}_n^*, \left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \bmod n\right\}$ *mit* $n > 2$ *ungerade.*

- *Falls* n *prim ist, so gilt* $\#P = n - 1$.
- *Falls* n *nicht prim ist, so gilt* $\#P < \frac{1}{2}(n-1)$.

Beweis: Falls n eine Primzahl ist, so liefert das Euler-Kriterium die Richtigkeit der Behauptung. Es sei nun n eine zusammengesetzte Zahl mit der Primfaktorzerlegung $n = \prod_{i=1}^{r} p_i^{e_i}$. Wir definieren zur Vereinfachung zwei Abbildungen

$$\begin{array}{rccc} \epsilon: & \mathbb{Z}_n^* & \to & \mathbb{Z}_n^* \\ & a & \mapsto & \left(\frac{a}{n}\right) \end{array} \tag{3.6}$$

und

$$\begin{array}{rccc} \delta: & \mathbb{Z}_n^* & \to & \mathbb{Z}_n^* \\ & a & \mapsto & a^{(n-1)/2} \end{array}. \tag{3.7}$$

Es ist also zu zeigen, daß $\#\{a \in \mathbb{Z}_n^* \mid \epsilon(a) = \delta(a)\} < \frac{1}{2}(n-1)$ erfüllt ist. Wir machen hierzu eine Fallunterscheidung.

1. Es gibt ein j mit $e_j > 1$ (d. h. n ist nicht quadratfrei).

   Angenommen es sei $\epsilon(a) = \delta(a)$ für alle $a \in \mathbb{Z}_n^*$. Dann gilt

$$a^{n-1} \equiv \left(a^{(n-1)/2}\right)^2 \equiv \delta(a)^2 \equiv \epsilon(a)^2 \equiv \left(\frac{a}{n}\right)^2 \equiv 1 \pmod{n} \tag{3.8}$$

für alle $a \in \mathbb{Z}_n^*$. Es gilt ferner

$$\begin{aligned} \#\mathbb{Z}_n^* &= \varphi(n) \\ &= \varphi(p_1^{e_1}) \cdots \varphi(p_r^{e_r}) \\ &= p_1^{(e_1-1)}(p_1-1) \cdots p_r^{(e_r-1)}(p_r-1). \end{aligned}$$

Daraus folgt $p_j \mid \#\mathbb{Z}_n^*$, da $e_j > 1$. Also gibt es ein Element $a_0 \in \mathbb{Z}_n^*$ der Ordnung p_j, d.h. p_j ist der kleinste Exponent, für den $a_0^{p_j} \equiv 1 \pmod{n}$ gilt. Da p_j kein Teiler von $n-1$ ist, gilt $a_0^{n-1} \not\equiv 1 \pmod{n}$. Dies ist aber ein Widerspruch zu Gleichung (3.8). Es ist also $\epsilon \neq \delta$ und damit ist die Menge $\{a \in \mathbb{Z}_n^* \mid \epsilon(a) = \delta(a)\}$ eine echte Untergruppe (sie ist multiplikativ und enthält das neutrale Element) von $\mathbb{Z}_n^*$. Damit gilt für die Untergruppenordung

$$\#\{a \in \mathbb{Z}_n^* \mid \epsilon(a) = \delta(a)\} \leq \frac{1}{2}\varphi(n)$$

und somit die Behauptung, da $\varphi(n) < n-1$ für zusammengesetztes n gilt.

2. Es gilt $e_i = 1$ für alle $i = 1, \ldots, r$ ($\mathbb{Z}_n$ ist isomorph zu dem direkten Produkt der Primkörper $\mathbb{Z}_{p_1}, \ldots, \mathbb{Z}_{p_r}$).

   Angenommen, es ist $\epsilon(a) = \delta(a)$ für alle $a \in \mathbb{Z}_n^*$. Wähle a_1 mit $\left(\frac{a_1}{p_1}\right) = -1$ und a_i mit $\left(\frac{a_i}{p_i}\right) = 1$ für $i = 2, \ldots, r$. Nach dem Chinesischem Restesatz existiert ein $a \in \mathbb{Z}$ mit $a \equiv a_i \pmod{p_i}$ für $i = 1, \ldots, r$. Es gilt nach Definition des Jacobi-Symbols und Korollar 3.16

   $$\epsilon(a) = \left(\frac{a}{n}\right) = \prod_{i=1}^{r}\left(\frac{a}{p_i}\right) = \prod_{i=1}^{r}\left(\frac{a_i}{p_i}\right) = -1 \cdot 1^{r-1} = -1$$

   und nach der Annahme ist $\delta(a) = -1$. Betrachten wir den durch den Chinesischen Restesatz gegebenen Isomorphismus

   $$\phi : \mathbb{Z}_n \longrightarrow \mathbb{Z}_{p_1} \times \ldots \times \mathbb{Z}_{p_r},$$

   der -1 auf $(-1, \ldots, -1)$ abbildet. Es gilt:

   $$\left(a_1^{(n-1)/2}, \ldots, a_r^{(n-1)/2}\right) = \phi(a^{(n-1)/2}) = \phi(\delta(a)) = \phi(-1) = (-1, \ldots, -1). \quad (3.9)$$

   Nach dem Chinesischen Restesatz existiert ein $b \in \mathbb{Z}$ mit $b \equiv 1 \pmod{p_1}$ und $b \equiv a_i \pmod{p_i}$ für $i = 2, \ldots, r$. Damit gilt

   $$\delta(b) = b^{(n-1)/2} \mapsto \left(1^{(n-1)/2}, a_2^{(n-1)/2}, \ldots, a_r^{(n-1)/2)}\right) = (1, -1, \ldots, -1).$$

   Es ist also $b^{(n-1)/2} \not\equiv \pm 1$, da $\phi(1) = (1, 1, \ldots, 1)$ und $\phi(-1) = (-1, -1, \ldots, -1)$, was ein Widerspruch zu $\delta(b) = \epsilon(b) \in \{\pm 1\}$ ist.

   Damit gilt $\epsilon \neq \delta$ und mit dem selben Untergruppenkriterium wie in ersten Fall folgt die Behauptung. □

3.5 Probabilistischer Primtest nach Rabin und Miller

Der folgende Primtest wurde von Rabin aufbauend auf den Ideen von Miller entwickelt. Er liefert eine Verbesserung des Primtests von Solovay und Strassen, da die Menge der Fälle, in denen er sich irrt, eine echte Untermenge der Fälle ist, in denen sich der Primtest von Solovay und Strassen irrt.

Der Algorithmus geht von folgender grundsätzlicher Überlegung aus: Nehmen wir an, wir hätten eine Pseudoprimzahl n zur Basis a, also mit

$$a^{n-1} \equiv 1 \pmod{n},$$

gegeben. Betrachten wir Quadratwurzeln aus a^{n-1}, so würde für eine Primzahl n gelten, daß die Quadratwurzel ± 1 sind, da in einem Körper zwei Lösungen für $X^2 \equiv 1 \pmod{n}$ existieren. Das heißt, für eine Primzahl n würde die Folge

$$a^{(n-1)/2}, a^{(n-1)/2^2}, \ldots, a^{(n-1)/2^k}$$

für $n-1 = 2^k m$ mit m ungerade aus entweder beliebig vielen 1en, gefolgt von einer -1 und dann beliebigen Zahlen aus $\mathbb{Z}_n^*$ oder lauter 1en bestehen. Diese Beobachtung wird in dem Miller-Rabin-Test ausgenutzt. Zur Berechnung geht man hier natürlich von $a^{(n-1)/2^k}$ aus und quadriert.

Algorithmus

1. Schreibe $n - 1 = 2^k \cdot m$ mit m ungerade.
2. Wähle eine Zufallszahl a mit $1 < a < n$ und $\mathrm{ggT}(a, n) = 1$.
3. Initialisiere $i := 0$ und $b := a^m$.
4. Falls $b = 1$, dann ist n wahrscheinlich prim. Stop
5. Falls $b = -1$, dann ist n wahrscheinlich prim. Stop
6. Falls ($i > 0$ und $b = 1$), dann ist n sicher zusammengesetzt. Stop
   Sonst $b := b^2$ und $i := i + 1$
7. Falls $i = k$, dann ist n sicher zusammengesetzt. Stop
   Sonst gehe zu Schritt 5.

Der folgende Satz liefert eine Abschätzung der Fehlerwahrscheinlichkeit dieses Algorithmus.

Satz 3.31 (Rabin). *Es sei $n \in \mathbb{N}$ ungerade und mit $n-1 = 2^k \cdot m$, wobei m ungerade ist. Zu $a \in \mathbb{N}$ mit $1 < a < n$ definieren wir eine Folge $a_0, \ldots, a_k$ mit $a_i := a^{2^i \cdot m}$ für $0 \leq i \leq k$. Es sei*

$$P := \{a \in \mathbb{Z}_n^* \mid (a_0 \equiv 1 \pmod{n}) \text{ oder } (\exists i < k : a_i \equiv -1 \pmod{n})\}.$$

- *Falls n prim, so gilt $\#P = n - 1$.*
- *Falls n nicht prim, so gilt $\#P \leq \frac{1}{4}(n-1)$.*

Beweis: siehe [8] □

Der Algorithmus erkennt die Primzahlen mit einer Wahrscheinlichkeit größer als 3/4. Durch t-faches Wiederholen des Algorithmus kann die Fehlerwahrscheinlichkeit auf $1/4^t$ gesenkt werden.

Der nächste Satz zeigt, daß der Algorithmus von Rabin auch wirklich eine Verbesserung des Algorithmus von Solovay und Strassen darstellt.

Satz 3.32. *Es sei*

$$P' := \left\{a \in \mathbb{Z}_n^* \;\middle|\; \left(\frac{a}{n}\right) \equiv a^{(n-1)/2}\right\}$$

und

$$P := \{a \in \mathbb{Z}_n^* \mid (a_0 \equiv 1 \pmod{n}) \text{ oder } (\exists i < k : a_i \equiv -1 \pmod{n})\},$$

wobei $1 < a < n$, n ungerade und zusammengesetzt mit $n-1 = 2^k \cdot m$ und $(a_i)_i$ eine Folge mit $a_0, \ldots, a_k$ wie oben. Dann gilt $P \subset P'$.

Beweis: siehe [8] □

Diesen theoretischen Untersuchungen, daß der Test von Rabin und Miller (deutlich) bessere Schranken besitzt, als der Test von Solovay und Strassen stellen wir die ernüchternde Tabelle 3.1 gegenüber:

Dort wird die Anzahl der Fehlversuche der beiden Tests beim Durchlaufen aller Elemente aus $\mathbb{Z}_n^*$ angegeben.

Betrachten wir nun den Fall eines zusammengesetzten n, für das der Test von Solovay/Strassen für ein $a \in \mathbb{Z}_n^*$ fehlschlägt, der Test von Rabin/Miller aber die richtige Antwort „n zusammengesetzt“ gibt.

Der Test von Rabin/Miller liefert diese Aussage nur im Schritt 6 oder Schritt 7 des Algorithmus. Da aber $a^{(n-1)/2} = \left(\frac{a}{n}\right)$, folgt $a^{n-1} = 1$ und es wurde bei der Berechnung also ein $b \neq \pm 1$ mit $b^2 = 1$ gefunden. Daraus berechnet sich direkt: $(b-1)(b+1) \equiv b^2 - 1 \equiv 0 \bmod n$ und damit liefert $\mathrm{ggT}((b-1), n)$ einen nichttrivialen Faktor von n. Dies läßt sich in der folgenden Bemerkung zusammenfassen:

Bemerkung 3.33. *Es sei n zusammengesetzt, und $a \in \mathbb{Z}_n^*$ eine Basis, bei der der Primtest von Rabin/Miller richtig und der von Solovay/Strassen falsch antwortet. Dann kann mit einem Rechenaufwand von $2\log(n)$ Multiplikationen* mod *n und einer* ggT*-Berechnung ein nichttrivialer Faktor von n berechnet werden.*

n	SoSt	RaMi	Diff.	NTeil
9	0	0	0	2
15	0	0	0	6
21	0	0	0	8
25	2	2	0	4
27	0	0	0	8
33	0	0	0	12
35	0	0	0	10
39	0	0	0	14
45	2	0	2	20
49	4	4	0	6
105	6	0	6	56
111	0	0	0	38
115	0	0	0	26
117	2	0	2	44
119	0	0	0	22
121	8	8	0	10
123	0	0	0	42
125	2	2	0	24
129	0	0	0	44
133	16	16	0	24
135	0	0	0	62
141	0	0	0	48
143	0	0	0	22
145	6	4	2	32
147	0	0	0	62
1001	38	8	30	280
1003	0	0	0	74
1005	2	0	2	476
1007	0	0	0	70
1011	0	0	0	338
1015	4	4	0	342
1017	6	0	6	344
1023	0	0	0	422
1025	14	4	10	224
1027	16	16	0	90
1029	0	0	0	440
1035	20	20	0	506
1037	6	4	2	76
1041	0	0	0	348
1043	0	0	0	154
1045	34	16	18	342
1047	0	0	0	350
1053	2	0	2	404
10001	30	20	10	208
10003	16	16	0	1434
10005	6	0	6	5076
10011	68	68	0	3570
10013	2	0	2	1372
10015	0	0	0	2006
10017	6	0	6	4400
10019	0	0	0	274
10021	48	48	0	920
10023	0	0	0	3878

n	SoSt	RaMi	Diff.	NTeil
10025	14	4	10	2024
10027	160	160	0	306
10029	0	0	0	3344
10031	0	0	0	1438
10033	16	16	0	204
10035	0	0	0	4706
10041	0	0	0	3348
10043	0	0	0	1022
10045	22	4	18	3324
10047	0	0	0	3774
10049	6	4	2	784
100001	48	48	0	9100
100005	6	0	6	48036
100007	0	0	0	1126
100009	430	52	378	21384
100011	0	0	0	40106
100013	0	0	0	1072
100015	4	4	0	21294
100017	6	0	6	33344
100021	6	4	2	3476
100023	0	0	0	48182
100025	14	4	10	2002
100027	0	0	0	4370
100029	0	0	0	33344
100031	0	0	0	1558
100033	0	0	0	764
100035	0	0	0	53378
100037	2	0	2	17236
100039	0	0	0	1478
100041	0	0	0	33348
100045	6	0	6	32204
100047	0	0	0	33350
1000001	4998	3748	1250	10000
1000005	6	0	6	471236
1000007	96	96	0	34510
1000009	6	4	2	3704
1000011	0	0	0	333338
1000013	2	0	2	147388
1000015	0	0	0	200006
1000017	2	0	2	362456
1000019	0	0	0	21322
1000021	48	48	0	90920
1000023	0	0	0	333342
1000025	126	16	110	308824
1000027	1456	1456	0	206874
1000029	2	0	2	354908
1000031	48	48	0	24430
1000035	12	12	0	475874
1000041	2	0	2	428672
1000043	0	0	0	97162
1000045	6	4	2	200012
1000047	0	0	0	333350
1000049	126	84	42	3184

Tabelle 3.1: Anzahl der Fehlversuche bei den probabilistischen Primtests von Solovey/Strassen (SoSt) und Rabin/Miller (RaMi). „Diff“ gibt die Differenz der beiden Zahlen an und „NTeil“ die Anzahl der Zahlen a mit $\mathrm{ggT}(a, n) \neq 1$.

Neben den deutlich kleineren Zahlen für die Fehlversuche der Primtests, als die bewiesenen Schranken angeben, verdeutlicht die letzte Spalte von Tabelle 3.1 an den relativ kleinen n auch recht gut, daß eine falsche Antwort eines der probabilistischen Primtests unwahrscheinlicher ist, als zufällig einen Teiler von n zu wählen. Eine solche, wenn auch von den Beispielen her wahrscheinliche Aussage, ließ sich bisher allerdings nicht beweisen.

Kapitel 4

Faktorisierungsalgorithmen

4.1 Faktorisierungsalgorithmen mit Gruppen

Den beiden in diesem Abschnitt vorgestellten Algorithmen liegt eine gemeinsame Idee zugrunde. Der chinesische Restesatz besagt, daß für zusammengesetztes n der Restklassenring $\mathbb{Z}_n$ isomorph ist zu einem direkten Produkt von kleineren Restklassenringen. Wir definieren eine algebraische Struktur, die von dem Restklassenring $\mathbb{Z}_n$ abhängt. Wenn wir in dieser Struktur rechnen, so rechnen wir implizit auch in der algebraischen Struktur, die gewissermaßen von den kleineren Komponenten des direkten Produkts abhängt, und erhalten darüber Information über eine der kleineren Strukturen und können dadurch einen Faktor von n abspalten. Im Falle der $p-1$-Methode ist die algebraische Struktur $\mathbb{Z}_n$ selbst und bei der Faktorisierung mit elliptischen Kurven eine elliptische Kurve $E(\mathbb{Z}_n)$ über dem Restklassenring $\mathbb{Z}_n$.

4.1.1 Pollard $p-1$-Methode

Der Aufwand dieses Algorithmus hängt im wesentlichen von der Größe des zu findenden Faktors ab. Nehmen wir also an, die Zahl n, die faktorisiert werden soll, enthält einen Primfaktor p, der die Eigenschaft besitzt, daß $p-1$ in kleine Faktoren zerfällt. Konkret sollen alle Faktoren von $p-1$ kleiner als eine Schranke k sein, und es soll darüberhinaus gelten, daß $k!$ durch $(p-1)$ teilbar ist.

Da die Exponentiation modulo n effizient berechnet werden kann, kann die Berechnung von

$$m :\equiv 2^{k!} \pmod{n}$$

für nicht zu große k durchgeführt werden. Der kleine Fermat liefert wegen $p-1 \mid k!$:

$$m \equiv 2^{k!} \equiv 2^{k! \pmod{p-1}} \equiv 2^0 \equiv 1 \pmod{p}.$$

Damit teilt p also $m-1$. Falls k nicht zu groß angesetzt ist, teilt n mit einer hohen Wahrscheinlichkeit $m-1$ nicht, so daß der größte gemeinsame Teiler

$$g := \mathrm{ggT}(m-1, n)$$

den nicht trivialen Teiler p von n liefert.

Bei der praktischen Anwendung dieser Vorgehensweise weiß man natürlich nicht, in welcher Größenordung der richtige Parameter k gewählt werden sollte. Damit nicht alle Primfaktoren von n erfaßt werden, ist es wichtig von Zeit zu Zeit $\mathrm{ggT}(m-1, n)$ zu berechnen. Falls der ggT gleich 1 ist, so erhöht man k und rechnet weiter.

Sollte $m \equiv 1 \bmod n$ sein, so sind alle Primfaktoren von n schon erreicht worden, so daß man entweder mit kleinerem k oder mit einer anderen Basis als 2 einen neuen Versuch macht.

Der Algorithmus führt nicht zum Erfolg, falls die Primfaktoren p_i von n eine ähnliche Zerlegung für $p_i - 1$ haben. Dann gilt $(p_i - 1) \mid k!$ für alle Faktoren p_i gleichzeitig. In diesem Fall wird ganz n abgespalten.

Verfeinerung des Algorithmus

Anstatt $k!$ als Exponent zu verwenden, ist es günstiger eine etwas andere Wahl zu treffen, denn für sehr kleine Primzahlen q taucht q in $k!$ mit einem sehr hohem Exponenten auf. Es ist aber unwahrscheinlich, daß dies für die Bedingung $p - 1 \mid k!$ benötigt wird. In der Praxis kann man beispielsweise so vorgehen, daß man bis zu einer gewissen Schranke B_1 alle Primzahlen und Primpotenzen kleiner B_1 betrachtet. An Stelle von $k!$ wählt man dann das Produkt dieser Primzahlen und Primpotenzen. Das Produkt sei l.

Man wählt eine weitere Schranke $B_2 > B_1$ und testet zusätzlich zu

$$\mathrm{ggT}(2^l - 1 \bmod n, n)$$

auch noch

$$\mathrm{ggT}(2^{l \cdot q_i} - 1 \bmod n, n)$$

für alle Primzahlen q_i mit $B_1 < q_i < B_2$, um die Primfaktoren zu trennen.

Laufzeitbetrachtungen

Die Laufzeit dieses Algorithmus hängt nicht nur von der Größe der zu faktorisierenden Zahl n ab, sondern wird auch stark von der Zerlegung von $p - 1$ für alle Primfaktoren p beeinflußt. Die Primfaktorzerlegung von n ist im voraus nicht bekannt. Daher kann man die Laufzeit des Algorithmus nur mit einer Wahrscheinlichkeitsbetrachung abschätzen. Es sei p der kleinste Primfaktor von n. Die Laufzeit hängt dann nur von der größten Primpotenz von $p - 1$ ab, sofern p, wie im allgemeinen zu erwarten ist, als erster Faktor gefunden wird.

Betrachten wir $p-1$ als zufällig gewählte natürliche Zahl, dann hat der größte Primfaktor im Mittel eine Größenordung von $(p-1)^{0.63}$. Falls also die größte Primpotenz kleiner der vorgegeben Schranke B ist, so führt der Algorithmus zum Erfolg, wenn die Primfaktoren weit genug auseinander liegen.

Für den ursprünglichen Algorithmus ergibt sich unter der Voraussetzung für die Zerlegung von $p-1$ die Beziehung zu k und damit die direkte Beziehung zur Laufzeit des Algorithmus ($O(\log_2(k!))$-Multiplikationen modulo n):

k	100	1 000	10 000	100 000	1 000 000
$\log_2(k!)$	525	8 530	118 000	1 500 000	18 500 000
$p \approx$	1 500	57 800	2 200 000	85 000 000	3 300 000 000

Ein k in der Größenordnung von 1 000 000 ist sogar bei der einfachsten Version des Algorithmus kein Problem. Damit können routinemäßig 10-stellige Faktoren gefunden werden. Durch Verwenden der Verfeinerung lassen sich auch 20-stellige Faktoren finden. Das Problem dieser Methode ist jedoch, daß sie in ungünstigen Fällen kein Ergebnis liefert und zudem nicht besonders gut parallelisierbar ist. Ein etwas besseres Verhalten in diesen beiden Punkten weist die nächste Methode auf.

4.1.2 Faktorisierung mit elliptischen Kurven

Der Ansatz, mit elliptischen Kurven zu faktorisieren, geht auf H.W. Lenstra zurück. Dieser Ansatz beruht wie die $p-1$-Methode von Pollard darauf, Zyklen in kleinen Untergruppen zu finden. Der Aufwand beim Faktorisieren mit elliptischen Kurven ist im wesentlichen auch von der Größenordnung des gesuchten Primfaktors abhängig.

Elliptische Kurven über Körpern

Es sei $\mathbb{K}$ ein Körper. Im Hinblick auf die betrachteten Anwendungen genügt es, Körper $\mathbb{K}$ der Charakteristik $\mathrm{char}(\mathbb{K}) \neq 2, 3$ zu betrachten. Dann lassen sich alle elliptischen Kurven beschreiben durch die Gleichungen der Form

$$y^2 = x^3 + Ax + B, \tag{4.1}$$

wobei $A, B \in \mathbb{K}$ mit $4A^3 + 27B^2 \neq 0$. Diese Nebenbedingung sichert, daß das Polynom $x^3 + Ax + B$ keine doppelten Nullstellen enthält.

Für $\mathbb{K} = \mathbb{R}$ lassen sich die elliptischen Kurven geometrisch veranschaulichen. Interessant werden die elliptischen Kurven $E(\mathbb{R})$ dadurch, daß man auf ihnen in einem abstrakten Sinne addieren kann. Zu je zwei Punkten P_1 und P_2 der Kurve findet man durch eine geometrische Konstruktion stets einen dritten, der ebenfalls auf der Kurve liegt und $P_1 + P_2$ genannt wird. Die so definierte Addition folgt den üblichen Rechenregeln: Sie ist assozitativ, kommutativ, es existiert ein neutrales Element, und zu jedem Punkt existiert ein Inverses. Eine elliptische Kurve bildet also eine kommutative Gruppe bezüglich dieser Addition.

Das Inverse $-P$ eines Kurvenpunktes P ist dessen Spiegelbild an der x-Achse. Die Summe zweier Punkte P_1 und P_2 findet man im allgemeinen, indem man die Gerade durch die

Punkte P_1 und P_2 zieht. Sie schneidet die Kurve in genau einem dritten Punkt Q. Dessen Spiegelbild $-Q$ bezüglich der x-Achse ist die gesuchte Summe $P_1 + P_2$.

In dem Sonderfall $P_1 = P_2$ tritt an die Stelle der Geraden durch P_1 und P_2 die Tangente an die Kurve im Punkt $P_1 = P_2$. Wenn schließlich $P_1 = -P_2$ ist, trifft die Gerade durch P_1 und P_2 keinen weiteren Punkt der Kurve. Man stellt sich ersatzweise vor, daß die Gerade die Kurve im Unendlichen schneidet. Darum fügt man noch einen weiteren Punkt hinzu, den man sich im Unendlichen denkt. Man nennt ihn $\mathcal{O}$ und setzt $P_1 - P_1 = \mathcal{O}$. Der Punkt $\mathcal{O}$ ist das neutrale Element der additiven Gruppe $E(\mathbb{R})$.

Die geometrische Defintion der Punktaddition kann in Formeln umgesetzt werden. Wenn $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$ Punkte der elliptischen Kurve sind, dann gilt

$$-P_1 = (x_1, -y_1), \tag{4.2}$$

$$P_1 + P_2 = \begin{cases} \mathcal{O} & \text{falls } P_1 = -P_2 \\ P_2 & \text{falls } P_1 = \mathcal{O} \\ P_1 & \text{falls } P_2 = \mathcal{O} \end{cases} \tag{4.3}$$

In den anderen Fällen ist die Summe $P_3 = P_1 + P_2 = (x_3, y_3)$ definiert durch

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda\,(x_1 - x_3) - y_1 \end{aligned} \tag{4.4}$$

wobei

$$\lambda = \frac{y_1 - y_2}{x_1 - x_2}$$

falls $P_1 \neq P_2$, und

$$\lambda = \frac{3x_1^2 + A}{2y_1}$$

falls $P_1 = P_2$.

Diese Formeln können nun eine Art Eigenleben gewinnen. Man darf sie anwenden, ohne sich unter x und y Koordinaten von Punkten in der Ebene vorstellen zu müssen. Es kann ein beliebiger endlicher Körper $\mathbb{F}$ verwendet werden. Geraden und Tangenten haben dann nicht mehr viel mit geometrischer Anschauung zu tun.

Betrachten wir nun näher die elliptischen Kurven über Primkörpern $\mathbb{F}_p$. Offensichtlich ist $E(\mathbb{F}_p)$ endlich. Man schreibt üblicherweise die Anzahl der Elemente in der Form

$$\#E(\mathbb{F}_p) = p + 1 - t,$$

wobei t als die Spur bezeichnet wird. Für die Spur gilt die sogenannte Hasse-Schranke

$$-2\sqrt{p} < t < 2\sqrt{p}.$$

Die Gruppe $E(\mathbb{F}_p)$ ist entweder eine zyklische Gruppe oder das Produkt zweier zyklischer Gruppen.

Elliptische Kurven über Ringen

Die Definition einer elliptischen Kurve über einem Restklassenring ist ganz analog zur Definition der elliptischen Kurven über einem Körper.

Definition 4.1. Eine elliptische Kurve über dem Restklassenring $\mathbb{Z}_n$ ist definiert als die Menge der Paare $(x, y) \in \mathbb{Z}_n \times \mathbb{Z}_n$, die die Weierstrass-Gleichung

$$y^2 = x^3 + Ax + B$$

mit A,$B \in \mathbb{Z}_n$ und $4A^3 + 27B^2 \neq 0$ erfüllen, zusammen mit einem Punkt im Unendlichen $\mathcal{O}$. Sie wird mit $E(\mathbb{Z}_n)$ bezeichnet. Wir setzen dabei voraus, daß 6 eine Einheit im Ring $\mathbb{Z}_n$ ist.

Auf dieser Punktmenge wird analog zur Addition auf elliptischen Kurven über Körpern eine Operation definiert. Eine direkte Verwendung der oben definierten Additionsformeln ist nicht möglich, da die dort auftretenden inversen Elemente nicht notwendigerweise existieren, da $\mathbb{Z}_n$ nicht nullteilerfrei ist, falls n nicht prim ist.

Es sei zur Vereinfachung n als quadratfrei vorausgesetzt, also $n = p_1 \cdot \ldots \cdot p_r$ für paarweise verschiedene Primzahlen p_i für $i = 1, \ldots, r$. Dann existieren die Abbildungen

$$h_i : E(\mathbb{Z}_n) \longrightarrow E(\mathbb{Z}_{p_i}),$$

die komponentenweise mittels der Projektionen

$$\pi_i : \mathbb{Z}_n \longrightarrow \mathbb{Z}_{p_i}$$

für $i = 1, \ldots, r$ definiert sind, wobei wir annehmen, daß $4A^3 + 27B^2 \neq 0 \pmod{p_i}$ ist. Durch Zusammensetzen der h_i erhält man eine Abbildung

$$h : E(\mathbb{Z}_n) \longrightarrow E(\mathbb{Z}_{p_1}) \times \ldots \times E(\mathbb{Z}_{p_r}),$$

die den gesamten Zielraum ausschöpft, bis auf alle Tupel, in denen das neutrale Element einer der Kurven $E(\mathbb{Z}_{p_i})$ vorkommt und die nicht das Tupel $(\mathcal{O}, \ldots, \mathcal{O})$ sind. Dabei definieren wir die Abbildung h so, daß das neutrale Element $\mathcal{O}$ auf das Tupel bestehend aus den neutralen Elementen der Kurven $E(\mathbb{Z}_{p_1})$ bis $E(\mathbb{Z}_{p_r})$ abgebildet wird. Der Chinesische Restesatz liefert, daß allen Elementen im Wertebereich dieser Funktion ein Urbild in eindeutiger Weise zugeordnet werden kann. Für diese Fälle ist damit auch eine Operation auf $E(\mathbb{Z}_n)$ eindeutig definiert, indem man die zu verknüpfenden Elemente zunächst mittels h nach $E(\mathbb{Z}_{p_1}) \times \ldots \times E(\mathbb{Z}_{p_r})$ abbildet, dort koordinatenweise addiert und gegebenfalls dem so erhaltenen Tupel wieder sein Urbild unter h zuordnet.

Die so definierte Operation kann auch durch Rechnen modulo n direkt in $\mathbb{Z}_n$ ausgeführt werden. Da diese Verknüpfung, falls sie definert ist, der Addition auf elliptischen Kurven über Körpern entspricht, bezeichnet man sie als *Pseudoadditon*. Die mit dieser Operation versehene Kurve $E(\mathbb{Z}_n)$ bildet offensichtlich keine Gruppe, da die Operation nicht total definiert ist.

Sei $a_i := \mathrm{ord}_{p_i}(P)$ die Ordnung des Punktes P auf der Kurve $E(\mathbb{Z}_{p_i})$, dann gilt

$$a_{i_0} \cdot h(P) = (P_1, \ldots, P_{i_0-1}, \mathcal{O}, P_{i_0+1}, \ldots, P_r)$$

für ein $i_0 \in \{1, \ldots, r\}$. Falls a_{i_0} kein Vielfaches der Ordnungen $\mathrm{ord}_{p_j}(P)$ des Punktes P auf allen Kurven $E(\mathbb{Z}_{p_j})$ mit $j = 1, \ldots, r$ ist, so ist h^{-1} an diesem Punkt nicht definiert. Dies gilt offensichtlich wenn a_{i_0} kein Vielfaches von $\mathrm{ord}_n(P) = \mathrm{kgV}\left(\mathrm{ord}_{p_1}(P), \ldots, \mathrm{ord}_{p_r}(P)\right)$ ist. Dies ist die Eigenschaft, die für die Faktorisierung ausgenutzt wird.

Man führt beim Faktorisieren mit elliptischen Kurven eine Berechnung aus, die sicher gelingen würde, falls n eine Primzahl wäre. Weil n aber zusammengesetzt ist, kann die Rechnung scheitern. Im Verlauf der Berechnung muß man inverse Elemente berechnen, wozu man den erweiterten Euklidschen Algorithmus zum Berechnen des größten gemeinsamen Teilers mit n verwendet. Die Rechnung kann nur dann fortgesetzt werden, wenn der größte gemeinsame Teiler gleich 1 ist, also die Zahlen teilerfremd sind.

Die Idee bei diesem Faktorisierungsalgorithmus ist es die Gruppe $\mathbb{Z}_n^*$, die bei der $p-1$-Methode von Pollard verwendet wird, durch die allgemeinere Gruppe der rationalen Punkte über den elliptischen Kurven zu ersetzen. Auch hier ist der Algorithmus erfolgreich, falls $\#E(\mathbb{F}_p)$ in kleine Primfaktoren zerfällt. Der wesentliche Vorteil liegt jedoch darin, daß zu einem vorgegeben n verschiedene Gruppen mit verschiedener Ordnung aus dem Intervall

$$[p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}]$$

zur Verfügung stehen.

Algorithmus

Für den Algorithmus kann man konkret wie folgt vorgehen. Zu einer Schranke k wählt man ähnlich wie bei der $(p-1)$-Methode von Pollard einen Multiplikator $m := k!$ oder einen ähnlichen Multiplikator, der nur aus Faktoren kleiner k besteht. Dann wählt man sich eine elliptische Kurve $E(\mathbb{Z}_n)$ und einen Punkt P auf der Kurve. In der Praxis kann man dabei so vorgehen, daß man zunächst nur den Kurvenparameter A vorgibt, dann einen Punkt P zufällig wählt und dann den Kurvenparameter B so bestimmt, daß der Punkt P auf der Kurve liegt. Dann berechnet man $m \cdot P$ auf der Kurve. Diese Multiplikation kann sehr effizient berechnet werden, da es nicht nötig ist, m mal P zu addieren. Man kann ähnlich wie bei „square and multiply“ vorgehen. Die Additionen führt man wie in der definierten Pseudoaddition durch. Bei der Berechnung müssen Inverse in $\mathbb{Z}_n$ bestimmt werden, die in $\mathbb{Z}_n$ nicht immer existieren müssen. Ein Nullteiler hat einen nichttrivialen Teiler mit n. In diesem Fall liefert die Berechnung des größten gemeinsamen Teilers einen Teiler von n.

Der Zufall spielt bei diesem Algorithmus wirklich mit, da es sehr viele elliptische Kurven zur Auswahl gibt, an denen man die Berechnungen durchführen kann. Lenstra konnte zeigen, daß es für jedes zusammengesetzte n Kurven gibt, die einen Teiler liefern. Falls man keinen nichttrivialen Teiler findet, kann man ein neues k bestimmen oder eine neue elliptische Kurve wählen.

Der Aufwand dieses Faktorisierungsalgorithmus läßt sich unter gewissen bisher unbewiesenen heuristischen Annahmen wie folgt abschätzen. Es sei p der kleinste Faktor der zu faktorisierenden Zahl n. Es sei ferner g eine beliebige natürliche Zahl. Dann findet der Algorithmus mit Wahrscheinlichkeit $\geq 1 - e^{-g}$ den Faktor p und hat einen asymptotischen Aufwand von

$$g \cdot \exp\left(\sqrt{(2 + o(1))(\log p \log\log p)}\right) \cdot (\log n)^2$$

für $p \to \infty$.

Der Algorithmus eignet sich in hervorragender Weise zur Parallelisierung. Man wählt für jeden zur Verfügung stehenden Prozessor eine elliptische Kurve und rechnet auf allen Kurven solange parallel, bis ein Faktor gefunden wurde. Dabei ist nur minimale Vorbereitung und Nachbereitung auf jedem der Prozessoren notwendig.

4.2 Faktorisierung mit quadratischen Kongruenzen

Der Aufwand der Faktorisierungsalgorithmen mit Gruppen hängt von der Größe der Primfaktoren der zu faktorisierenden Zahl n ab. Für die Suche nach größeren Teilern benutzt man andere Algorithmen, die auf einer anderen Grundidee basieren. Die Laufzeit dieser Algorithmen hängt nur von n ab. Wir werden hier zwei Vertreter dieser Algorithmenklasse kennenlernen.

Idee:
Man will eine zusammengesetzte Zahl n faktorisieren, indem man eine quadratische Kongruenz der Art

$$x^2 \equiv y^2 \pmod{n} \tag{4.5}$$

aufstellt. Diese Kongruenz ist gleichbedeutend mit der Aussage, daß n die Differenz $x^2 - y^2$ teilt. Nach der dritten binomischen Formel ist

$$x^2 - y^2 = (x - y)(x + y).$$

Wenn n nicht selbst Teiler von $x - y$ oder $x + y$ ist, so muß ein echter Teiler von n in $x - y$ und ein anderer echter Teiler in $x + y$ enthalten sein. Durch die Berechnung von $\mathrm{ggT}(x + y, n)$ kann dann ein nicht trivialer Faktor gefunden werden.

Die Tatsache, daß sich über solche Kongruenzen ein Faktor finden läßt, ist schon länger bekannt. Sie läßt sich praktisch aber nur nutzen, wenn man geeignete Verfahren kennt, solche Kongruenzen zu finden. Dies wurde für nicht triviale Fälle erst durch die Verwendung von leistungsfähigen Computern möglich.

Die Wahrscheinlichkeit, mit der das obige Verfahren zum Erfolg führt, kann mit dem folgenden Lemma genauer angegeben werden:

Lemma 4.2. *Es seien $x, y \in \mathbb{Z}_n$ mit $x^2 \equiv y^2 \bmod n$ gegeben, wobei n quadratfrei sei. Dann gilt mit Wahrscheinlichkeit $1 - \frac{1}{2^{t-1}}$, daß $x \not\equiv \pm y \bmod n$, wobei t die Anzahl der Faktoren von n ist.*

Beweis: Übung □

Als nächstes Ziel werden wir den relativ einfachen Algorithmus von Dixon betrachten, an dem sich eine ganze Reihe von grundlegenden Ideen besonders gut erklären lassen. Er nimmt sofern eine Sonderstellung ein, als daß für diesen Algorithmus der subexponentielle Aufwand mit mathematischer Strenge bewiesen werden kann, während für die meisten anderen Varianten der Aufwand immer unter Verwendung unbewiesener, lediglich auf heuristische Annahmen gestützte Vermutungen abgeschätzt wird.

4.2.1 Grundlagen der Laufzeitanalyse

Bevor wir uns den eigentlichen Algorithmen zuwenden, führen wir einige Definitionen ein und stellen einige Sätze vor, die uns die Abschätzung der Laufzeit der Faktorisierungsalgorithmen erleichtern werden.

Wir benutzen im folgenden die Abkürzung

$$L(n) := \exp\left(\sqrt{\log n \log\log n}\right). \tag{4.6}$$

Ferner benutzen wir die Abkürzung L^α für die Funktionenklasse

$$L(n)^{\alpha+o(1)}. \tag{4.7}$$

Lemma 4.3. *Für diese Funktionenklasse gelten die folgenden Rechenregeln:*

- $2L^\alpha = L^\alpha$;
- $L^\alpha + L^\beta = L^{\max\{\alpha,\beta\}}$.

Beweis: Übung □

Lemma 4.4. *Für die Funktion $\pi(x)$, die die Anzahl der Primzahlen kleiner $x \in \mathbb{R}$ angibt, gilt*

$$\pi(L^\alpha) = L^\alpha \tag{4.8}$$

Beweis: Übung □

Eine wichtige Formel, die bei der Laufzeitanalyse benötigt wird, ist die Abschätzung der Anzahl der natürlichen Zahlen, die nur Primfaktoren kleiner oder gleich einer vorgegebenen Schranke haben. Solche Zahlen werden als *glatt* (engl. *smooth*) bzw. *b-glatt* bezeichnet, wobei b die Schranke für die Größe der Primfaktoren ist. Dieser Begriff wurde von R. Rivest eingeführt.

Definition 4.5. Die Funktion $\Psi(x, y)$ gibt die Anzahl der Elemente $n < x$ an, die glatt bezüglich der Schranke y sind:

$$\Psi(x, y) := \#\{n \in \mathbb{N} \mid n < x \text{ und } ((p \mid n, p \text{ prim }) \Rightarrow p \leq y)\}. \tag{4.9}$$

Lemma 4.6. *Es gilt*

$$\Psi(n, L^{\alpha}) = n \cdot L^{-\frac{1}{2\alpha}}. \tag{4.10}$$

4.2.2 Dixons Algorithmus

Der Algorithmus von Dixon ist für praktische Zwecke nicht brauchbar, aber verschiedene Grundideen können anhand dieses Algorithmus besonders einfach erklärt werden. Außerdem handelt es sich um den historisch ersten Algorithmus, für den eine subexponentielle Laufzeit exakt nachgewiesen werden konnte.

Beschreibung des Algorithmus

Es sei n eine zusammengesetzte Zahl, die nicht durch Primzahlen $p < L(n)$ teilbar ist. Zu $m \in \mathbb{Z}$ bezeichne $Q(m)$ den kleinsten nicht-negativen Rest von $m^2 \bmod n$. Es sei a eine Konstante $0 < a < 1$, die wir später optimal festlegen werden. Um eine quadratische Kongruenz modulo n aufzustellen, benötigen wir den folgenden Teilalgorithmus:

1. Wähle $m \in \{1, \ldots, n-1\}$ zufällig aus und berechne $Q(m)$.
2. Teste, ob $Q(m)$ nur Faktoren $p \leq L(n)^a$ besitzt. Falls dies der Fall ist, so faktorisiere $Q(m)$ vollständig und trage m, $Q(m)$ und das Tupel $(e_1, \ldots, e_r)$, wobei $Q(m) = \prod_{i=1}^{r} p_i^{e_i}$ und $p_1, \ldots, p_r$ die Primzahlen kleiner $L(n)^a$ sind, in einer Tabelle ein.

Dieser Teilalgorithmus wird so oft ausgeführt, bis $\pi(L(n)^a)+1$ Quadrate $Q(m)$, die $L(n)^a$-glatt sind, gefunden wurden. Zu jedem Eintrag bilden wir einen Vektor

$$v(m) \in \mathbb{F}_2^{\pi(L^a)},$$

dessen i-te Komponente eine 0 oder eine 1 ist, je nachdem, ob die Primzahl p_i in gerader oder in ungerader Potenz in $Q(m)$ auftritt.

Da wir $\pi(L(n)^a) + 1$ viele Vektoren der Länge $\pi(L(n)^a)$ haben, existiert eine lineare Abhängigkeit unter diesen Vektoren. Wir rechnen über dem Skalarkörper $\mathbb{F}_2$, das heißt uns interessiert nur, ob der Exponent gerade oder ungerade ist. Nach Umnumerierung können wir die lineare Abhängigkeit als

$$v(m_1) + v(m_2) + \ldots + v(m_k) = \mathbf{0}.$$

ausdrücken. Also ist das Produkt $Q(m_1) \cdot Q(m_2) \cdot \ldots \cdot Q(m_k)$ ein Quadrat modulo n und es existiert folglich ein $x \in \mathbb{Z}$ mit

$$Q(m_1) \cdot Q(m_2) \cdot \ldots \cdot Q(m_k) \equiv x^2 \pmod{n}.$$

Dieses x kann sehr einfach mit Hilfe der im 2-ten Schritt gespeicherten Tupel berechnet werden. Ferner läßt sich $y \equiv m_1 \cdot m_2 \cdot \ldots \cdot m_k \bmod n$ berechnen und es gilt hierfür:

$$y^2 \equiv m_1^2 \cdot m_2^2 \cdot \ldots \cdot m_k^2 \equiv Q(m_1) \cdot Q(m_2) \cdot \ldots \cdot Q(m_k) \equiv x^2 \pmod{n}$$

Wir haben also eine quadratische Kongruenz modulo n aufgestellt. Durch Berechnen von $\mathrm{ggT}(x+y, n)$ erhalten wir mit Wahrscheinlichkeit größer oder gleich $1/2$ einen nichttrivialen Faktor von n. Falls $x \not\equiv \pm y \bmod n$ liefert die Berechnung von $\mathrm{ggT}(x+y, n)$ sicher einen nichttrivialen Faktor von n. Die genaue Erfolgswahrscheinlichkeit wurde in Lemma 4.2 in Abhängigkeit der Anzahl der Faktoren von n bereits berechnet.

Um die Erfolgswahrscheinlichkeit bei diesem Algorithmus zu erhöhen, berechnet man anstatt $\pi(L(n)^a) + 1$ beispielsweise $\pi(L(n)^a) + 10$ solche Einträge und erhält damit einen 10-dimensionalen Lösungsraum, also 2^{10} verschiedene lineare Abhängigkeiten, von denen dann eine mit sehr hoher Wahrscheinlichkeit zu der gesuchten Zerlegung von n führt.

Laufzeitanalyse

Wir führen jetzt eine grobe Laufzeitanalyse durch, bei der auch der Parameter a optimal bestimmt wird.

In dem Teilalgorithmus müssen wir für $\pi(L^a) = L^a$ viele Primzahlen testen, ob sie $Q(m)$ teilen. Wir definieren $b := b(n)$ indirekt dadurch, daß wir sagen, wir benötigen L^b Durchläufe des Teilalgorithmus, bis wir $\pi(L^a) + 1$ viele geeignete Einträge gefunden haben. Dies ergibt einen Aufwand von L^{a+b} Schritten, bis wir genügend viele gute $Q(m)$ gefunden haben. Die Gaußelimination benötigt naiv implementiert L^{3a} Schritte. Die restliche Berechnung der Werte x, y und $\mathrm{ggT}(x + y, n)$ benötigt nur L^a viele Schritte. Also kann die gesamte Laufzeit des Algorithmus durch

$$L^{\max\{a+b, 3a\}}$$

abgeschätzt werden.

Wir müssen zur Bestimmung der Laufzeit die Funktion oder Konstante b ermitteln und a optimal wählen. Wir benutzen die heuristische Annahme, daß bei einer zufälligen Wahl von m auch das modulo n reduzierte Quadrat $Q(m)$ zufällig verteilt ist. Damit beträgt die Wahrscheinlichkeit, daß $Q(m)$ in Faktoren $p < L^a$ zerfällt,

$$\frac{\Psi\left(n, L^a\right)}{n},$$

da $\Psi(n, L^a)$ die Anzahl der L^a-glatten Zahlen kleiner n angibt und $\frac{1}{n}$ unter der heuristischen Annahme die Wahrscheinlichkeit jeder der möglichen Zahlen ist. Mit Lemma 4.6 sehen wir, daß

$$\frac{\Psi(n, L^a)}{n} = L^{-\frac{1}{2a}}$$

gilt. Um also $\pi(L^a) + 1 = L^a$ viele L^a-glatte $Q(m)$ zu finden, müssen wir

$$L^a \cdot \left(L^{-\frac{1}{2a}}\right)^{-1} = L^{a+\frac{1}{2a}}$$

viele Werte für m untersuchen. Also gilt für die Anzahl b der notwendigen Durchläufe des Teilalgorithmus

$$b = a + \frac{1}{2a},$$

und somit beträgt der Aufwand

$$L^{a+b} = L^{2a+\frac{1}{2a}}.$$

Die Funktion $2a + \frac{1}{2a}$ hat ihr Minimum bei $a = 1/2$. Dies ergibt eine Laufzeit für den gesamten Algorithmus von

$$L^{\max\{a+b,3a\}} = L^{\max\{2a+\frac{1}{2a},3a\}} = L^{\max\{2,1.5\}} = L^2,$$

das heißt es werden $L(n)^{2+o(1)}$ viele Schritte und ein Speicheraufwand von $L(n)^{1+o(1)}$ für die $L^a \times L^a$-Matrix benötigt.

Satz 4.7. *Der Faktorisierungsalgorthmus von Dixon benötigt asymptotisch eine Laufzeit von $L(n)^{2+o(1)}$ vielen Schritten und einen Speicherplatzbedarf von $L(n)^{1+o(1)}$ bit.*

Unser Beweis dieses Satzes ist nicht mathematisch exakt durchgeführt worden, da wir eine heuristische Annahme benutzt haben. Diese Annahme betrifft die Wahrscheinlichkeit, mit der ein $Q(m)$ für ein zufällig gewähltes m in Primfaktoren $p < L^a$ zerfällt. Die $Q(m)$ sind nicht zufällig gleichverteilt, was wir angenommen haben. Es läßt sich jedoch zeigen, daß innerhalb einer Abweichung, die in $L(n)^{o(1)}$ abgefangen werden kann, dies auch für die Quadrate $Q(m)$ zutrifft.

4.2.3 Quadratischer Siebalgorithmus

Ein weiterer wichtiger Ansatz, wie eine Kongruenz der Art

$$x^2 \equiv y^2 \pmod{n}$$

aufgestellt werden kann, geht auf Pomerance zurück und verwendet ein ähnliches Verfahren, wie es beim Sieb des Eratosthenes eingesetzt wird. Die Zahlen, die dem Siebprozeß unterworfen werden, sind Quadratzahlen, daher spricht man von einem *quadratischen Sieb*.

In dem quadratischen Siebalgorithmus gehen wir gegenüber der Funktion $Q(x)$ beim Dixon-Algorithmus von einem anderen quadratischen Polynom

$$f(x) = (x + \lfloor\sqrt{n}\rfloor)^2 - n \tag{4.11}$$

aus. Für dieses quadratische Polynom mit ganzzahligen Koeffizienten gilt, wie bei $Q(x)$, für beliebige $x \in \mathbb{Z}$

$$\left(x + \lfloor\sqrt{n}\rfloor\right)^2 \equiv f(x) \pmod{n}.$$

Finden wir $m_1, \ldots, m_k \in \mathbb{N}$, so daß

$$f(m_1) \cdot \ldots \cdot f(m_k)$$

ein Quadrat ist, so erhalten wir genau wie beim Dixon-Algorithmus mit

$$x^2 := f(m_1) \cdot \ldots \cdot f(m_k)$$

und

$$y := (m_1 + \lfloor\sqrt{n}\rfloor) \cdot \ldots \cdot (m_k + \lfloor\sqrt{n}\rfloor)$$

eine quadratische Kongruenz

$$x^2 \equiv y^2 \pmod{n},$$

mit der wir einen Faktor von n bestimmen können.

Um die Wahrscheinlichkeit einen Faktor zu finden zu erhöhen, sollte auch hier die Anzahl der Gleichungen, die in das lineare Gleichungssystem über $\mathbb{F}_2$ eingehen, etwas höher sein als die Anzahl der Unbestimmten plus eins.

Es sei B die Menge von Primzahlen, bzgl. derer wir die Werte $f(m)$ versuchen zu faktorisieren; wir bezeichnen diese Menge als *Faktorbasis*.

Eine geeignete Wahl für B ist sehr wichtig für die Laufzeit des Algorithmus.

- Wird B recht groß gewählt, so müssen zunächst viele $f(m)$ gefunden werden, bis genügend Gleichungen vorhanden sind, so daß das System linear abhängig wird. Dann muß dieses große Gleichungssystem gelöst werden.
- Wenn B recht klein gewählt wird, dann müssen zwar nur wenige m gefunden werden, für die $f(m)$ über B zerfällt, es wird durch die sehr wenigen Primzahlen, die in B zur Verfügung stehen, aber deutlich unwahrscheinlicher, ein solches m zu finden.

Heuristische Überlegungen legen nahe, daß die Größe von B in der Größenordnung

$$\exp\left(\frac{1}{2}\sqrt{\log n \log\log n}\right)$$

gewählt werden sollte.

Die Methode in der jetztigen Form eignet sich noch nicht zur Faktorisierung großer Zahlen. Durch Ausnutzen einiger Eigenschaften von $f(x)$ ergibt sich aber ein sehr effizienter Algorithmus:

Je kleiner eine natürliche Zahl $f(m)$ ist, um so größer ist die Wahrscheinlichkeit, daß $f(m)$ über einer vorgegebenen Faktorbasis B zerfällt. Daher sollte m so gewählt werden, daß $f(m)$ möglichst kleine Werte annimmt.

Für m aus dem Intervall $1 \leq m \leq n^{1/4}$ gilt für die Werte von $f(m)$

$$f(m) = \left(m + \lfloor\sqrt{n}\rfloor\right)^2 - n = 2m\lfloor\sqrt{n}\rfloor + m^2 + \left(\lfloor\sqrt{n}\rfloor^2 - n\right) \approx 2m\lfloor\sqrt{n}\rfloor + m^2 \in O(n^{3/4}).$$

Mit einem geeigneten Siebprozeß können wir auf wesentlich effektivere Weise geeignete Werte für m bestimmen. Wir konzentrieren uns dabei nicht auf einzelne Werte für m, sondern betrachten gleich eine ganze Reihe von Werten.

Dazu benötigen wir jedoch noch einige Vorüberlegungen. Zunächst ist klar, daß für einen Primfaktor p von $f(m_0)$ gilt, daß p auch Primfaktor von $f(m_0 + k \cdot p)$ für alle $k \in \mathbb{Z}$ ist. Um die Menge der Werte für m zu bestimmen, für die p ein Faktor von $f(m)$ ist, genügt es folglich, die quadratischen Kongruenzen

$$f(m) \equiv 0 \pmod{p}$$

für $m \in \{0, \ldots, p-1\}$ zu lösen.

Aus der speziellen Form des quadratischen Polynoms 4.11 sieht man, daß für $p \geq 2$, das nicht selbst Primfaktor von n ist, die Kongruenz

$$f(m) = (m + \lfloor\sqrt{n}\rfloor)^2 - n \equiv 0 \pmod{p} \tag{4.12}$$

genau dann eine (oder auch zwei) Lösungen besitzt, falls n ein Quadrat modulo p ist, d.h. das Legendresymbol $\left(\frac{n}{p}\right) = 1$. Falls n kein Quadrat modulo p ist, d.h. das Legendresymbol $\left(\frac{n}{p}\right) = -1$, existiert keine Lösung. In diesem Fall ist p kein Teiler von $f(x)$ für alle $x \in \mathbb{N}$. Wir wählen daher nicht wie bei dem Dixon-Algorithmus die ersten B Primzahlen, sondern $\{p_1, \ldots, p_B\}$, die ersten B Primzahlen mit $\left(\frac{n}{p_i}\right) = 1$ für $i = 1, \ldots, B$. Diese Menge ist unsere *Faktorbasis*.

Die quadratische Siebprozedur funktioniert nun wie folgt. Zu jeder Primzahl aus der Faktorbasis werden die beiden, nicht notwendigerweise verschiedenen Lösungen a_p, b_p der Kongruenz 4.12 berechnet. Wir betrachten nun nicht nur einzelne zufällig gewählte Werte für m, sondern wir untersuchen ein Intervall von T vielen aufeinanderfolgenden m-Werten.

Zur Vorbereitung berechnen wir zunächst für jedes m den Wert $\lfloor\log_2 f(m)\rfloor$, also fast immer die Anzahl bits minus 1, die für die Binärdarstellung von $f(m)$ benötigt werden. Dieser Wert ist für viele aufeinanderfolgende m-Werte gleich und daher effizient zu berechnen. Die m-Werte werden zusammen mit $\lfloor\log_2 f(m)\rfloor$ in einer Tabelle gespeichert.

Danach wird beim Sieben für jede Primzahl p aus der Faktorbasis der Wert $\lfloor \log_2 p \rfloor$ von dem Wert $\lfloor \log_2 f(m) \rfloor$ abgezogen, falls

$$m \equiv a_p \pmod{p} \text{ oder } m \equiv b_p \pmod{p}$$

gilt.

Für ein $f(m)$, das höhere Potenzen einer Primzahl p aus der Faktorbasis enthält, werden noch zusätzliche Tests benötigt. Das heißt für Primzahlen p mit $p^k < C$ muß für $i < k$

$$f(m) \equiv 0 \pmod{p^i}$$

getestet und gegebenfalls mehrfach $\lfloor \log_2 p \rfloor$ abgezogen werden.

Am Ende des Siebens wurde also von jedem Eintrag $\lfloor \log_2 f(m) \rfloor$ für jeden Primteiler p von $f(m)$ der Wert $\lfloor \log_2 p \rfloor$ entspechend oft abgezogen. Falls also $f(m)$ über der Faktorbasis zerfällt, ist dieser Eintrag innerhalb der Rechenungenaugigkeit für die Rundung etwa gleich Null.

Mit diesem Sieb-Verfahren kann also aus einem Intervall von T vielen aufeinanderfolgenden Werten für m eine Liste von Werten $f(m)$ erstellt werden, die mit hoher Wahrscheinlichkeit über der Faktorbasis zerfallen. Diese Werte werden dann noch einmal im Einzelnen betrachtet, da während des Siebverfahren die Faktoren nicht protokolliert wurden. Aber jeder dieser Werte ergibt mit hoher Wahrscheinlichkeit einen Vektor über $\mathbb{F}_2$, so daß auf die oben skizzierte Art und Weise über die lineare Abhängigkeit ein Faktor von n gefunden werden kann.

Eine auf heuristische Annahmen gestützte Laufzeitanalyse ergibt für diesen Algorithmus einen Aufwand von

$$L(n)^{1+o(1)} = \exp\left((1+o(1))\sqrt{\log n \log\log n}\right).$$

Kapitel 5

Diskrete Logarithmen

Für die meisten endlichen Gruppen (z. B. Einheitengruppe eines endlichen Körpers, Gruppe einer elliptischen Kurve) sind bis heute keine Algorithmen bekannt, die effizient den diskreten Logarithmus eines Gruppenelements einer zyklischen (Unter-)Gruppe berechnen. Damit wird das diskrete Logarithmusproblem neben dem Faktorisieren als eines der beiden bekannten harten Probleme, die sich für kryptographische Zwecke eignen, angesehen. Das Verfahren von ElGamal stützt sich auf diese Tatsache.

5.1 Einführung

Es sei G eine endliche zyklische Gruppe und α ein Erzeuger von G. Dann gilt

$$G = \left\{ \alpha^i \mid 0 \leq i < \#G \right\},$$

wobei $\#G$ die Gruppenordnung von G ist.

Der diskrete Logarithmus eines Elements β zur Basis α in G ist eine ganze Zahl i mit $\alpha^i = \beta$. Wir schreiben $i = \log_\alpha \beta$. Wenn i auf das Intervall $0 \leq i < \#G$ beschränkt ist, dann ist der diskrete Logarithmus von β zur Basis α eindeutig.

Man kann sofort zwei naive Algorithmen zur Berechnung der diskreten Logarithmen in einer endlichen zyklischen Gruppe G angeben. Der erste berechnet einmal eine Liste der Logarithmen aller Gruppenelemente. Der zweite berechnet so lange Potenzen von α, bis eine Übereinstimmung mit β auftritt. Diese trivialen Algorithmen sind wertlos, wenn $\#G$ groß ist.

5.2 Baby-Step-Giant-Step-Algorithmus

Der Baby-Step-Giant-Step-Algorithmus eignet sich, um Logarithmen in beliebigen zyklischen Gruppen zu berechnen. Er stellt eine erhebliche Verbesserung gegenüber den in der

Einführung beschriebenen trivialen Algorithmen dar, ist aber praktisch ebenfalls nicht einsetzbar, wenn die Gruppenordnung größer ist.

Wir wollen den diskreten Logarithmus $\log_\alpha \beta$ berechnen. Es sei $m = \lceil \sqrt{\#G} \rceil$.

1. Erstelle eine Liste der Paare (i, α^i) für $0 \leq i < m$ (offensichtlich gilt $i = \log_\alpha \alpha^i$) und sortiere diese Liste nach der zweiten Komponente.
2. Berechne $\beta\alpha^{-jm}$ für jedes j mit $0 \leq j < m$ und überprüfe (mit Binärsuche), ob dieses Element als zweite Komponente eines Listeneintrags vorkommt.
3. Falls $\beta\alpha^{-jm} = \alpha^i$ für ein i mit $0 \leq i < m$, dann gilt $\beta = \alpha^{i+jm}$ und $\log_\alpha \beta = i + jm$.

Der Algorithmus erstellt eine Liste mit $\mathrm{O}(m)$ Einträgen und benötigt $\mathrm{O}(m \log m)$ Operationen, um die Liste zu sortieren und dann in der Liste zu suchen. Mit einer Operation ist eine Gruppenoperation oder ein Vergleich gemeint. Steht genügend Speicherplatz zur Verfügung, so kann das Speichern und Suchen über eine Hash-Tabelle realisiert werden. Damit reduziert sich der Rechenaufwand auf $\mathrm{O}(m)$ Operationen. Der Name beschreibt in kennzeichnender Weise die Funktionsweise des Algorithmus.

Übung: Wie sieht die Giant-Step-Baby-Step-Version des Algorithmus aus?

5.3 Pohlig-Hellman-Algorithmus

Der Pohlig-Hellman-Algorithmus eignet sich ebenfalls, um Logarithmen in beliebigen zyklischen Gruppen zu berechnen. Er ist besonders effizient, falls die Gruppenordnung in kleine Primfaktoren zerfällt.

Wir wollen wieder den diskreten Logarithmus $x = \log_\alpha \beta$ berechnen. Wir nehmen an, daß G die Ordnung $\#G = p_1^{e_1} \dots p_r^{e_r}$ mit unterschiedlichen Primzahlen $p_1, \dots, p_r$ hat.

Wenn wir nun $x_i = x \bmod p_i^{e_i}$ für alle $i = 1, \dots, r$ kennen, kann man mithilfe des Chinesischen Restesatzes (Satz 2.5) x ausrechnen.

Wir betrachten ein festes i. Nun zerlegen wir $x_i < p_i^{e_i}$ in

$$x_i = l_0 + l_1 p_i + l_2 p_i^2 + \dots + l_{e_i-1} p_i^{e_i-1}$$

mit $0 \leq l_0, \dots, l_{e_i-1} < p_i$. Die Berechnung von x_i ist also äquivalent zur Berechnung von $(l_0, \dots, l_{e_i-1})$.

Es gilt

$$\begin{aligned} & \alpha^x &=& \beta \\ \Rightarrow \quad & \alpha^{x\frac{n}{p_i}} &=& \beta^{\frac{n}{p_i}} \\ \Leftrightarrow \quad & (\alpha^{\frac{n}{p_i}})^x &=& (\alpha^{\frac{n}{p_i}})^{l_0} = \beta^{\frac{n}{p_i}} \end{aligned}$$

Der erste Wert l_0 ist also der diskrete Logarithmus $l_0 = \log_{\alpha^{\frac{n}{p_i}}} \beta^{\frac{n}{p_i}}$ in der zyklischen Gruppe $\langle \alpha^{\frac{n}{p_i}} \rangle$ von Ordnung p_i. Der diskrete Logarithmus in dieser Gruppe kann durch vollständige Suche oder den Baby-Step-Giant-Step-Algorithmus berechnet werden.

Sobald $l_0, \ldots, l_{k-1}$ bekannt sind, kann l_k in ähnlicher Weise berechnet werden:

$$\begin{aligned} & & \alpha^{x_i \frac{n}{p_i^{k+1}}} &= \beta^{\frac{n}{p_i^{k+1}}} \\ \Leftrightarrow\ & \alpha^{(l_k p_i^k + l_{k+1} p_i^{k+1} + \cdots + l_{e_i-1} p_i^{e_i-1}) \frac{n}{p_i^{k+1}}} &= \beta^{\frac{n}{p_i^{k+1}}} \alpha^{-(l_0 + l_1 p_i + \cdots + l_{k-1} p_i^{k-1}) \frac{n}{p_i^{k+1}}} \\ \Leftrightarrow\ & \alpha^{(l_k p_i^k) \frac{n}{p_i^{k+1}}} = \alpha^{(\frac{n}{p_i}) l_k} &= \beta^{\frac{n}{p_i^{k+1}}} \alpha^{-(l_0 + l_1 p_i + \cdots + l_{k-1} p_i^{k-1}) \frac{n}{p_i^{k+1}}} \end{aligned}$$

Also ist

$$l_k = \log_{\alpha^{\frac{n}{p_i}}} \beta^{\frac{n}{p_i^{k+1}}} \alpha^{-(l_0 + l_1 p_i + \cdots + l_{k-1} p_i^{k-1}) \frac{n}{p_i^{k+1}}},$$

wieder in der zyklischen Gruppe $\langle \alpha^{\frac{n}{p_i}} \rangle$ von Ordnung p_i.

So können alle x_i für $i = 1, \ldots, r$ und schließlich x bestimmt werden. Der Aufwand des Pohlig-Hellman-Algorithmus ist $\mathrm{O}\left(\sum_{i=1}^{r} e_i (\log(n) + \sqrt{p_i})\right)$. Für ein schwieriges dlog-Problem sollte die verwendete Gruppe G daher Primordnung haben.

5.4 Index Calculus

Wir beschreiben generisch die Funktionsweise der Index Calculus Methode. Es sei G eine endliche zyklische Gruppe der Ordnung n, die durch α erzeugt wird. Wir wollen die Logarithmen der Gruppenelemente zu dieser Basis berechnen. Es sei $S = \{p_1, p_2, \ldots, p_t\}$ eine Untermenge von G mit der Eigenschaft, daß ein „bedeutender“ Teil der Gruppenelemente sich als Produkt der Elemente aus S schreiben läßt. Die Menge S wird als die *Faktorbasis* bezeichnet.

Im ersten Schritt versuchen wir die Logarithmen aller Elemente aus S zu bestimmen. Dazu wählen wir eine Zufallszahl a und versuchen α^a als Produkt der Elemente aus S zu schreiben:

$$\alpha^a = \prod_{i=1}^{t} p_i^{\lambda_i}. \tag{5.1}$$

Wenn wir so eine Darstellung finden können, erhalten wir aus 5.1 eine lineare Kongurenz

$$a \equiv \sum_{i=1}^{t} \lambda_i \log_\alpha p_i \pmod{n}. \tag{5.2}$$

Nachdem wir eine genügend große Anzahl (größer als t) an Relationen der Form 5.2 aufgestellt haben, können wir erwarten, daß das zugehörige lineare Gleichungssystem eine eindeutige Lösung für die Unbekannten $\log_\alpha p_i$ mit $1 \leq i \leq t$ besitzt.

Im zweiten Schritt berechnen wir individuelle Logarithmen in G. Es sei ein $\beta \in G$ gegeben, von dem wir den Logarithmus $x = \log_\alpha \beta$ bestimmen möchten. Wir wählen solange Zufallszahlen s, bis $\alpha^s \beta$ sich als Produkt von Elementen aus S schreiben läßt:

$$\alpha^s \beta = \prod_{i=1}^{t} p_i^{b_i}. \tag{5.3}$$

Es gilt dann

$$\log_\alpha \beta \equiv \sum_{i=1}^{t} b_i \log_\alpha p_i - s \pmod{n}.$$

Um die Beschreibung der Index Calculus Methode abzuschließen, müssen wir angeben, wie ein geeignetes S zu wählen ist und wie effizient die Relationen 5.1 und 5.3 zu erzeugen sind. Ein geeignetes S sollte nicht zu groß sein, das Gleichungssystem im ersten Schritt wird sonst sehr groß. Gleichzeitig muß die Anzahl der Gruppenelemente, die über S faktorisieren, groß sein, um die erwartete Zeit für die Aufstellung der Relationen 5.1 und 5.3 in Grenzen zu halten. Diese Berechnungen werden üblicherweise durch die Faktorisierung von Elementen durchgeführt. Da in einer Gruppe die Begriffe Faktorisierung und Primfaktor keinen Sinn geben, müssen wir hier zusätzliche Strukturen verwenden. Ein Beispiel sind endliche Körper, für die wir die Index Calcus Methode genauer beschreiben. Eine ausführliche Studie und Gegenüberstellung der verschiedenen Versionen kann in [6] nachgelesen werden. Der Algorithmus macht (notwendigerweise) Gebrauch von den unterschiedlichen Darstellungsmöglichkeiten der Körperelemente. Daher wird zunächst eine kurze Einführung in die Theorie der endlichen Körper gegeben.

5.4.1 Endliche Körper

Satz 5.1 (Restklassenkörper). *Für $m \in \mathbb{N}$ sind äquivalent:*

(i) m ist eine Primzahl und

(ii) $\mathbb{Z}/m\mathbb{Z}$ ist ein Körper.

Die endlichen Körper $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$, wobei p eine Primzahl ist, heißen *Primkörper*.

Definition 5.2 (Körpererweiterung). Unter einer Körpererweiterung versteht man ein Paar von Körpern $K \subset L$, wobei K ein Unterkörper von L ist. Bezeichnung: L/K oder $K < L$. Es heißt L dann Erweiterungskörper von K.

Satz 5.3 (Primkörper). *Jeder endliche Körper enthält einen Unterkörper mit p Elementen, wobei p eine Primzahl ist. Ein Körper mit primzahl vielen Elementen heißt endlicher Primkörper.*

Beweis: (Idee) Betrachte die durch mehrfache Addition von 1 erzeugte endliche Unterstruktur. Sie hat Primzahl viele Elemente (wegen der Nullteilerfreiheit) und erfüllt die Körperaxiome. □

Bemerkung 5.4. *Ein endlicher Körper hat p^m Elemente, wobei p eine Primzahl und $m \in \mathbb{N}_{\geq 1}$ ist.*

Beweis: Es sei L eine Körpererweiterung von K, $\#K = p$ mit p prim. Man kann die Multiplikation $L \times L \to L$ einschränken auf eine Multiplikation $K \times L \to L$ und auf diese Weise L als K-Vektorraum auffassen.

Damit ist L ein m-dimensionaler K-Vektorraum ($m < \infty$, da L endlich ist). Es sei $\{\omega_1, \ldots, \omega_m\}$ eine Basis von L. Jedes Element aus L läßt sich in eindeutiger Weise als Linerakombination $a_1\omega_1 + \ldots + a_m\omega_m$ mit $a_1, \ldots, a_m \in K$ schreiben. Die Anzahl der Elemente des Körpers L ist also p^m. □

Definition 5.5 (Zerfällungskörper). Ein Zerfällungskörper eines Polynoms $f \in K[x]$ ist ein Erweiterungskörper L des Körpers K, in dem f vollständig in Linearfaktoren zerfällt.

Satz 5.6 (Isomorphie der minimalen Zerfällungskörper). *Es seien L_1 und L_2 minimale Zerfällungskörper eines Polynoms $f \in K[x]$. Dann sind L_1 und L_2 isomorph.*

Satz 5.7. *Es sei p Primzahl, $n \geq 1$ und N die Menge der Nullstellen von $x^{p^n} - x \in \mathbb{F}_p[x]$ im Zerfällungskörper.*
Dann gilt: N ist ein Körper mit p^n Elementen.

Beweis: Es seien $\alpha, \beta \in N$, dann gilt $\alpha^{p^n} = \alpha$ und $\beta^{p^n} = \beta$. Damit berechnet man:

- $(\alpha \pm \beta)^{p^n} = \alpha^{p^n} \pm \beta^{p^n} = \alpha \pm \beta$
- $(\alpha \cdot \beta)^{p^n} = \alpha^{p^n} \cdot \beta^{p^n} = \alpha \cdot \beta$
- $\alpha^{p^n} = \alpha \Rightarrow \alpha^{p^n-2} = \alpha^{-1}$ da $p \geq 2$ ist $p^n - 2 \geq 0$ ist die Existenz von α^{-1} gesichert. $\alpha^{p^n-2} \in N$ ergibt sich direkt.

Weiter sind die neutralen Elemente $0, 1$ in N enthalten, so daß nur noch zu zeigen bleibt: $\sharp N = p^n$, also daß alle Nullstellen von $x^{p^n} - x$ verschieden sind. Betrachte dazu die Ableitung

$$\left(x^{p^n} - x\right)' = p^n \cdot x^{p^n-1} - 1 = -1 \neq 0.$$

Daher hat $x^{p^n} - x$ keine mehrfachen Nullstellen. □

Definition 5.8 (Irreduzibel). Ein Polynom $f(x) \in K[x] \setminus K$, wobei K ein Körper ist, heißt *irreduzibel*, falls für jede Zerlegung $f(x) = g(x)h(x)$ mit $g(x), h(x) \in K[x]$ gilt $g(x) \in K^*$ oder $h(x) \in K^*$.

Satz 5.9 (Konstruktion eines Erweiterungskörpers). *Es sei $f(x) \in \mathbb{F}_p[x]$ ein irreduzibles Polynom mit $\deg(f) = n$. Dann ist der Restklassenring $\mathbb{F}_p[x] \Big/ f(x)$ ein Körper mit p^n Elementen.*

Beweis: Es sind die Körperaxiome nachzuprüfen. Als Quotientenring ist die Abgeschlossenheit bzgl. Addition/Subtraktion gesichert und es bleibt lediglich die Existenz von Inversen zu zeigen.

Es sei $a(x) \in \mathbb{F}_p[x] \setminus \{0\}$ vom Grad kleiner n. Aus der Irreduzibilität von $f(x)$ läßt sich über den euklidschen Algorithmus $q_1(x), q_2(x) \in \mathbb{F}_p[x]$ berechnen mit

$$1 = \text{ggT}(f(x), a(x)) = q_1(x) \cdot f(x) + q_2(x) \cdot a(x) \equiv q_2(x) \cdot a(x) \bmod f(x),$$

was die Existenz von a^{-1} zeigt. □

Beispiel 5.10. Für die Konstruktion eines Körpers mit $4 = 2^2$ Elementen muß ein irreduzibles Polynom $f(x) \in \mathbb{F}_2[x]$ vom Grad 2 gefunden werden. Es genügt wegen des kleinen Grades für die Irreduzibilität zu überprüfen, daß $f(x)$ in $\mathbb{F}_2$ keine Nullstelle besitzt.

Das einzige Polynom, das weder 0 noch 1 als Nullstelle hat ist $f(x) = x^2 + x + 1 \in \mathbb{F}_2[x]$.

Die Elemente von $\mathbb{F}_{2^2}$ sind also $\{0, 1, x, x+1\}$. Die Addition und Multiplikation werden modulo $f(x)$ durchgeführt. Es gilt z. B. $x^2 = x + 1$ und $x \cdot (x + 1) = 1$.

Lemma 5.11. *Es sei $\alpha \in \mathbb{F}_{p^n}$, dann gilt $\alpha^{p^n} = \alpha$.*

Beweis: Für 0 ist die Gleichung offensichtlich erfüllt.
Alle Elemente $\alpha \neq 0$ bilden eine multiplikative Gruppe; nach dem Satz von Lagrange gilt also $\alpha^{p^n - 1} = 1$. □

Satz 5.12. *Es sei p Primzahl und $n \geq 1$. Dann existiert ein Körper mit p^n Elementen und alle Körper mit p^n Elementen sind isomorph.*

Beweis: Die Existenz wurde bereits in Satz 5.7 gezeigt. Sei $\alpha \in \mathbb{F}$, einem Körper mit p^n Elementen, dann gilt $\alpha^{p^n} = \alpha$. Daher läßt sich α als Nullstelle des Polynoms $x^{p^n} - x$ in den minimalen Zerfällungskörper von $x^{p^n} - x$ einbetten. Da minimale Zerfällungskörper bis auf Isomorphie eindeutig sind, ist die Behauptung bewiesen. □

Einen weiteren wichtigen Satz zitieren wir ohne Beweis:

Satz 5.13 (Struktur der multiplikativen Gruppe $\mathbb{F}_q^*$). *Die multiplikative Gruppe eines endlichen Körpers ist zyklisch. Ein Element $\omega \in \mathbb{F}_q$ mit $\langle \omega \rangle = \mathbb{F}_q^*$ heißt* primitives Element.

5.4.2 Index-Calculus für $\mathbb{F}_{p^n}$

Die Elemente des endlichen Körpers $\mathbb{F}_{p^n}$ werden als Polynome vom Grad kleiner n aus $\mathbb{F}_p[x]$ aufgefaßt. Die Addition ist gewöhnliche Polynomaddition und die Multiplikation wird modulo eines festgelegten irreduziblen Polynoms $f(x) \in \mathbb{F}_p[x]$ ausgeführt. Wir benutzen also die Isomorphie $\mathbb{F}_{p^n} \cong \mathbb{F}_p[x] \big/ (f(x))$.

Definition 5.14. Ein Polynom $w(x) \in \mathbb{F}_p[x]$ heißt *glatt* bezüglich der Schranke $b \in \mathbb{N}$, genau dann, wenn es in Polynome aus $\mathbb{F}_p[x]$ vom Grad kleiner oder gleich b faktorisiert.

Mit der Polynomdarstellung des Körpers $\mathbb{F}_{p^n}$ ist es naheliegend, die Faktorbasis S als die Menge aller irreduziblen Polynome aus $\mathbb{F}_p[x]$ vom Grad kleiner oder gleich b zu wählen. Im ersten Schritt werden die Logarithmen aller Elemente der Faktorbasis bestimmt.

Algorithmus:

Es sei $\alpha(x)$ ein Erzeuger von $\mathbb{F}_{p^n}^*$. Die Faktorbasis S wird als die Menge aller normierten, irreduziblen Polynome $p(x) \in \mathbb{F}_p[x]$ mit $\deg(p(x)) \leq b$ gewählt. Die Schranke b geben wir später an.

1. Vorberechnungsphase: Es werden die diskreten Logarithmen aller Elemente aus der Faktorbasis S berechnet.

   (a) Wähle $a_i \in \{1, \ldots, p^n - 1\}$ zufällig aus und berechne $y_i(x) := \alpha(x)^{a_i}$. Wir können effizient mit dem ggT-Algorithmus überprüfen, ob $y_i(x)$ in Faktoren $p_i(x) \in S$ zerfällt. Es werden alle maximalen Potenzen von $p_i(x) \in S$ von $y_i(x)$ abgespalten. Falls wir danach ein Polynom vom Grad $> b$ erhalten, wissen wir, daß $y_i(x)$ nicht b-glatt ist. Falls $y_i(x)$ b-glatt ist, so speichern wir die Faktorisierung

   $$y_i(x) = c_i \cdot \prod_{p_j(x) \in S} p_j(x)^{e_{ij}}$$

   mit $c_i \in \mathbb{F}_p$. Für die diskreten Logarithmen a_i gilt dann

   $$a_i = \log_{\alpha(x)} y_i(x) = \log_{\alpha(x)} c_i + \sum_{p_j(x) \in S} e_{ij} \cdot \log_{\alpha(x)} p_j(x) \tag{5.4}$$

   Wir wiederholen dies so lange, bis wir mehr als $\#S$ Gleichungen der Form 5.4 haben. Die Logarithmen der Elemente aus dem Grundkörper berechnen wir direkt durch ausprobieren.

   (b) Löse dieses lineare Gleichungssystem mit den Unbekannten $\log_{\alpha(x)} p_j(x)$ für $p_j(x) \in S$.

2. Berechnen eines konkreten Logarithmus von $y(x) = \alpha(x)^l$.

   Wähle s so lange zufällig aus, bis $\alpha(x)^s y(x)$ b-glatt ist. Das Polynom $\alpha(x)^s y(x)$ kann dann wie folgt

   $$\alpha(x)^s y(x) = c \cdot \prod_{p_j(x) \in S} p_j(x)^{e_j}$$

   mit $c \in \mathbb{F}_p$ faktorisiert werden. Daraus kann bereits der diskrete Logarithmus $\log_{\alpha(x)} y(x)$ berechnet werden, denn es gilt

   $$\log_{\alpha(x)} y(x) = \log_{\alpha(x)} c - s + \sum_{p_j(x) \in S} e_j \cdot \log_{\alpha(x)} p_j(x).$$

   Die diskreten Logarithmen $\log_{\alpha(x)} p_j(x)$ für $p_j(x) \in S$ und $\log_{\alpha(x)} c$ für $c \in \mathbb{F}_p$ wurden im Schritt 1.(a) bestimmt.

Die Schranke b sollte so gewählt werden, daß für die Menge S

$$\#S \approx \exp\left(d \cdot \sqrt{\log p^n \log\log p^n}\right)$$

gilt, wobei $d > 0$ eine geeignete Konstante ist.

Die erwartete Laufzeit des Algorithmus ist $L\left[p^n; \frac{1}{2}; c\right]$ für ein $c > 0$, wobei

$$L[q, \alpha, c] := \mathrm{O}\left(\exp\left((c + o(1))(\log q)^{\alpha}(\log\log q)^{1-\alpha}\right)\right),$$

mit Konstanten $0 \leq \alpha \leq 1$ und c die für die Laufzeit solcher Algorithmen üblicherweise verwendete Darstellung ist. (Wenn $\alpha = 0$, dann ist $L[q, \alpha, c]$ polynomial in $\log q$. Bei $\alpha = 1$ ist $L[q, \alpha, c]$ dagegen exponentiell in $\log q$. Wenn $0 < \alpha < 1$, dann ist $L[q, \alpha, c]$ *subexponentiell* in $\log q$.)

5.4.3 Index-Calculus für $\mathbb{F}_p$

Wir stellen die Elemente von $\mathbb{F}_p^*$ als die Menge der Zahlen $\{1, 2, \ldots, p-1\}$ dar. Die Multiplikation wird modulo p ausgeführt. Der Erzeuger von $\mathbb{F}_p^*$ sei α. Es sei m eine Zahl, die eine Funktion von p ist. Die Faktorbasis S ist die Menge aller Primzahlen kleiner m.

Der erste und zweite Schritt werden wie im generischen Fall ausgeführt, wobei die Faktorisierung in $\mathbb{Z}$ benutzt wird.

Wenn wir die Wahrscheinlichkeit, daß eine Zufallszahl kleiner p in Primfaktoren kleiner m zerfällt, betrachten, können wir m optimal wählen, um die Vorberechnungsphase (erster Schritt) und die Berechnung eines beliebigen Logarithmus (zweiter Schritt) zu beschleunigen. Das führt zu einer unter heuristischen Annahmen subexponentiellen Version des Index Calculus Algorithmus mit einer erwarteten Laufzeit von $L[p, 1/2, c]$.

Kapitel 6

Digitale Signatur

6.1 Einführung

Eine digitale Signatur soll ähnlich wie eine handschriftliche Unterschrift folgende Eigenschaften erfüllen:

- Authentizität, d. h. einer Person eindeutig (und normalerweise unter deren Mitwirkung) zugeordnet;
- Integrität, d. h. Teile des Dokuments oder das Dokument sind nach der Unterschrift nicht änderbar, die Signatur ist nicht „kopierbar“;
- Unabstreitbarkeit (non repudiation);
- kurz im Vergleich zum Dokument.

Es ist schwierig, elektronisch alle Kriterien gleichzeitig zu realisieren.

6.1.1 Signatur mit Hilfe eines Notars und eines symmetrischen Kryptosystems

Übung: Geben Sie unter Verwendung eines symmetrischen Kryptosystems und eines Notars ein Signaturverfahren an. Wie werden die folgenden Aufgaben gelöst:

- Wie übermittelt Alice an Bob eine signierte Nachricht?
- Wie kann Bob Celia nachweisen, daß das Dokument von Alice signiert wurde?

Welche Kriterien werden von Ihrem Verfahren erfüllt?

6.1.2 Signatur mit Public-Key-Verfahren

Es gibt auch Public-Key-Verfahren für Signaturen und Authentifizierung, die Idee dazu ist einfach:

1. Alice wendet den Signaturalgorithmus mit ihrem geheimen Schlüssel auf die Nachricht an und erhält eine Signatur.
2. Alice schickt ihre Nachricht mit ihrer Unterschrift an Bob.
3. Bob wendet den Verifikationsalgorithmus mit Alices öffentlichem Schlüssel auf die Nachricht und die Signatur an und kann dadurch verifizieren ob Alice die Nachricht signiert hat.

Es ist wichtig, daß ein Angreifer mit Hilfe des öffentlichen Schlüssels nicht aus einer „zufälligen“ Signatur eine dazugehörige „zufällige“ Nachricht erzeugen kann und damit ein als korrekt akzeptiertes Paar kennt.

6.1.3 Signatur mit Hashfunktionen

Public-Key-Verfahren sind oft ineffizient und man möchte vermeiden, die ganze Nachricht m mit dem Public-Key-Verfahren zu signieren. Dieses Problem läßt sich mit Hilfe einer kryptographischen Hashfunktion beheben, indem nur der Hashwert der Nachricht signiert wird. Durch die Einweg-Eigenschaft der Hashfunktion wird auch das oben angesprochene Problem bei zufälligen Nachrichten gelöst.

1. Alice berechnet den Hashwert $h = H(m)$ der Nachricht m.
2. Alice signiert den Hashwert h und damit indirekt die Nachricht m.
3. Alice schickt ihre Nachricht m mit ihrer Unterschrift an Bob.
4. Bob berechnet den Hashwert der empfangenen Nachricht und verifiziert die Signatur für den Hashwert.

Vorteil: Deutlich geringerer Rechenaufwand und Trennung von Nachricht und kurzer Unterschrift.
Es lassen sich auch Zufallszahlen signieren.
Nachteil: Die Sicherheit hängt auch von der Wirksamkeit der Angriffe gegen Hashfunktionen ab.

Übung: Eine kryptographische Hashfunktion ist einweg und, stärker, kollisionsresistent. Was bedeutet es für die Sicherheit der Signatur, wenn die (a) nur Einweg-Eigenschaft und (b) auch die Kollisionsresistenz gebrochen wird?

6.2 Signieren mit RSA

Die erste Idee für eine Signatur mit dem RSA-Algorithmus ist die folgende: Eine Signatur einer Nachricht m berechnet man durch $(\sigma := m^d \bmod n)$, und überprüft sie durch $(\sigma^e \bmod n \stackrel{?}{=} m)$.

Das so gewonnene Signaturverfahren hat jedoch einige Nachteile:

- Es können offenbar nur Nachrichten $m \in \{0, \ldots, n-1\}$ signiert werden.
- Jedes Element $\sigma \in \{0, \ldots, n-1\}$ ist eine gültige Signatur zur „Nachricht“ $m := \sigma^e \bmod n$. Es lassen sich also auch ohne Signaturschlüssel gültige Signaturen erzeugen. (Die dazugehörigen Nachrichten sind aber im allgemeinen sinnlos.)
- Angenommen, es sind Signaturen $p_i^d \bmod n$ zu kleinen Primzahlen p_i bekannt. Damit kann aufgrund der Homomorphieeigenschaft der Operation „Exponentiation mit e modulo n“ auch eine gültige Signatur $m^d \bmod n$ zu jeder Nachricht $m \in \{0, \ldots, n-1\}$ hergestellt werden, welche sich als Produkt der p_i darstellen läßt:

$$m = \prod_{i=1}^{t} p_i^{\lambda_i} \implies m^d \equiv \prod_{i=1}^{t} (p_i^d)^{\lambda_i} \bmod n \quad (\lambda_i \in \mathbb{N}_0).$$

Diese Nachteile können behoben werden, wenn man eine Signatur zu einer Nachricht m durch $\sigma := H(m)^d \bmod n$ statt $\sigma := m^d \bmod n$ für eine kryptographische Hashfunktion $H : \{0,1\}^* \to \mathbb{Z}_n$ generiert. Hierbei ist zu beachten, daß die verwendete Hashfunktion H einen hinreichend großen Teil des Rings $\mathbb{Z}_n$ „abdecken“ sollte; es reicht z. B. nicht, eine Hashfunktion wie SHA-1 mit einer Ausgabelänge von 160 Bit zu benutzen, wenn wir annehmen, daß 160-Bit-Zahlen effizient faktorisiert werden können. In diesem Fall kann bei einigen gegebenen Nachricht/Signatur-Paaren $(m_i, H(m_i)^d \bmod n)$ durch Faktorisieren der Hashwerte $H(m_i)$ in ähnlicher Weise wie oben eine Signatur zu einer gegebenen neuen Nachricht m' erzeugt werden. (Wie?)

Eine besondere Rolle spielt auch die Kollisionsresistenz der verwendeten Hashfunktion. Sie garantiert insbesondere eine Integrität der Signatur. (Sonst wäre nicht mehr gesichert, daß eine Signatur wirklich zu der behaupteten Nachricht gehört – die Nachricht könnte gegen eine mit demselben Hashwert ausgetauscht worden sein!)

6.2.1 Blinde Signaturen

Angenommen, es wird zur RSA-Signatur und -Verschlüsselung (jeweils in der Textbook-Variante) dasselbe Schlüsselpaar verwendet. Eve belauscht die von Bob an Alice geschickte Nachricht und kennt dadurch das Chiffrat $c \equiv m^e \pmod n$, wobei e der öffentliche und d der geheime Schlüssel von Alice sind. Eve wählt sich ein r zufällig mit $\mathrm{ggT}(r, n) = 1$ und berechnet $x \equiv r^e \pmod n$ und $y \equiv xc \pmod n$. Eve schafft es weiterhin irgendwie, daß

Alice die Nachricht y signiert (beispielsweise als Zeitstempelsignatur). Damit kennt Eve y^d. Sie berechnet

$$r^{-1}y^d \equiv r^{-1}x^d c^d \equiv r^{-1}(r^e)^d c^d \equiv r^{-1}rm \equiv m \pmod{n}$$

und erhält damit die Nachricht m. Das Problem ist: Alice führt eine blinde Signatur aus.

6.3 Public-Key-Signatursysteme

6.3.1 Unterschiedliche Sicherheitsstufen

Auch bei Signaturen können wir uns verschieden starke Angriffe vorstellen, deren Auswirkungen eines entsprechenden realen Angriffes sich unterscheiden. Die Unterscheidung kann dabei z. B. nach der Schwere des Bruchs bzw. dem Angriffsziel oder nach den Möglichkeiten, die man einem Angreifer zur Verfügung stellt, erfolgen.

Mögliche Ziele eines Angriffs sind dabei u. a.:

- totaler Bruch (Ermittlung des geheimen Signaturschlüssels)
- universelle Fälschbarkeit (effizienter Algorithmus zum Fälschen von Signaturen)
- selektive Fälschbarkeit (Signaturen zu a priori bestimmten Nachrichten sind fälschbar)
- existenzielle Fälschbarkeit (Signatur zu einer vom Angreifer gewählten Nachricht ist fälschbar)

Als Arten von Angriffen, die sich in der Mächtigkeit des Angreifers äußern, kennzeichnet man z. B.:

- Key-Only-Angriff (Angreifer erhält nur den Verifikationsschlüssel)
- Known-Message-Angriff (Angreifer erhält einige Nachricht-Signatur-Paare)
- Adaptive-Chosen-Message-Angriff (Angreifer erhält Zugriff auf ein Signaturorakel (z. B. Chipkarte))

Um eine größtmögliche Sicherheit zu gewährleisten, konzentrieren wir uns also auf das schwächste Angriffsziel mit dem stärksten Angreifer, so daß wir Signaturen haben wollen, die sicher sind gegen existenzielle Fälschbarkeit unter einem Adaptive-Chosen-Message-Angriff.

6.3.2 Definition

Auch für Signatursysteme kann ein Sicherheitsbegriff sauber gefaßt werden. Hierzu muß zunächst fixiert werden, was wir mit einem Public-Key-Signatursystem meinen:

Definition 6.1. Ein *Public-Key-Signatursystem* S besteht aus drei PPT-Algorithmen KeyGen, Sign, Verify. Für beliebige $k \in \mathbb{N}$ und $vk, sk, m, s \in \{0,1\}^*$ werden folgende *syntaktische* Bedingungen gestellt:

- $\mathsf{KeyGen}(1^k) \in \{0,1\}^* \times \{0,1\}^*$,
- $\mathsf{Sign}(1^k, sk, m) \in \{0,1\}^*$,
- $\mathsf{Verify}(1^k, vk, m, s) \in \{\texttt{valid}, \texttt{invalid}\}$.

Es wird *Korrektheit* in dem Sinne gefordert, daß die Wahrscheinlichkeit als Funktion in k vernachlässigbar sein muß, ein Schlüsselpaar $(vk, sk) \leftarrow \mathsf{KeyGen}(1^k)$ zu generieren, so daß eine Nachricht $m \in \{0,1\}^*$ existiert, für die $\texttt{invalid} \leftarrow \mathsf{Verify}(1^k, vk, m, \mathsf{Sign}(1^k, sk, m))$ *möglich* ist. Weiter wird *Eindeutigkeit* der Verifikation gefordert, in dem Sinne, daß für beliebige, aber feste $k \in \mathbb{N}$ und $vk, m, s \in \{0,1\}^*$ die Ausgabe von $\mathsf{Verify}(1^k, vk, m, s)$ konstant ist.[1]

In dieser Definition entspricht sk dem *Signaturschlüssel*, mit welchem Nachrichten signiert werden können; vk bezeichnet den *Verifikationsschlüssel*, mit dessen Hilfe signierte Nachrichten auf Authentizität überprüft werden können. Man beachte aber, daß ähnlich wie für Public-Key-Kryptosysteme die gegebene Definition nur die *Korrektheit* eines Signatursystems einfängt. Damit genügt etwa ein triviales System, welches alle Signaturen als gültig akzeptiert (d. h. $\mathsf{Verify}(\cdot,\cdot,\cdot) = \texttt{valid}$) schon Definition 6.1, obwohl hier natürlich Signaturen trivial fälschbar sind, und damit keine *Nicht-Abstreitbarkeit* von Signaturen gegeben ist.

Die Frage der *Sicherheit* eines Signatursystems soll sogleich behandelt werden, zuvor jedoch noch zwei technische Bemerkungen: Zunächst sind analog zu Definition 2.9 seltene „schlechte“ Schlüsselpaare (vk, sk) erlaubt. Dies geschieht in erster Linie, um tatsächlich eingesetzte Signatursysteme zuzulassen. Man denke hier beispielsweise an RSA- oder ElGamal-basierte Systeme, welche aus Effizienzgründen häufig probabilistische und prinzipiell fehleranfällige Primzahltests verwenden.

Des weiteren wird vom Verifikationsalgorithmus deterministisches Verhalten gefordert. Zwar sind die Verifikationsalgorithmen aller in diesem Kapitel erwähnten Signaturverfahren tatsächlich deterministisch, jedoch herrscht in der Literatur keine einhellige Auffassung darüber, ob ein „echt probabilistischer“ Verifikationsalgorithmus zuzulassen ist.

Man beachte dazu, daß es bei einem probabilistischen Verifikationsalgorithmus prinzipiell möglich ist, aus einem legitim erzeugten Nachricht-Signatur-Paar (welches also wegen der

[1] Effektiv wird hier also gefordert, daß der PPT-Algorithmus Verify durch einen deterministischen, polynomial laufzeitbeschränkten Algorithmus ersetzt werden kann.

Korrektheit des Signaturverfahrens immer als gültig verifiziert wird) eine neue Signatur für dieselbe Nachricht generiert werden kann, welche etwa mit Wahrscheinlichkeit 1/2 als gültig erkannt wird. Ein solches Verfahren kann sogar leicht aus einem in einem noch zu definierenden Sinne „sicheren“ Public-Key-Signatursystem konstruiert werden, so daß das entstandene Verfahren immer noch „sicher“ ist.

Die Existenz solcher „halbgültiger“ Signaturen scheint jedoch für den Einsatz des Signaturverfahrens in einem größeren Protokoll alles andere als wünschenswert: Man denke hier etwa an signierte Verträge, die an mehrere Parteien gleichzeitig geschickt werden. Insofern scheint die Modellierung eines notwendig deterministischen Verifikationsalgorithmus berechtigt.[2]

Doch nun zur Sicherheit eines Signatursystems. Intuitiv soll natürlich verhindert werden, daß ein nicht durch Kenntnis eines Signaturschlüssels befugter Angreifer eine Signatur fälscht. Hierbei kann unterschieden werden, ob ein solcher Angriff nur als erfolgreich gilt, wenn eine Signatur zu einer *vorgegebenen* Nachricht gefälscht wurde, oder ob es reicht, eine Signatur zu *irgendeiner* Nachricht zu fälschen. Im letzteren Fall ist sofort einsichtig, daß die Grundform der oben beschriebenen RSA- und ElGamal-Signaturverfahren anfällig für Angriffe ist.

Unabhängig davon können auch – ähnlich wie im Fall von Public-Key-Kryptosystemen – die Möglichkeiten des Angreifers bei einem Angriff variiert werden. Etwa kann dem Angreifer ein *Signaturorakel* zugestanden werden, das Klartexte nach Wahl des Angreifers signiert. Es ist klar, daß durch das Signaturorakel erhaltene Signaturen nicht als erfolgreich gefälschte Signaturen gewertet werden dürfen.

Ein sehr starkes Sicherheitskriterium („EUF-CMA“ = „existential unforgeability with respect to adaptive chosen-message attacks“) erhält man also dadurch, wenn man fordert, daß noch nicht einmal *irgendwelche* gültigen Signaturen gefälscht werden können, selbst wenn ein Signaturorakel zur Verfügung steht. (Die durch das Signaturorakel generierten Signaturen sind dabei natürlich nicht als „gefälscht“ zu zählen.)

Definition 6.2. Sei $S = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ ein Public-Key-Signatursystem. Für einen Angreifer (d. h. PPT-Algorithmus) A sei folgendes Experiment definiert:

$$\begin{array}{l}\text{Experiment } \mathbf{Exp}^{\text{ef-cma}}_{S,A}(k)\text{:}\\ \quad (vk, sk) \leftarrow \mathsf{KeyGen}(1^k)\\ \quad (m, s) \leftarrow A^{\mathsf{Sign}(1^k, sk, \cdot)}(1^k, vk)\\ \quad \textbf{Return } \mathsf{Verify}(1^k, vk, m, s)\end{array}$$

Hierbei wird verlangt, daß A nie Nachrichten m zurückgibt, für die A beim $\mathsf{Sign}(1^k, sk, \cdot)$-Orakel Signaturen erfragt hat. Dann heißt S EUF-CMA-sicher, wenn für alle in vorstehendem Sinne erlaubten Angreifer A die Funktion

$$\mathsf{Pr}\left[\mathbf{Exp}^{\text{ef-cma}}_{S,A}(k) \to \texttt{valid}\right]$$

[2] Alternativ ist natürlich auch eine Abschwächung dahingehend denkbar, daß der Verifikationsalgorithmus nur mit überwältigender Wahrscheinlichkeit ein gewisses Ergebnis liefern muß.

vernachlässigbar in k ist.

Für das vorgestellte ElGamal-Verfahren in der „gehashten“ Form ist derzeit nicht bekannt, ob es EUF-CMA-sicher ist. Allerdings gibt es gepaddete RSA-Varianten, welche EUF-CMA-sicher sind. Dies gilt beispielsweise für das RSA-PSS-Verfahren, welches im Public Key Cryptography Standard der RSA Laboratories zu finden ist.

6.4 ElGamal-Signaturverfahren

Das Signaturverfahren läßt sich nur über einem Primkörper realisieren. Es gilt hier also $g \in \mathbb{F}_p^*$ ist primitives Element. Es soll wieder g^x als öffentlicher Schlüssel verwendet werden.

Bob will eine Nachricht m signieren. Die Idee bei der Signatur ist es, die Nachricht m als

$$m \equiv ax \pmod{p-1}$$

auszudrücken. Die Signatur a kann dann durch

$$g^m \equiv g^{ax} \equiv y^a \pmod{p}$$

überprüft werden. Der geheime Schlüssel x kann aber aus der Signatur a

$$x \equiv ma^{-1} \pmod{p-1}$$

berechnet werden. Deswegen wird eine lineare Verschiebung benutzt. Dazu wählt Bob k zufällig mit $\mathrm{ggT}(k, p-1) = 1$ und berechnet $a \equiv g^k \pmod{p}$. Bob berechnet außerdem noch ein b, so daß

$$m \equiv xa + kb \pmod{(p-1)}$$

gilt. Die Signatur ist dann das Tupel (a, b). Andere Teilnehmer können Bobs Signatur überprüfen, indem sie

$$g^m \equiv g^{xa+kb} \equiv g^{xa} g^{kb} \equiv y^a a^b \pmod{p}$$

nachrechnen. Die Signatur funktioniert nur bei Primkörpern, weil sonst Probleme mit dem Datentyp auftreten würden. Man müßte mit einem Gruppenelement potenzieren. Bei Primkörpern werden die Gruppenelemente als natürliche Zahlen aufgefaßt. In den anderen Fällen ist dies nicht möglich. Denkbar wäre allerdings der Einsatz einer Funktion, welche ein Gruppenelement auf einen Exponenten abbildet; eine solche Variante des ElGamal-Systems auf elliptischen Kurven wird später beschrieben.

Es ist wichtig, daß jedesmal ein anderes k gewählt wird. Falls $a = g^k \pmod{p}$ zweimal verwendet wird, kann das lineare Gleichungssystem

$$\begin{aligned} m_1 &= xa + kb_1 \\ m_2 &= xa + kb_2 \end{aligned}$$

aufgestellt und der geheime Schlüssel x bestimmt werden.

Auch können bei diesem Verfahren – ähnlich wie bei dem oben vorgestellten RSA-basierten Signatursystem – gültige Signaturen zu (im allgemeinen unsinnigen) Nachrichten erzeugt werden:

1. Wähle $c, u \in \{0, \ldots, p-2\}$ teilerfremd zu $p-1$.
2. Setze $a := g^c y^u \bmod p$, $b := -a/u \bmod p-1$, $m := c \cdot b \bmod p-1$.
3. Dann ist (a, b) eine gültige Signatur zur Nachricht m.

Dieses Problem – wie auch das Problem, daß nur Nachrichten $m \in \{0, \ldots, p-2\}$ signiert werden können – kann wie im Fall des RSA-Systems durch ein Hashen der zu signierenden Nachricht m behoben werden. Hier ist es sicher genug, beispielsweise den Hash-Algorithmus SHA-1 (es gibt aber Bestrebungen, zu neuen Hashfunktionen wie SHA-256 und SHA-512 zu wechseln) zu verwenden, wie beim im Folgenden beschriebenen Digital Signature Algorithm (DSA).

Auch beim DSA ist ein Primkörper $\mathbb{F}_p^*$ gegeben, sowie ein Element $g \in \mathbb{F}_p^*$ von Primordnung q. Der geheime Schlüssel ist nun ein $x \in \mathbb{Z}_q$ und als öffentlicher Schlüssel dient auch hier $y := g^x$.

Eine Nachricht m wird nun folgendermaßen signiert:

1. Wähle $k \in \{0, \ldots, q-1\}$ zufällig.
2. Berechne $r := (g^k \bmod p) \bmod q$.
3. Berechne $s := (k^{-1}(\text{SHA-1}(m) + x * r)) \bmod q$.
4. Dann ist (r, s) eine gültige Signatur zur Nachricht m.

Die Verifikation akzeptiert genau dann, wenn $r \equiv (g^{s^{-1}\,\text{SHA-1}(m)} y^{s^{-1}r} \bmod p) \bmod q$.

6.5 Signieren mit elliptischen Kurven

Eine Variante des DSA gibt es auch für elliptische Kurven. Kryptographie auf elliptischen Kurven wird zunehmend populärer. Der Hauptvorteile von Verfahren, die auf elliptischen Kurven basieren, sind kürzere Schlüssellängen und insbesondere bei Anwendungen, die eine langfristige Sicherheit erfordern, daraus resultierend auch eine bessere Effizienz, wie man in der Tabelle 6.1 über nötige Schlüssellängen ablesen kann. Das liegt daran, daß bisher kein subexponentieller Algorithmus wie Index Calculus zur DLOG-Berechnung in elliptische Kurven bekannt ist.

symmetrische Verschlüsselung	elliptische Kurven p, q in EC-DSA	RSA-Modulus oder p in DSA
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

Tabelle 6.1: Schlüssellängen bei vergleichbarer Sicherheit

Für den EC-DSA wird zunächst als öffentlichen Parameter ein endlicher Primkörper $\mathbb{F}_p$ und eine elliptische Kurve $E(\mathbb{F}_p)$ über $\mathbb{F}_p$ benötigt. Die elliptische Kurve muß eine Untergruppe von Primzahlgröße q enthalten, ähnlich wie bei DSA. Zusätzlich ist ein öffentlicher Punkt $G \in E(\mathbb{F}_p)$ gegeben, der die Untergruppe der Ordnung q erzeugt.

Der geheime Signierschlüssel ist wieder ein $x \in \{1, \ldots, q-1\}$, der öffentliche Verifikationsschlüssel der Punkt $P := x \cdot G$. Wie schon erwähnt wird eine Abbildung benötigt, die einen Punkt auf der elliptischen Kurve auf eine Zahl abbildet, die im Exponenten verwendet werden kann. Dazu dient die Abbildung $\pi(\cdot)$, die einen Punkt auf die Zahldarstellung seiner x-Koordinate abbildet. Letztendlich wird auch hier eine kollisionsresistente Hashfunktion $H(\cdot)$ verwendet. Eine Signatur einer Nachricht m wird nun folgendermaßen berechnet:

1. Wähle zufällig $k \in_R \{1, \ldots, q-1\}$.
2. Berechne $r = \pi(k \cdot G) \bmod q$.
3. Berechne $s = k^{-1}(xr + H(m)) \bmod q$.
4. Die Signatur ist nun das Paar (r, s).

Die Signatur (r, s) für $m \in \{0,1\}^*$ wird verifiziert und nur akzeptiert falls

$$r = \pi((s^{-1}H(m) \bmod q) \cdot G + (s^{-1}r \bmod q) \cdot P) \bmod q.$$

6.6 Ist EUF-CMA sicher genug?

Wenn Signaturen in einem Protokoll verwendet werden, muß genau darauf geachtet werden, ob die Garantien des Sicherheitskriteriums EUF-CMA ausreichen. Falls ein Protokoll implizit zusätzliche Anforderungen an die Signatur stellt, ist nicht jede Signatur geeignet, um eingesetzt zu werden. Im schlimmsten Fall kann ein Protokoll neue Angriffsmöglichkeiten bieten, die ohne Signatur nicht vorhanden wären. Dazu werden im folgenden zwei Angriffe vorgestellt, die von der Definition von EUF-CMA nicht berücksichtigt werden.

6.6.1 Key-Substitution-Angriffe

Die Definition von EUF-CMA konzentriert sich auf eine einzelne signierende Partei. In einem Mehrbenutzerszenario können jedoch neue Probleme auftreten, die so noch nicht abgedeckt sind. Eine Aufgabe von Signaturen ist es, Nachrichten zu authentifizieren, wie in den gewünschten Eigenschaften aufgelistet wurde. Dazu muß eine Bindung zwischen öffentlichem Schlüssel und geheimen Schlüssel hergestellt werden. Die Aufgabe, öffentliche Schlüssel Benutzern zuzuordnen, fällt üblicherweise einer Zertifizierungsstelle zu. Die einfachste denkbare Zertifizierungsstelle nimmt einen öffentlichen Schlüssel von einem Benutzer authentifiziert entgegen und veröffentlicht (Benutzername, öffentlicher Schlüssel) in einer Liste. In einem Mehrbenutzerszenario könnten nun jedoch zwei Parteien denselben öffentlichen Schlüssel registriert haben und damit eine eindeutige Zuordnung einer Signatur zu einem Signierer erschweren oder verhindern.

Aber auch wenn die Zertifizierungsstelle prüft, daß keine öffentlichen Schlüssel doppelt vorkommen, oder sogar einen Beweis verlangt, daß dem Benutzer auch der geheime Schlüssel bekannt ist, ist EUF-CMA noch nicht genug. Das verdeutlichen die Key-Substitution-Angriffe:

Definition 6.3 (Key-Substitution-Angriff). Ein *(starker) Key-Substitution-Angriff* auf ein Signaturverfahren $S = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ ist ein PPT-Algorithmus KSA, der als Eingabe einen öffentlichen Schlüssel y und Zugriff auf ein Signaturorakel bekommt. Der Algorithmus berechnet einen neuen öffentlichen Schlüssel $\overline{y}$, so daß für ein Nachrichten-Signatur-Paar (m, s), das KSA vom Orakel bekommen hat, neben $\mathsf{Verify}(m, s, y) = \texttt{valid}$ auch $\mathsf{Verify}(m, s, \overline{y}) = \texttt{valid}$.

Der Schlüsselersetzungsangriff heißt *schwach*, falls KSA zusätzlich auch einen geheimen Schlüssel $\overline{x}$ zu $\overline{y}$ ausgeben muß.

Ein schwacher Key-Substitution-Angriff verhindert die Zuordnung einer Signatur zu einem Signierer also selbst dann, wenn zur Zertifizierung eines Schlüssels Kenntnis des geheimen Schlüssels erforderlich ist.

Eine Verallgemeinerung eines solchen Schlüsselersetzungsangriffs ist die Einbeziehung eines böswilligen Signierers, der mit dem Angreifer kooperiert. Der Signierer kann schon bei der Generierung seiner Schlüssel den Schlüsselersetzungsangriff planen. Bei einem solchen Angriff benötigt der Angreifer nur den Sicherheitsparameter als Eingabe und muß $(y, \overline{y}, m, s)$ mit $\mathsf{Verify}(m, s, y) = \texttt{valid}$ und $\mathsf{Verify}(m, s, \overline{y}) = \texttt{valid}$ ausgeben. Wieder kann man eine schwache Variante definieren, bei denen der Angreifer auch zugehörige geheime Schlüssel x und $\overline{x}$ berechnen muß.

Ein Key-Substitution-Angriff auf RSA

Der folgende Algorithmus führt einen schwachen Key-Substitution-Angriff auf das RSA-Verfahren durch. Der Angriff erfordert, daß der öffentliche Exponent e frei gewählt werden darf.

Gegeben ist ein öffentlicher Schlüssel (n, e) sowie eine Signatur σ für eine Nachricht m. Der folgende Algorithmus führt einen Key-Substitution-Angriff durch.

1. Wähle Primzahl $\overline{p}$, so daß $\overline{p} - 1$ in kleine Faktoren zerfällt und σ und $H(m)$ beide $\mathbb{Z}_{\overline{p}}^*$ generieren.
2. Wähle Primzahl $\overline{q}$, so daß $\overline{n} = \overline{pq} > n$, $\overline{q} - 1$ in kleine Faktoren zerfällt, ggT$(\overline{p} - 1, \overline{q} - 1) = 2$ und σ und $H(m)$ beide $\mathbb{Z}_{\overline{q}}^*$ generieren.
3. Da $\overline{p} - 1$ und $\overline{q} - 1$ in kleine Faktoren zerfallen, ist das DLOG-Problem lösbar (Pohlig-Hellman-Algorithmus). Berechne x_1 und x_2, so daß $\sigma^{x_1} \equiv H(m) \pmod{\overline{p}}$ und $\sigma^{x_2} \equiv H(m) \pmod{\overline{q}}$.
4. Finde das eindeutige $\overline{e}$, $1 < \overline{e} < \varphi(\overline{n})/2$, so daß $\overline{e} \equiv x_1 \pmod{(\overline{p} - 1)}$ und $\overline{e} \equiv x_2 \pmod{(\overline{q} - 1)}$.
5. Gib den Schlüssel $(\overline{n}, \overline{e})$ aus.
6. Es gilt nun $\sigma^{\overline{e}} \equiv H(m) \pmod{\overline{n}}$, da offensichtlich $\sigma^{\overline{e}} \equiv H(m) \pmod{\overline{p}}$ und $\sigma^{\overline{e}} \equiv H(m) \pmod{\overline{q}}$.

Ein geheimer Schlüssel $\overline{d}$ kann wie bei RSA berechnet werden, da die Faktorisierung von $\overline{n}$ bekannt ist.

Ein Key-Substitution-Angriff auf EC-DSA

Auch ein schwacher Key-Substitution-Angriff auf EC-DSA ist möglich – hierbei wird der Signierer als böswillig angenommen.

Gegeben sei ein öffentlicher Schlüssel P sowie eine Nachricht m mit Signatur (r, s). Die Verifikationsgleichung für die Signatur (r, s) ist

$$r = \pi((s^{-1}H(m) \bmod n) \cdot G + (s^{-1}r \bmod n) \cdot P) \bmod q,$$

wobei π einen Punkt auf dessen x-Koordinate projeziert. Es gilt

$$r = \pi(\pm(s^{-1}H(m) \bmod q) \cdot G + (s^{-1}r \bmod q) \cdot P) \bmod q,$$

da der inverse Punkt dieselbe x-Koordinate hat.

Ein neuer öffentlicher Schlüssel ist demnach der Punkt $\overline{P}$, mit

$$(s^{-1}H(m) \bmod q) \cdot G + (s^{-1}r \bmod q) \cdot \overline{P} = \\ -((s^{-1}H(m) \bmod q) \cdot G + (s^{-1}r \bmod q) \cdot P).$$

Der geheime Schlüssel $\overline{x}$ kann zunächst, falls der Signierer böswillig ist und x benutzt, durch $\overline{x} = -(2r^{-1}H(m) + x) \bmod q$ berechnet werden und danach der öffentliche Schlüssel

durch $\overline{P} = \overline{x} \cdot G$. (In dem sehr unwahrscheinlichen Fall $H(m) = -xr \bmod q$ und $\overline{x} \neq 0$ klappt das Verfahren nicht.)

Eine Variante von EC-DSA, bei der die Funktion π durch eine kryptographische Hashfunktion ersetzt wird, kann als sicher gegen (starke) Key-Substitution-Angriffe bewiesen werden.

DSA ist sicher gegen starke Key-Substitution-Angriffe

Ein Key-Substitution-Angriff auf DSA müßte eine Signatur (r, s) von m liefern, die für zwei öffentliche Schlüssel $y \neq \overline{y}$ gültig ist. Aus der Verifikationsgleichung folgt dann

$$y^{s^{-1}r} \bmod p \equiv \overline{y}^{s^{-1}r} \bmod p \pmod q.$$

Mit den üblichen Parametern, wobei p eine 1024-bit Primzahl, q eine 160-bit Primzahl ist und g eine Untergruppe von $\mathbb{Z}_p^*$ von Ordnung q erzeugt, ist $\bmod q$ jedoch eine kollisionsresistente Funktion auf $\langle g \rangle$.

6.6.2 Subliminale Kanäle

Ein subliminaler Kanal ist ein verdeckter Kanal in einem kryptographischen Verfahren wie beispielsweise in digitalen Signaturen. Der Kanal ermöglicht es, in einer Signatur eine subliminale Nachricht so einzubetten, daß sie nicht von Außenstehenden entdeckt werden kann.

Das Prinzip soll hier am Beispiel des Verfahrens RSA-PSS gezeigt werden. Wir nehmen an, daß der Signierer (Sender der subliminalen Nachricht) und Empfänger der subliminalen Nachricht einen gemeinsamen symmetrischen Schlüssel ssk miteinander vereinbart haben. Dann muß die subliminale Nachricht M so verschlüsselt werden, daß $r = E_{ssk}(M)$ wie ein Zufallsstring aussieht, der dann zum Berechnen des PSS-Encodings $EM(m)$ der zu signierenden Nachricht m verwendet wird. Zum Verifizieren der Signatur muß das PSS-Encoding überprüft werden, wodurch der Verifizierer den verwendeten Zufallsstring r, hier $E_{ssk}(M)$, lernt. Wer im Besitz von ssk und $EM(m)$ ist, kann also das Chiffrat entschlüsseln und lernt M.

Dadurch ist es möglich, daß ein manipuliertes Signierprogramm beispielsweise den geheimen Signier-Schlüssel sk über den subliminalen Kanal an einen bestimmten Empfänger (den Manipulator des Programms) überträgt. Der Besitzer der geheimen Schlüssels sk, der mit dem Programm Nachrichten signiert, merkt davon nichts: er kann nicht unterscheiden, ob der verwendete Zufall echt oder ein Chiffrat mit einem ihm unbekannten Schlüssel ssk ist.

Beim DSA ist es nicht so einfach, den zum Signieren verwendeten Zufallsstring zu extrahieren. Nur wenn neben einer DSA-Signatur (r, s) zu einer Nachricht m auch der geheime Schlüssel x gegeben ist, kann die verwendete Zufallszahl k rekonstruiert werden:

$$k = (\text{SHA-1}(m) + rx)s^{-1}.$$

Umgekehrt kann natürlich auch der geheime Schlüssel x ausgerechnet werden, falls der verwendete Zufall k bekannt ist. Der Empfänger der subliminalen Nachricht muß, falls DSA verwendet werden soll, also ebenfalls den geheimen Schlüssel x kennen, um die subliminale Nachricht empfangen zu können.

Dieser Nachteil besteht nicht bei einem sogenannten *Schmalband*-Kanal, der bei probabilistischen Signaturverfahren prinzipiell möglich ist. Dabei berechnet der Signierer eine Signatur σ für m so oft, bis die letzten l Bits von $H(\sigma)$ der gewünschten (verschlüsselten) subliminalen Nachricht entsprechen, wobei H eine beliebige zwischen Sender und Empfänger der subliminalen Nachricht vereinbarte Hashfunktion ist. Dazu werden jedoch bis zu 2^l Signieroperationen nötig sein, so daß sich das Verfahren nur für logarithmisch kurze subliminale Nachrichten M eignet.

Es gibt auch ein Signaturverfahren, das beweisbar keinen subliminalen Kanal hat. Das bereits bekannte RSA-PSS Signaturverfahren kann deterministisch verwendet werden. Wenn der Signierer zusätzlich beweist, daß sein Schlüssel dem Standard entspricht, (genauer, wenn e und $\varphi(n)$ teilerfremd sind), ist sichergestellt, daß es nur eine einzige gültige Signatur pro Nachricht gibt. Subliminale Kommunikation ist damit ausgeschlossen.

Kapitel 7

Schlüsselaustausch, Authentifikation und verwandte Aufgaben

Wenn sich Alice in einen Hostrechner einloggt, wie kann der Rechner wissen, daß es wirklich Alice ist? Wie kann er erkennen, daß Eve versucht, sich als Alice auszugeben? Üblicherweise wird dieses Problem mit Hilfe eines Paßworts gelöst. Alice gibt ihr Paßwort ein, und der Rechner überprüft, ob es korrekt ist. Alice und der Rechner haben ein gemeinsames Geheimnis, das er jedesmal von Alice verlangt, wenn sie sich einloggen will.

7.1 Authentifikation mit Einwegfunktionen

Roger Needham und Mike Guy bemerkten, daß der Host gar nicht die Paßwörter zu kennen braucht. Der Host muß nur die richtigen Paßwörter von den falschen unterscheiden können. Dies kann einfach mit Einwegfunktionen realisiert werden. Der Host speichert nur die Werte $f(\textit{Paßwort})$, wobei f eine Einwegfunktion ist.

1. Alice sendet ihr Paßwort dem Host.
2. Der Host wendet eine Einwegfunktion auf das Paßwort an.
3. Der Host vergleicht das Ergebnis mit dem zuvor gespeicherten Wert.

Da jetzt der Host nicht die Paßwörter speichert, sondern nur die Werte der mit einer Einwegfunktion verschlüsselten Paßwörter, ist diese Liste für einen Angreifer nicht direkt verwendbar, da er daraus die Paßwörter nicht mehr so leicht rekonstruieren kann.

Diese Authentifikation ist nicht sonderlich sicher, da ein *Wörterbuchangriff* unternommen werden kann. Eine Datei, die die mit einer Einwegfunktion verschlüsselten Paßwörter enthält, stellt immer noch einen Schwachpunkt dar. Mallory erstellt eine Liste der am häufigsten benutzten Paßwörter, verschlüsselt jedes Paßwort mit der Einwegfunktion und speichert die Ergebnisse. Mallory schafft es, die Datei mit den verschlüsselten Paßwörtern,

die sich auf dem Hostrechner befindet, zu stehlen und sucht nach Übereinstimmungen in den beiden Dateien. Dieser Angriff ist oft erstaunlich erfolgreich, was nicht an dem Verfahren mit der Einwegfunktion liegt, sondern den unachtsamen Benutzern anzulasten ist, da sie einfach zu erratende Paßwörter wählen.

7.2 Authentifikation mit Public-Key-Verfahren

Die Protokolle zur Authentifikation mit Hilfe von Paßwörtern haben eine weitere enorme Sicherheitslücke. Wenn Alice das Paßwort zu ihrem Host sendet, kann Eve ihr Paßwort lesen, wenn sie Zugang zu dem Kommunikationsweg oder dem Prozessorspeicher des Hosts hat.

Mit Hilfe der Public-Key-Kryptographie kann dieses Problem behoben werden. Der Host hat eine Datei mit den öffentlichen Schlüsseln aller Benutzer und alle Benutzer haben ihren geheimen Schlüssel. Ein einfaches Protokoll kann wie folgt aussehen:

1. Der Host sendet Alice eine Zufallszahl r.
2. Alice signiert r mit ihrem geheimen Signaturschlüssel und schickt das Chiffrat an den Host.
3. Der Host verifiziert die Signatur mit dem öffentlichen Schlüssel von Alice und überprüft r.

Ein Angreifer kann sich jetzt nicht dem Host gegenüber als Alice identifizieren, da er keinen Zugang zu Alices geheimen Schlüssel hat. Alice sendet niemals ihren geheimen Schlüssel über den Kommunikationsweg zu dem Host. Eve, die die Interaktion zwischen Alice und dem Host abhört, kann keine Information gewinnen, die es ihr ermöglichen würde, Alices geheimen Schlüssel zu berechnen oder sich als Alice auszugeben, ohne daß es der Host bemerkt. Weder der Host noch der Kommunikationsweg müssen sicher sein.

Es ist für Alice nicht geschickt, eine beliebige Zufallszahl r, die sie zugeschickt bekommt, zu signieren. Ein verbessertes Protokoll sieht wie folgt aus:

1. Alice berechnet etwas, wobei sie eine eigene Zufallszahl und ihren geheimen Schlüssel verwendet, und schickt das Ergebnis an den Host.
2. Der Host sendet Alice eine andere Zufallszahl.
3. Alice verwendet eine vorgegebene Funktion, um einen Funktionswert, der von den beiden Zufallszahlen und ihrem geheimen Schlüssel abhängt, zu berechnen und schickt das Ergebnis an den Host.
4. Der Host überprüft dies.

Falls Alice genauso wenig dem Host vertraut, wie der Host Alice traut, fordert sie ihn auf, sich ihr gegenüber zu authentifizieren.

7.3 Schlüsselaustausch

Diese Protokolle verbinden die Authentifikation mit einem Schlüsselaustausch, um ein generelles Problem zu lösen: Alice und Bob sind an zwei verschiedenen Enden eines Netzwerks und wollen sicher Daten übertragen. Wie können Alice und Bob einen geheimen Schlüssel austauschen und dabei sicher sein, daß keiner von ihnen in Wirklichkeit mit einem Angreifer Mallory spricht? Eine gute Einführung in Protokolle zum Schlüsselaustausch findet sich im ersten Kapitel von [2].

Es wird üblicherweise ein Angreifer angenommen, der

1. alle Nachrichten lesen kann,
2. beliebig Nachrichten unterdrücken oder einfügen kann,
3. ein legitimer Protokollteilnehmer (Insider), ein außenstehender Angreifer oder beides sein kann,
4. Schlüssel vergangener Protokolldurchläufe kennt. (Das modelliert eine beliebige Verwendbarkeit eines Sitzungsschlüssels.)

Das wichtigste Ziel, das Alice und Bob erreichen wollen, ist *implizite Schlüsselauthentifizierung*. Das bedeutet, beide wissen, daß niemand außer ihnen den Schlüssel kennt. Sie wissen jedoch nicht, ob der jeweilige Partner den Schlüssel berechnen konnte. Es gibt viele weitere Ziele eines Schlüsselaustauschs, jedoch muß implizite Schlüsselauthentifizierung immer erfüllt sein.

7.3.1 Wide-mouth frog

Das Wide-Mouth-Frog-Protokoll ist vielleicht das einfachste Protokoll, das ein symmetrisches Verfahren und eine vertrauenswürdige Schlüsselzentrale benutzt. Alice und Bob besitzen jeweils einen geheimen Schlüssel A bzw. B mit der Schlüsselzentrale. Diese Schlüssel werden nur für den Schlüsselaustausch und nicht zur Verschlüsselung von eigentlichen Nachrichten verwendet. Mit Hilfe von nur zwei Nachrichten übermittelt Alice an Bob einen geheimen Sitzungsschlüssel K. Mit T_A und T_B werden Zeitstempel bezeichnet. Die Verschlüsselungsfunktionen E_A und E_B sind mit den geheimen Schlüsseln parametrisiert, die Alice und Bob mit der Schlüsselzentrale teilen.

1. Alice erzeugt das Tupel $(Alice, E_A(T_A, Bob, K))$ und schickt es an die Schlüsselzentrale.
2. Die Schlüsselzentrale entschlüsselt die Nachricht von Alice. Sie erzeugt einen neuen Zeitstempel T_{SZ} und sendet das verschlüsselte Tupel $E_B(T_{SZ}, Alice, K)$ an Bob.

Die wichtigste Annahme bei diesem Protokoll ist, daß Alice kompetent ist, gute (sichere) Sitzungsschlüssel zu erzeugen. Man muß beachten, daß es schwierig ist, Zufallszahlen zu erzeugen, Alice ist möglicherweise nicht dazu in der Lage.

7.3.2 Needham-Schroeder

Das Protokoll von Needham und Schroeder benutzt ein symmetrisches Verfahren und eine Schlüsselzentrale. Es ist eines der ersten Verfahren, die in realen Systemen eingesetzt wurden.

1. Alice erzeugt eine Zufallszahl R_A und schickt das Tupel $(Alice, Bob, R_A)$ an die Schlüsselzentrale.
2. Die Schlüsselzentrale erzeugt einen zufälligen Sitzungsschlüssel K und schickt das Tupel $E_A(R_A, Bob, K, E_B(K, Alice))$ an Alice.
3. Alice entschlüsselt und erhält den Sitzungsschlüssel K. Sie überprüft die Zufallszahl R_A und schickt das Tupel $E_B(K, Alice)$ an Bob.
4. Bob entschlüsselt und erhält den Sitzungsschlüssel K. Er erzeugt eine Zufallszahl R_B und schickt das Tupel $E_K(R_B)$ an Alice.
5. Alice entschlüsselt und erhält die Zufallszahl R_B. Sie berechnet $R_B - 1$ und schickt das Tupel $E_K(R_B - 1)$ an Bob.
6. Bob prüft, ob $R_B - 1$ geschickt wurde.

Die Zufallszahlen R_A, R_B sollen „Replay-Attacken“ verhindern. In Schritt 2 ist Alice sicher, Kontakt mit der Schlüsselzentrale gehabt zu haben; Schritt 5 und 6 sichern Bob zu, daß Alice am Protokollablauf beteiligt war. Jedoch kann der Angreifer einen Replay-Angriff der Nachricht $E_B(K, Alice)$ an Bob erfolgreich durchführen, sobald er den alten Sitzungsschlüssel K gelernt hat.

7.3.3 Kerberos

Das Kerberos-Protokoll ist eine Variante des Needham-Schroeder-Protokolls. Die Zufallszahlen R_A, R_B werden durch Zeitstempel ersetzt. Durch die Zufallszahlen beim Protokoll von Needham-Schroeder sind alte Paare $(K, E_B(K, Alice))$ wertvoll; mit einem solchen Paar kann M sich als Alice ausgeben. Dies wird bei Kerberos durch Zeitstempel und Gültigkeitsdauern ausgeschlossen.

1. Alice sendet das Tupel $(Alice, Bob)$ an die Schlüsselzentrale.
2. Die Schlüsselzentrale schickt das Tupel $(E_A(T_{SZ}, L, K, Bob), E_B(T_{SZ}, L, K, Alice))$ an Alice. Die Nachricht wird mit einer Gültigkeitsdauer L und einem Zeitstempel T_{SZ} versehen.
3. Alice schickt das Tupel $(E_K(Alice, T_A), E_B(T_{SZ}, L, K, Alice))$ an Bob.
4. Bob sendet das Tupel $E_K(T_A + 1)$ an Alice.

Das Problem bei diesem Protokoll ist, daß die Uhren der verschiedenen Parteien synchron laufen müssen, was in der Realität nicht immer gegeben ist.

7.3.4 Neuman-Stubblebine

Dieses Protokoll soll das Problem von nicht synchronen Uhren von Kerberos beheben (und schafft dies auch). Von Schneier [9] wird es als ein exzellentes Protokoll bezeichnet, hat sich aber nicht durchgesetzt.

1. Alice erzeugt eine Zufallszahl R_A und sendet das Tupel $(Alice, R_A)$ an Bob.
2. Bob erzeugt eine Zufallszahl R_B und sendet das Tupel $(Bob, R_B, E_B(Alice, R_A, T_B))$ mit einem Zeitstempel T_B versehen an die Schlüsselzentrale.
3. Die Schlüsselzentrale erzeugt einen zufälligen Sitzungsschlüssel K und sendet das Tupel $E_A(Bob, R_A, K, T_B), E_B(Alice, K, T_B), R_B$ an Alice.
4. Alice entschlüsselt und überprüft, ob R_A mit dem Wert im 1. Schritt übereinstimmt. Alice sendet $E_B(Alice, K, T_B), E_K(R_B)$ an Bob.
5. Bob entschlüsselt und erhält den Sitzungsschlüssel K. Er überprüft, ob T_B und R_B mit den Werten im 2. Schritt übereinstimmen.

Eine kleine Protokollanalyse zeigt einige Stellen, an denen etwas verbessert werden kann:

- Sendet M die Anfrage $(Alice, R_A)$ zum Verbindungsaufbau an B, so wird B diese entsprechend an die Schlüsselzentrale weiterleiten. Damit kann ein Unbekannter (sehr schlecht zurückverfolgbar) die Schlüsselzentrale „beschäftigen“ und gleichzeitig in gewissem Sinne einen Chosen-Plaintext-Angriff auf den Schlüssel von B durchführen (Wahl der Zufallszahl!).
- B kann einen Known-Plaintext-Angriff auf den Schlüssel von A durchführen, sofern er die Antwort der Schlüsselzentrale abfangen kann.
- M kann einen Known-Plaintext-Angriff auf den Sitzungsschlüssel K durchführen, da R_B im Klartext und verschlüsselt übertragen wird.

Es ist nicht einfach nachzuvollziehen, weshalb B sowohl mit T_B, als auch mit R_B den korrekten Protokollablauf prüft. Außer B weiß niemand, wie genau die Uhr von B stimmt, diese Zeit hat daher auch nur die Funktionalität einer Zufallszahl.

7.4 Ein Modell für beweisbare Sicherheit

Auch für Schlüsselaustauschprotokolle hat es sich durchgesetzt, daß sie beweisbar sicher sein sollen. Es gibt mehrere Modelle, um Schlüsselaustausch zu formalisieren, hier wird ein grober Überblick über ein Modell gegeben, das auf Bellare und Rogaway zurückgeht und weit verbreitet ist. Das Prinzip ist ähnlich wie bei den Modellen zur Verschlüsselung

und zu Signaturen. Es wird auch hier ein Spiel definiert, das der Angreifer höchstens mit einer gewissen Wahrscheinlichkeit gewinnen können soll.

Gegeben ist eine Menge von Benutzern $\{U_1, \ldots, U_n\}$. Jeder Benutzer U_i wiederum hat mehrere Instanzen (Orakel) $\Pi_i^1, \Pi_i^2, \ldots$, die unabhängige Prozesse darstellen und das Schlüsselaustauschprotokoll ausführen. Jeder Benutzer U_i hat einen geheimen Schlüssel SK_i, der ihn identifiziert. Der zugehörige öffentliche Schlüssel PK_i ist allen Benutzern bekannt. Die Instanzen eines Benutzer haben alle Zugriff auf den geheimen Schlüssel des Benutzers.

Der Angreifer interagiert mit den Benutzern nun durch folgende Anfragen:

$\mathsf{Send}(U_i, s, M)$ Dieses Kommando versendet die Nachricht M an die Instanz Π_i^s. Die Instanz versucht, die Nachricht als Protokollnachricht zu interpretieren, und antwortet dem Angreifer entsprechend. Das modelliert ein unsicheres, asynchrones Netzwerk zwischen Benutzern. Die Instanzen bekommen die Nachrichten nur, wenn der Angreifer sie mittels dieses Kommandos sendet. Der Angreifer kann dabei natürlich auch die Nachrichten verändern oder eigene Nachrichten versenden.

$\mathsf{Reveal}(U_i, s)$ Dieses Kommando gibt dem Angreifer den Sitzungsschlüssel, den die Instanz Π_i^s bei einem früheren Schlüsselaustausch erhalten hat. Das modelliert die Verwendung des Sitzungsschlüssels durch die Parteien, was dazu führen kann, daß der Angreifer den Sitzungsschlüssel erfährt bzw. einfacher angreifen kann.

$\mathsf{Corrupt}(U_i)$ Dieses Kommando gibt dem Angreifer den geheimen Schlüssel SK_i von U_i. Das erlaubt dem Angreifer von dem Moment an als U_i an Protokollausführungen teilzunehmen.

$\mathsf{Test}(U_i, s)$ Dieses Kommando definiert nun das eigentliche Spiel. Der Angreifer darf das Kommando nur einmal benutzen, und die Instanz Π_i^s muß einen Schlüssel ausgetauscht haben. Dann wird ein zufälliges Bit b gezogen und falls $b = 1$ wird der Sitzungsschlüssel von Π_i^s zurückgegeben, falls $b = 0$ wird ein zufälliger Wert aus dem Sitzungsschlüsselraum zurückgegeben. Der Angreifer soll nun raten, ob $b = 1$ oder $b = 0$.

Der Angreifer darf $\mathsf{Test}(U_i, s)$ nur benutzen, falls Π_i^s an einem Schlüsselaustausch teilgenommen hat und weder korrumpiert (vom Angreifer gesteuert) war, noch durch Reveal nach dem Sitzungsschlüssel gefragt wurde. Dasselbe muß auch für die Partner von Π_i^s gelten, also die Instanzen, mit denen Π_i^s einen Schlüssel austauschen wollte. Wer ein Partner von Π_i^s ist, wird üblicherweise über einen Sitzungsidentifikator definiert. Wer den gleichen Identifikationswert hat und mit denselben Teilnehmern einen Schlüssel austauschen wollte, ist des anderen Partner. Falls das nicht erfüllt ist, ist es klar, daß der Angreifer den Schlüssel kennt.

Ein Protokoll ist nun sicher, falls die Wahrscheinlichkeit, daß ein Angreifer, der polynomiale Laufzeit hat, das Spiel gewinnt, nur vernachlässigbar besser als 50% ist. Eine 50%-Chance ergibt sich einfach daraus, das Bit b zufällig zu raten.

7.5 Public-Key-Verfahren zum Schlüsselaustausch

7.5.1 Diffie-Hellman-Verfahren

Das Verfahren von Diffie und Hellman ist ein Public-Key-Verfahren, das es Alice und Bob erlaubt, ein gemeinsames Geheimnis über eine unsichere Leitung auszutauschen. Das Protokoll kann mit einer beliebigen endlichen zyklischen Gruppe mit schwierigem Dlog-Problem realisiert werden. Die Gruppe G und ihr Erzeuger g sind öffentlich bekannt.

1. Alice wählt eine Zufallszahl a, berechnet g^a in G und sendet g^a an Bob.
2. Bob wählt eine Zufallszahl b, berechnet g^b in G und sendet g^b an Alice.
3. Alice empfängt g^b und berechnet $(g^b)^a$.
4. Bob empfängt g^a und berechnet $(g^b)^a$.

Alice und Bob haben jetzt ein gemeinsames Gruppenelement g^{ab}, das nur ihnen bekannt ist. Eve kennt durch Abhören nur die Gruppenelemente g^a und g^b. Es wird allgemein angenommen, daß die Gruppenelemente g^a und g^b nicht ausreichen, um g^{ab} auszurechnen, wenn es praktisch unmöglich ist, den diskreten Logarithmus von Gruppenelementen zu bestimmen.

Das *Diffie-Hellman-Problem* ist es, einen effizienten Algorithmus zu finden, der g^{ab} aus g^a und g^b berechnet. Es ist klar, daß die Lösung des diskreten Logarithmusproblems erlaubt, das Diffie-Hellman-Problem zu lösen. Die Umkehrung ist nicht bekannt.

Man beachte, daß bei dem Diffie-Hellman-Verfahren in der gerade beschriebenen Form ein *Man-in-the-middle-Angriff* (vgl. Abschnitt 1.6.2) möglich ist. Ein Angreifer kann sich – bei nicht-authentifizierten Kanälen – also zwischen Alice und Bob schalten und letztlich einen gemeinsamen Schlüssel κ_A mit Alice und einen gemeinsamen Schlüssel κ_B mit Bob erzeugen. Werden später diese Schlüssel von Alice und Bob zur sicheren Nachrichtenübertragung genutzt, so kann dieser Angreifer die so übertragenen Nachrichten abhören, ohne bemerkt zu werden. (Trotzdem gilt im allgemeinen $\kappa_A \neq \kappa_B$.)

7.5.2 Diffie-Hellman mit Signaturen

Um den Man-in-the-middle-Angriff zu verhindern, müssen sich Alice und Bob authentifizieren. Dazu können sie beispielsweise Signaturen benutzen. Beide haben jeweils einen geheimen Schlüssel SK_A bzw. SK_B und kennen die zugehörigen öffentlichen Schlüssel PK_A und PK_B zum Verifizieren der Signatur. Das folgende Protokoll vermeidet einen Man-in-the-middle-Angriff und erreicht implizite Schlüsselauthentifizierung:

1. Alice wählt eine Zufallszahl a, berechnet g^a in G und sendet die signierte Nachricht $(g^a, A, B)_{SIG_A}$ an Bob.

2. Bob wählt eine Zufallszahl b, berechnet g^b in G und sendet die signierte Nachricht $(g^b, A, B)_{SIG_B}$ an Alice.
3. Alice empfängt g^b und berechnet $K_{AB} = (g^b)^a$.
4. Bob berechnet $K_{AB} = (g^b)^a$.

Falls stattdessen nur $(g^a)_{SIG_A}$ bzw. $(g^b)_{SIG_B}$ gesendet wird, ist der folgende Angriff möglich:

Unknown-Key-Share-Angriff. Hier versucht der Angreifer C, Bob zu täuschen, wer sein Partner bei dem Schlüsselaustausch ist. Letztendlich gilt der Angriff jedoch Alice, die trotz eines erfolgreichen Schlüsselaustauschs mit Bob nicht wissen kann, ob Bob weiß, mit wem er spricht.

1. Alice wählt eine Zufallszahl a, berechnet g^a in G und sendet die signierte Nachricht $(g^a)_{SIG_A}$ an Bob.
2. Der Angreifer fängt die Nachricht ab und sendet stattdessen $(g^a)_{SIG_C}$ an Bob. Der Angreifer muß hierzu legitimer Teilnehmer sein und ein Schlüsselpaar besitzen, das haben wir dem Angreifer jedoch zugebilligt.
3. Bob wählt eine Zufallszahl b, berechnet g^b in G und sendet die signierte Nachricht $(g^b)_{SIG_B}$ an C. C leitet die Nachricht an Alice weiter.
4. Alice berechnet $K_{AB} = (g^b)^a$.
5. Bob berechnet $K_{CB} = (g^b)^a$.

Es gilt nun $K := K_{CB} = K_{AB}$. Falls Bob (Bank) nun eine mit K verschlüsselte Nachricht mit Geld von Alice bekommt, glaubt er, die Nachricht käme vom Angreifer C, und schreibt ihm den Wert gut.

7.5.3 MTI-Protokoll

Das MTI-Protokoll versucht, den Diffie-Hellman-Schlüsselaustausch mit der Authentifikation in einer Gleichung zu verbinden. Das Protokoll bildet die Grundlage für viele weitere Protokolle. Hier haben Alice und Bob die langfristigen geheimen Schlüssel x bzw. y, sowie zugehörige öffentliche Schlüssel g^x und g^y.

1. Alice wählt eine Zufallszahl a, berechnet g^a in G und sendet g^a an Bob.
2. Bob wählt eine Zufallszahl b, berechnet g^b in G und sendet g^b an Alice.
3. Alice empfängt g^b und berechnet $K_{AB} = (g^b)^x (g^y)^a = g^{ay+bx}$.

4. Bob empfängt g^a und berechnet $K_{AB} = (g^a)^y(g^x)^b = g^{ay+bx}$.

Das MTI-Protokoll hat jedoch auch Nachteile, beispielsweise wird die sogenannte *Forward-Security* nicht mehr erreicht. Wenn ein Angreifer es schafft, die langfristigen Schlüssel zu brechen, also x und y zu kennen, dann kann er den Sitzungsschlüssel K_{AB} eines alten Schlüsselaustauschs berechnen, falls er sich g^a und g^b gespeichert hat:

$$K_{AB} = (g^b)^x(g^a)^y.$$

Ein anderer interessanter Angriff auf das MTI-Protokoll ist ein Angriff durch Zertifikatersetzung. Das klappt, falls C ein Zertifikat für einen öffentlichen Schlüssel bekommt, dessen geheimen Schlüssel er nicht kennt. Wenn Alices Schlüssel $PK_A = g^x$ ist, wählt C nun $PK_C = g^{xz}$ für ein zufälliges z als seinen öffentlichen Schlüssel. Wenn Bob PK_C anerkennt, ist ein Unknown-Key-Share-Angriff in folgender Weise möglich:

1. Alice wählt eine Zufallszahl a, berechnet g^a in G und sendet g^a an Bob.
2. Der Angreifer C fängt die Nachricht ab, und sendet g^a in seinem Namen an Bob.
3. Bob wählt eine Zufallszahl b, berechnet g^b in G und sendet g^b an C.
4. C modifiziert die Nachricht zu g^{zb} und sendet sie an Alice.
5. Alice berechnet $K_{AB} = (g^{zb})^x(g^y)^a = g^{ay+bxz}$.
6. Bob berechnet $K_{CB} = (g^a)^y(g^{xz})^b = g^{ay+bxz}$.

7.5.4 Transport Layer Security (TLS)

Dies ist der Nachfolger von „Secure Socket Layer“ (SSL) und mittlerweile das Standardprotokoll für Authentifikation und Schlüsselaustausch im Internet. Es basiert auf Public-Key-Verfahren und setzt die Existenz einer Zertifizierungsstelle (ZS) voraus. Will A dieses Protokoll benutzen, so muß zuerst ZS auf einem sicheren Weg der öffentliche Schlüssel übermittelt werden. Nach Überprüfung der Identität von A signiert ZS das Paar (A, öffentlicher Schlüssel von A) und übergibt es zusammen mit dem öffentlichen Schlüssel von ZS an A. Dieser Datensatz wird als Zertifikat des Benutzers A bezeichnet.

Es können mehrere ZS existieren, die meistens hierarchisch organisiert sind und jeweils Zertifikate von den übergeordneten Stellen besitzen.

Will A nun eine sichere Verbindung zu B herstellen:

- A schickt zunächst ein „Hello“ an B.
- B antwortet mit seinem Zertifikat und einer Zufallszahl, die in die Schlüsselberechnung eingeht.

- A überprüft das Zertifikat, und antwortet mit seinem Zertifikat und einer eigenen Zufallszahl, die mit dem öffentlichen Schlüssel von B verschlüsselt wird. A berechnet den gemeinsamen Schlüssel aus den beiden Zufallszahlen.
- B überprüft das Zertifikat, entschlüsselt die Zufallszahl und berechnet aus den beiden Zufallszahlen den gemeinsamen Schlüssel.

Dieses Protokoll läßt viele verschiedene Grundalgorithmen (z. B. RSA und Diffie-Hellman) zu, über die sich die Partner in der detaillierten Ausführung des Protokolls einigen. Das Protokoll bzw. der sehr ähnliche Vorgänger SSL wird bei sicheren Verbindungen im Internet verwendet und ist in den gängigen WWW-Browsern implementiert. Die obige Darstellung ist dabei nur eine sehr vereinfachte Version, zudem existieren viele Varianten und optionale Möglichkeiten (z. B. Zertifikat nur bei einem der Benutzer).

7.5.5 Secure Shell (SSH)

Beim Einloggen auf einem Rechner über ein Netzwerk wird üblicherweise das Passwort des Benutzers im Klartext übertragen. Dieses große Sicherheitsloch wird durch das Benutzen einer „Secure Shell“ behoben. Die Grundidee hinter der ersten SSH-Version wird im folgenden dargestellt, für eine konkrete Umsetzung sind natürlich viele weitere Details zu beachten, außerdem gibt es in der Zwischenzeit bessere Nachfolgeversionen.

Protokoll-Initialisierung

- Dem Rechner H, auf dem sich der Benutzer A einloggen will, ist ein 1024 Bit RSA-Schlüsselpaar (Host Key) zugeordnet. Beim Starten des Rechners wird ein 768 Bit RSA-Schlüsselpaar (Server Key) erzeugt, das danach jede Stunde durch ein neues Paar ersetzt wird.
- Der Benutzer A kann sich ein 1024 Bit RSA-Schlüsselpaar (User Key) erzeugen und den öffentlichen Schlüssel davon auf H ablegen.

Protokoll-Ablauf

- A übermittelt den Wunsch zum Einloggen an H.
- H sendet die öffentlichen Schlüssel des Host Keys und des Server Keys an A.
- Der empfangene Host Key wird, falls er bereits in einer Datenbank des Benutzers A vorhanden ist, mit diesem verglichen.
  Falls er nicht vorhanden ist oder abweicht, erhält der Benutzer A eine Warnung und muß über das weitere Vorgehen entscheiden (neuen Schlüssel akzeptieren oder abbrechen).
  Dann wählt A einen Sitzungsschlüssel, verschüsselt ihn nacheinander mit dem Server Key und dem Host Key und schickt das Chiffrat an H.

- Ab jetzt wird verschlüsselt mit dem Sitzungsschlüssel kommuniziert.
  - 1. Variante: A authentifiziert sich, wie bei einem normalen Login üblich, mit seinem Passwort. Dies wird hier allerdings verschlüsselt übertragen.
  - 2. Variante: H wählt eine Zufallszahl, verschüsselt sie mit dem öffentlichen User Key und schickt das Chiffrat an A.
    A entschlüsselt und antwortet mit dem Hashwert der Zufallszahl.

Unter den verschiedenen Varinten der Authentifikation des Benutzers und der Wahl der verwendeten symmetrischen Chiffren bzw. Hashfunktionen kann bei der Konfiguration von SSH ausgewählt werden.

Bei der etwas ungewöhnlichen doppelten Verschlüsselung bei der Authentifikation von H soll die langfristige Komponente (Host Key) verhindern, daß sich ein anderer Rechner als H ausgeben kann. Wird der geheime Schlüssel von H irgendwann kompromittiert, so beschränkt der Server Key den Schaden beim Lesen von alten Nachrichten auf einen Zeitraum von maximal einer Stunde.

7.6 Zero-Knowledge-Protokolle

Bei einem Zero-Knowledge-Protokoll weist ein Beweiser B einem Verifizierer V nach, die Lösung eines bestimmten Problems P zu kennen. Bei diesem Nachweis wird keine Information über die Struktur der Lösung preisgegeben. Genauer ist gefordert, daß bei polynomial vielen Abläufen des Protokolls nicht mehr Information preisgegeben wird, als die, die der Verifizierer selbst mit polynomialem Aufwand hätte bekommen können.

Die Zero-Knowledge-Protokolle können für Authentifikationsverfahren benutzt werden.

Beispiel 7.1. Der Keller hat eine Geheimtür. B will V beweisen, daß er die Geheimtür öffnen kann, ohne aber V das Geheimnis, wie sie geöffnet wird, zu verraten.

1. V steht bei e.
2. B geht entweder zu g oder h im Keller.
3. V geht zu f und ruft B zu, er solle links bzw. rechts herauskommen.
4. B öffnet bei Bedarf die Geheimtür und kommt am gewünschten Ort heraus.
5. Diese Prozedur wird n-mal wiederholt. Danach ist die Wahrscheinlichkeit, daß B in Wirklichkeit die Geheimtür gar nicht öffnen kann und jedesmal nur Glück hatte, nur $1/2^n$.

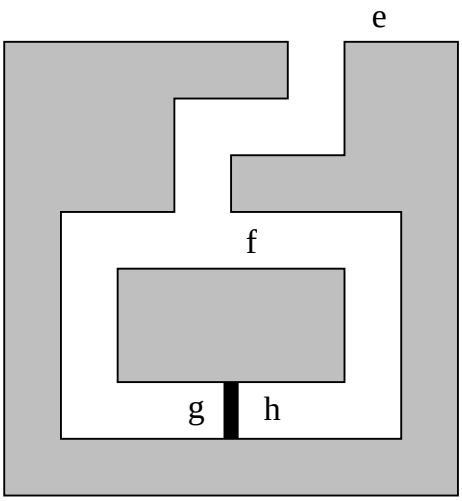


Abbildung 7.1: Keller mit Geheimtür

Um beweisen zu können, daß durch den Protokollablauf keine Information über das Geheimnis preisgegeben wird, greift man auf einen Simulator zurück. Dieser soll ohne Kenntnis des Geheimnisses ein Protokoll erzeugen, das einem echten Protokollablauf gleicht. Der Simulator darf dazu lediglich einzelne Durchläufe des Protokolls löschen.

Ein Verifizierer kann sich mit Hilfe eines Simulators auch ohne Kenntnis des Geheimnisses einen Protokollablauf erzeugen, der so aussieht wie ein echter. Aus einem echten Protokollablauf kann er daher nicht mehr lernen, als er sich nicht auch selber erzeugen könnte. Diese ist die Grundidee um den Begriff „zero-knowledge“ zu definieren und die typische Struktur eines kryptographischen Zero-Knowledge-Protokolls abzuleiten.

Typische Struktur eines Zero-Knowledge-Protokolls (Σ-Protokoll)

Der Beweiser B will dem Verifizierer V beweisen, die Lösung eines Problems P aus einer Problemklasse $\mathcal{P}$ zu kennen:

1. B bestimmt ein zu P äquivalentes Problen $P' \in \mathcal{P}$ mit Hilfe einer Transformation $\tau : P \mapsto P'$ und sendet P' als Referenzwert an V.

2. V kann sich für eine Herausforderung (challenge) r entscheiden. Im Regelfall ist dies die Frage, ob er die Lösung des Problems P' oder die Transformation $\tau : P \mapsto P'$ sehen will.

3. B sendet die zum Referenzwert P' und der Herausforderung r passende Antwort.[1]

Kennt der Beweiser die Herausforderung r, bevor er seinen Referenzwert P' offenlegen muß, so braucht er die Lösung gar nicht zu kennen, da er sich zuerst die Antwort wählen

[1] Transformation schwierig!

kann und dann einen zur Herausforderung passenden Referenzwert wählen kann. Daraus ergibt sich:

Typische Strukur eines Simulators

1. Wähle die Herausforderung r' zufällig und gleichverteilt.
2. Berechne einen zu r' passenden Referenzwert und eine dazugehörige Antwort.
3. Der Verifizierer stellt die Herausforderung r.
4. Falls $r \neq r'$ gehe zu Schritt 1.
5. Gebe den Protokollablauf mit Referenzwert, Herausforderung und Antwort aus.
6. Weiter mit Schritt 1.

Definition 7.2 (Perfekt zero-knowledge). Ein Protokoll heißt *perfekt zero-knowledge*, wenn es einen Simulator gibt, der mit polynomialen Aufwand das Protokoll so simuliert, daß der Transkript identisch zur richtigen Protokollausführung ist. Der Simulator kennt dabei das Geheimnis nicht.

Die abstrakte Definition wird weiter unten eingehender erläutert. Es läßt sich auch eine etwas schwächere Definition erstellen, die die absolute Sicherheit durch *nicht berechenbar mit heutigen Mitteln* ersetzt:

Definition 7.3 (Berechenbar zero-knowledge). Ein Protokoll ist *berechenbar zero-knowledge*, wenn es einen Simulator gibt, der mit polynomialen Aufwand das Protokoll so simuliert, daß es mit heutigen Mitteln nicht von einer richtigen Protokollausführung zu unterscheiden ist.

Berechenbar zero-knowledge ist aus systematischer Sicht nicht wirklich zufriedenstellend, in vielen Fällen ist es aber das Einzige, was über ein Protokoll bewiesen werden kann. Diese Einschränkung sollte man aber nicht zu hoch bewerten; die Zero-Knowledge-Eigenschaft sagt ja nur etwas darüber aus, ob durch den Protokollablauf Information über das Geheimnis preisgegeben wird, über die Sicherheit des Protokolls macht diese Eigenschaft aber keinerlei Aussage.

Der Simulator ist das zentrale Instrument, mit dem die Zero-Knowledge-Eigenschaft definiert und bewiesen wird. Die Aufgabe dieses Simulators ist es, in Interaktionen mit einem echten Verifizierer die Rolle des Beweisers zu übernehmen. Dabei steht dem Simulator das Geheimnis des Beweisers nicht zur Verfügung. Gefordert ist nun, daß er es in einem nicht vernachlässigbaren (polynomial großen) Teil der Protokollausführungen schafft, die erwarteten Antworten zu geben und das Protokoll damit korrekt ablaufen zu lassen. Der Simulator stellt einerseits keinen erfolgreichen Angriff auf das Protokoll dar, da er bei konkreten Implementierungen nur in einem Teil der Fälle die richtigen Antworten geben

kann. Andererseits sind die Antworten, in denen der Simulator korrekte Antworten geben konnte, nicht von den Antworten des Beweisers unterscheidbar. Daher kann sich ein Verifizierer mit Hilfe des Simulators beliebig (polynomial) viele Ausführungen des Protokolls beschaffen, ohne mit dem Beweiser interagieren zu müssen. Der Verifizierer erhält daher durch den Ablauf des Protokolls mit dem Beweiser keine Informationen, die er nicht auch mit dem Simulator hätte erhalten können. Deswegen kann der Beweiser sicher sein, daß er keine Information über sein Geheimnis preisgibt, wenn er das Protokoll polynomial oft ausführt.

Beobachtung 7.4. *Der Keller mit Geheimtür ist perfekt zero-knowledge. Was würde passieren, wenn der Eingang keinen Knick machen würde?*

Hier zeigt sich sowohl die Grenze des Beispiels, als auch die der heuristischen Definition, die wir hier gegeben haben. Eine exakte Definition über akzeptierende Automaten ist sehr unanschaulich, so daß hier davon abgesehen wird.

Eine Möglichkeit für die Realisierung von Zero-Knowledge-Protokollen ist relativ offensichtlich. Diese Möglichkeit basiert auf einem Beweiser, der behauptet, eine Lösung *einer Klasse* von Problemen zu besitzen. Der Beweis kann dann einfach angetreten werden, indem der Verifizierer ein Problem aus dieser Klasse benennt und der Beweiser eine Lösung für dieses Problem vorlegt.

Beispiel 7.5 (Faktorisieren). Der Beweiser behauptet, effizient große Zahlen faktorisieren zu können.

1. V gibt eine große Zahl an.
2. B faktorisiert sie und V überprüft das Ergebnis.

Der Simulator S wählt sich einige Primzahlen und berechnet das Produkt r'. Falls der Verifizierer V eine Zahl r wählt mit $r = r'$, so kann S ihre Faktorisierung angeben. Das Problem ist, daß die Wahrscheinlichkeit für $r = r'$ unendlich klein ist, da V jede der unendlich vielen Zahlen wählen kann. Gefordert ist aber, daß der Simulator in einem nicht vernachlässigbaren Teil der Protokollausführungen die erwarteten Antworten geben kann.

Beispiel 7.6 (Kenntnis von Quadratwurzeln). Es sei n ein öffentlich bekannter, genügend großer Modulus. Die Faktorisierung sei nicht öffentlich bekannt. Das Berechnen von Quadratwurzeln modulo n ist äquivalent zu der Faktorisierung von n und damit schwierig. Es sei y_B ein öffentlich bekannter Wert. Der Beweiser B behauptet, ein x_B mit $x_B^2 \equiv y_B \bmod n$ zu kennen. Das folgende Protokoll kann benutzt werden, um diese Behauptung zu überprüfen.

1. B wählt x zufällig und legt $y \equiv x^2 \bmod n$ offen.
2. V wählt sich $r \in \{0, 1\}$ und teilt es B mit.

3. B berechnet $z \equiv x_B^r x \bmod n$.
4. V überprüft, ob $z^2 \equiv y_B^r y \bmod n$.

Durch die Multipikation mit x wird das ursprüngliche Problem auf ein neues Problem, die Quadratwurzel aus $y_B y$ zu ziehen, transformiert. Falls V diese Quadratwurzel sehen will, kann B sie berechnen. Falls aber V die Transformation sehen will, so liefert V einfach die Zahl x. Die Transformation ist hier auch durch Wurzelziehen zu berechen und deshalb genauso schwierig wie das ursprüngliche Problem.

Um einen Simulator zu konstruieren, halten wir uns an die oben beschriebene typische Struktur eines Simulators. Der Simulator wählt zufällig und gleichverteilt die Herausforderung $r' \in \{0, 1\}$. Falls der Verifizierer die andere Herausforderung $r \neq r'$ wählt, gibt der Simulator auf. Sonst kann er die richtigen Antworten geben:

Simulator S: $r' = 0$

1. S wählt x zufällig und legt $y \equiv x^2$ offen.
2. V entscheidet sich für $r = 0$.
3. S berechnet $z := x_B^0 x \equiv x$.
4. V überprüft, ob $z^2 \equiv y_B^0 y$.

Simulator S: $r' = 1$

1. S wählt w zufällig und legt $v \equiv y_B^{-1} w^2$ offen.
2. V entscheidet sich für $r = 1$.
3. S berechnet $z := x_B \cdot (x_B^{-1} \cdot w) = w$.
4. V überprüft, ob $z^2 \equiv y_B^1 v$.

In der Hälfte der Fälle liefert der Simulator richtige Antworten, d.h. das Protokoll ist ein Zero-Knowledge-Protokoll. Es ist sogar perfekt zero-knowledge, da der Simulator nur quadratische Reste veröffentlicht, sofern andersweitig sichergestellt ist, daß y_B ein quadratischer Rest ist.

Beispiel 7.7 (Graphenisomorphie). Der Beweiser kennt einen Isomorphismus ϕ zwischen den Graphen G_1 und G_2 und will dies dem Verifizierer beweisen. Dazu eignet sich folgendes Protokoll:

1. B transformiert den Graphen G_1 durch eine zufällige Permutation τ und erhält einen neuen Graphen $H = \tau(G_1)$.
2. V wählt, ob er $G_1 \mapsto H$ oder $G_2 \mapsto H$ sehen will.

3. B gibt den geforderten Isomorphismus an, da er die beiden Isomorphismen

$$\begin{aligned} \tau &: G_1 \mapsto H \\ \tau \circ \phi^{-1} &: G_2 \mapsto H \end{aligned}$$

kennt.

Das Protokoll ist berechenbar zero-knowledge, da zunächst nicht klar ist, ob G_1 und G_2 isomorph sind. Ist dies andersweitig sichergestellt, so ist auch dieses Protokoll, wie das obere, perfekt zero-knowledge.

7.7 Digital Cash

Ein ideales System für „Digital Cash“ soll folgende sechs Eigenschaften erfüllen:

- *Unabhängigkeit*: das digitale Bargeld kann über Netze verschickt und gespeichert werden.
- *Sicherheit*: das digitale Bargeld kann nicht kopiert und mehrfach benutzt werden.
- *Nichtzurückverfolgbarkeit*: Die Privatsphäre des Benutzers ist gewahrt. Keiner kann feststellen, welche Transaktionen vom Benutzer vorgenommen wurden.
- *Off-line Bezahlung*: Bei der Bezahlung mit dem digitalen Bargeld ist keine Kommunikation mit der Bank notwendig. Die Bezahlung erfolgt off-line.
- *Übertragbarkeit*: Das digitale Bargeld kann an andere Benutzer übertragen werden.
- *Teilbarkeit*: Das digitale Bargeld kann in kleinere Beträge aufgeteilt werden.

Das erste vorgeschlagene Protokoll, das alle diese Punkte erfüllte, benötigt 200 MByte Datentransfer für eine Buchung. Mitterweile existieren Protokolle, die nur 20 KByte Datentransfer brauchen und in wenigen Sekunden arbeiten.

Im folgenden präsentieren wir ein Protokoll, das die ersten vier Eigenschaft erfüllt. Wir werden das Protokoll schrittweise ausbauen, um Sicherheitslücken zu beseitigen. Die Änderungen zum jeweils vorangegangenen Protokoll sind kursiv gedruckt.

7.7.1 Erste Protokolle

1. Protokoll

1. Alice stellt 100 anonyme Bankschecks über jeweils 1000 € aus.

2. Alice steckt jeden Bankscheck zusammen mit Durchschlagspapier in einen Umschlag und trägt dann alle Umschläge zur Bank.

3. Die Bank öffnet 99 Umschäge und prüft, ob der richtige Betrag (1000 €) auf jedem Scheck von Alice eingetragen wurde.

4. Die Bank unterzeichnet den letzten ungeöffneten Umschlag. Die Unterschrift der Bank wird durch das Durchschlagspapier auf den Scheck übertragen. Die Bank gibt den ungeöffneten Umschlag an Alice zurück und belastet ihr Konto mit 1000 €.

5. Alice öffnet den Umschlag und verwendet den Scheck bei einem Händler.

6. Der Händler überprüft die Unterschrift der Bank und akzeptiert den Scheck.

7. Der Händler bringt den Scheck zur Bank.

8. Die Bank überprüft ihre Unterschrift und zahlt 1000 € dem Händler aus.

Bei diesem Protokoll kann die Bank nicht feststellen, von wem der Scheck kommt. Die Wahrscheinlichkeit, daß ein Betrug von der Bank unentdeckt bleibt, beträgt durch die 1-aus-100-Wahl lediglich 1/100. Durch die Einführung einer Geldstrafe für Betrug lohnt es sich nicht für Alice zu betrügen.

Allerdings gibt es noch eine Sicherheitslücke, da Alice oder der Händler den Scheck kopieren und mehrfach verwenden könnten. Mit dem folgenden Protokoll soll diese Sicherheitslücke beseitigt werden.

2. Protokoll

1. Alice stellt 100 anonyme Bankschecks über jeweils 1000 € aus. *Auf jeden Scheck trägt sie eine Zufallszahl ein, die lang genug ist, daß die Wahrscheinlichkeit, daß eine andere Person dieselbe Zufallszahl benutzt, verschwindend klein ist.*

2. Alice steckt jeden Bankscheck zusammen mit Durchschlagspapier in einen Umschlag und trägt alle Umschläge zur Bank.

3. Die Bank öffnet 99 Umschäge und prüft, ob der richtige Betrag (1000 €) *und eine jeweils andere Zufallszahl* auf jedem Scheck von Alice eingetragen wurde.

4. Die Bank unterzeichnet den letzten ungeöffneten Umschlag. Die Unterschrift der Bank wird durch das Durchschlagspapier auf den Scheck übertragen. Die Bank gibt den ungeöffneten Umschlag an Alice zurück und belastet ihr Konto mit 1000 €.

5. Alice öffnet den Umschlag und verwendet den Scheck bei einem Händler.

6. Der Händler überprüft die Unterschrift der Bank und akzeptiert den Scheck.

7. Der Händler bringt den Scheck zur Bank.

8. Die Bank überprüft ihre Unterschrift und *schaut in einer Datenbank nach, ob ein Scheck mit derselben Zufallszahl eingelöst wurde. Falls nicht, dann* zahlt die Bank dem Händler 1000 €. aus. *Die Bank speichert die Zufallszahl in ihrer Datenbank ab.*

9. *Falls aber ein Scheck mit derselben Zufallszahl eingelöst wurde, dann akzeptiert die Bank den Scheck nicht.*

Die Bank kann jetzt zwar einen Betrug feststellen, kann aber den Betrüger nicht identifizieren. Der Händler ist im Nachteil, da er nicht feststellen kann, ob Alice ihm eine Kopie des Schecks aushändigt.

3. Protokoll

1. Alice stellt 100 anonyme Bankschecks über jeweils 1000 € aus. Auf jeden Scheck trägt sie eine Zufallszahl ein, die lang genug ist, um die Wahrscheinlichkeit, daß eine andere Person dieselbe Zufallszahl benutzt, verschwindend klein ist.

2. Alice steckt jeden Bankscheck zusammen mit Durchschlagspapier in einen Umschlag und trägt alle Umschläge zur Bank.

3. Die Bank öffnet 99 Umschäge und prüft, ob der richtige Betrag (1000 €) und eine jeweils andere Zufallszahl auf jedem Scheck von Alice eingetragen wurde.

4. Die Bank unterzeichnet den letzten ungeöffneten Umschlag. Die Unterschrift der Bank wird durch das Durchschlagspapier auf den Scheck übertragen. Die Bank gibt den ungeöffneten Umschlag an Alice zurück und belastet ihr Konto mit 1000 €.

5. Alice öffnet den Umschlag und verwendet den Scheck bei einem Händler.

6. Der Händler überprüft die Unterschrift der Bank.

7. *Der Händler fordert Alice auf, einen von ihm vorgegebenen Zufallsstring auf den Scheck zu schreiben.*

8. *Alice tut dies.*

9. Der Händler bringt den Scheck zur Bank.

10. Die Bank überprüft ihre Unterschrift und schaut in einer Datenbank nach, ob ein Scheck mit derselben Zufallszahl eingelöst wurde. Falls nicht, dann zahlt die Bank dem Händler 1000 € aus. Die Bank speichert die Zufallszahl *und den Zufallsstring* in ihrer Datenbank ab.

11. Falls aber ein Scheck mit derselben Zufallszahl eingelöst wurde, dann akzeptiert die Bank den Scheck nicht. *Sie überprüft den Zufallsstring auf dem Scheck mit dem in der Datenbank. Falls sie übereinstimmen, weiß die Bank, daß der Händler den Scheck kopiert hat. Falls sie verschieden sind, dann weiß die Bank, daß die Person, die den Scheck gekauft hat, ihn kopiert hat.*

Damit ist ein Betrug des Händlers, der sich der Bank zu erkennen gibt, nicht mehr möglich, die anonyme Alice ist aber in der Lage zu betrügen. Um Alices Anonymität zu sichern, solange sie sich korrekt verhält, ihre Identität aber offenzulegen, sobald sie versucht, zu betrügen, bedarf es einiger Hilfsmittel, die jetzt bereitgestellt werden. Weiterhin ist noch eine elektronische Version der „Unterschrift mit Durchschlagspapier“ zu finden.

7.7.2 Hilfsprotokolle

Blinde Unterschrift

Die Bank soll eine Nachricht m von Alice ungesehen unterschreiben. Für die Signatur wird das RSA-Signaturverfahren benutzt. Die Bank verwendet ihren geheimen Schlüssel d. Der öffentliche Schlüssel (e, n) ist allgemein zugänglich.

Alice berechnet

$$t := m \cdot k^e \pmod{n},$$

wobei k eine Zufallszahl ist. Die Bank signiert t, indem sie t^d berechnet und das Ergebnis Alice liefert. Alice berechnet

$$s := t^d \cdot k^{-1} \equiv m^d \cdot (k^e)^d \cdot k^{-1} \equiv m^d \cdot k^1 \cdot k^{-1} \equiv m^d \pmod{n}$$

und erhält die mit dem geheimen Schlüssel der Bank signierte Nachricht m^d. Die Bank kennt die Nachricht m nicht.

Die blinde Unterschrift entspricht dem Unterschreiben des Umschlags durch die Bank, wobei die Unterschrift durch das Durchschlagspapier auf den Scheck übertragen wird.

Verschleiern einer Nachricht

Alice veröffentlicht $f(I)$, wobei f eine bekannte Einwegfunktion ist. Jeder kann nach der Bekanntgabe von I verifizieren, daß auch wirklich $f(I)$ veröffentlicht wurde.

Mit dieser Technik läßt sich das Problem lösen, die Anonymität einer ehrlichen Alice zu sichern und bei einem Betrug doch die Identität des Betrügers zu erhalten:

Alice spaltet eine Bitfolge I, die ihre Identität vollständig beschreibt (Name, Adresse und alle anderen Informationen, die die Bank verlangt), in zwei Bitfolgen derart auf, daß $I = I_1 \oplus I_2$ gilt. (Dies geschieht z. B., indem man die Nachricht mit einem One-Time-Pad I_1 zu $I_2 := I \oplus I_1$ verschlüsselt.) Dann veröffentlicht Alice einige dieser „halbierten Identitäten“ in der Form $f(I_{i_1})$ und $f(I_{i_2})$. Läßt sich erreichen, daß bei korrektem Verhalten von jedem Paar nur eine Hälfte offengelegt wird, bei einem Betrug jedoch von mindestens einem Paar beide Teile, so ist diese scheinbar widersprüchliche Anforderung von Anonymität und Bekanntgabe der Identität erfüllt.

7.7.3 Ein vollständiges Protokoll

1. Alice stellt n anonyme Bankschecks über einen festen, jeweils identischen Betrag aus. Auf jeden Scheck trägt sie eine genügend lange Zufallszahl ein. *Auf jedem Scheck wird m mal der Identitätsstring von Alice eingetragen, jeweils auf verschiedene Weisen halbiert und verschleiert (s.o.).*

2. Alle n Schecks werden von Alice für eine blinde Unterschrift der Bank vorbereitet.

3. Alice öffnet nach Wahl der Bank $n - 1$ dieser Schecks; die Bank prüft den Betrag, die Zufallszahl und *alle (von Alice auch offengelegten) Identitätsstrings.*

4. Die Bank signiert den letzten Scheck, gibt ihn Alice und belastet ihr Konto.

5. Alice entfernt die Verschleierung (d. h. den Faktor k aus der blinden Unterschrift) und verwendet den Scheck bei einem Händler.

6. Der Händler überprüft die Unterschrift der Bank.

7. *Der Händler fordert Alice auf, von den m Identitätsstrings jeweils, nach seiner Wahl, die linke oder rechte Hälfte bekanntzugeben.*

8. Alice tut dies.

9. Der Händler bringt den Scheck zur Bank.

10. Die Bank überprüft ihre Unterschrift und schaut in einer Datenbank nach, ob ein Scheck mit derselben Zufallszahl eingelöst wurde. Falls nicht, dann zahlt die Bank dem Händler den Betrag aus. Die Bank speichert die Zufallszahl *und die offengelegten Hälften der Identitätsstrings* in ihrer Datenbank ab.

11. Falls aber ein Scheck mit derselben Zufallszahl eingelöst wurde, dann akzeptiert die Bank den Scheck nicht. *Sie vergleicht die offengelegten Hälften der Identitätsstrings mit denen des Schecks in der Datenbank. Falls sie alle übereinstimmen, weiß die Bank, daß der Händler den Scheck kopiert hat. Falls sie verschieden sind, dann weiß die Bank, daß die Person, die den Scheck gekauft hat, ihn kopiert hat. Da der zweite Händler höchst wahrscheinlich eine andere Kombination der halbierten Identitäten hat offenlegen lassen, kennt die Bank jetzt von einem der Identitätsstrings beide Hälften und damit die Identität des Betrügers.*

Damit ist ein Protokoll beschrieben, das die ersten vier oben geforderten Eigenschaften für digitales Geld erfüllt und keinen übertrieben großen Speicher- und Kommunikationsaufwand, weder beim Aussteller noch beim Benutzer, erfordert.

Literaturverzeichnis

[1] W.R. Alford, A. Granville, C. Pomerance; *There are infinitely many Carmichael numbers*; Ann. Math. (140), 1998, S. 703–722.

[2] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment.* Springer, 2004.

[3] D. Coppersmith, A. M. Odlyzko, R. Schroeppel; *Discrete Logarithms in $GF(p)$*; Algorithmica, 1986; S. 1–15.

[4] D. Denning; *Cryptography and Data Security*; Addison Wesley, London, 1983.

[5] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone; *Handbook of Applied Crytography*; CRC Press, 1996 oder http://www.cacr.math.uwaterloo.ca/hac/.

[6] A. M. Odlyzko; *Discrete logarithms in finite fields and their cryptographic significance*; Proceedings of Eurocrypt 1984; T. Beth, N. Cot, I. Ingemarson (Editoren); LNCS 209; Springer; S. 224–314; 1985.

[7] R. Pinch; *The Carmichael numbers up to* 10^{15}; Math. Comp. (61:203), 1993, S. 381–391.

[8] F. Schaefer-Lorinser; *Algorithmische Zahlentheorie*; Vorlesungsskript, Universität Karlsruhe.

[9] B. Schneier; *Applied Cryptography*; 2. Ausgabe, Wiley, 1996.

[10] J. Seberry, J. Pieprzyk; *Cryptography*; Prentice Hall, New York, 1989.

[11] G. J. Simmons (Ed.); *Contemporary Cryptology*; IEEE Press, 1992.

[12] D. R. Stinson; *Cryptography, Theory and Practice*; CRC Press, Boca Taton, Florida, 1995.

Index

aktiver Angriff, 7
aktiver Betrug, 7
Algorithmus
    Dixon, 55
    probabilistischer, 40
    quadratisches Sieb, 57
Algorithmus, asymmetrischer, 3
Algorithmus, Public-Key, 3
Angriff
    aktiver, 7
    brute-force, 4, 10
    chosen-ciphertext, 4
    chosen-plaintext, 3
    ciphertext-only, 3
    Hashfunktionen, 10
    known-plaintext, 3
    man-in-the-middle, 13
    meet-in-the-middle, 10
    passiv, 7
Angriffe gegen Protokolle, 7
Arithmetik, Hauptsatz der, 29
asymmetrischer Algorithmus, 3
Authentifikation, 82, 83
Authentizität, 2, 69

Baby-Step Giant-Step, 61
Bausteine für Protokolle, 8
berechenbar zero-knowledge, 94
Betrug
    aktiver, 7
    passiv, 7
Beweiser, 92
blinde Unterschrift, 100
Blockchiffre, 3
Blockchiffre als Hashfunktion, 10
Brute-Force-Angriff, 4, 10

Carmichael Zahl, 40
Chiffrat, 1
Chinesischer Restsatz, 20
chosen-ciphertext, 4
chosen-plaintext, 3
ciphertext-only, 3

Diffie, 8, 88
Diffie-Hellman-Schlüsselaustausch, 8, 88
Digital Cash, 97
Digitale Signatur, 69
diskrete Logarithmen, 61
Dixon, 55
Dixon Algorithmus, 55
DSA, 76

EC-DSA, 76
Einwegfunktion, 8, 82
    mit Falltür, 9
Element
    primitives, 66
ElGamal, 22–23, 75
elliptische Kurve, 49
    faktorisieren, 49
elliptische Kurve über Ringen, 51
endlicher Körper, 64
    Existenzsatz, 65
    multiplikative Gruppe, 66
Entschlüsselung, 1
Erweiterungskörper, 64
EUF-CMA-Sicherheit, 74
Euklid
    Satz von, 29
Euler

Satz von, 33
Euler Kriterium, 36
Euler-Kriterium, 40
Euler-Pseudoprimzahl, 40
eulersche φ-Funktion, 32
Existenzsatz, endlicher Körper, 65

φ-Funktion, 33
eulersche, 32
Multiplikativität, 33
Primpotenz, 33
Faktorbasis, 63
faktorisieren, 47, 49
Dixon Algorithmus, 55
elliptische Kurve, 49
mit Gruppen, 47
Pollards $p-1$-Methode, 47
quadratische Kongruenz, 53
quadratisches Sieb, 57
zero-knowledge, 95
Falltür, 9
Fermat
kleiner Satz, 33
Funktion
Einweg-, 8
Funktionen
Hash-, 9
kryptographische Hash-, 10

geheimer Schlüssel, 3
Giant-Step Baby-Step, 62
glatt, 55, 66
Graphenisomorphie, zero-knowledge, 96
Guy, 82

Hashfunktion
Blockchiffre als, 10
Signatur mit, 70
Hashfunktionen, 9
Angriff, 10
kryptographische, 10
Hauptsatz der Arithmetik, 29
Hellman, 8, 88

IND-CCA-Sicherheit, 27
Index Calculus, 63
Index-Calculus, 66, 68
informelles Protokoll, 5
Integrität, 2, 69
Interlock-Protokoll, 13
irreduzibel, 65

Jacobi Symbol, 37
Rechenregeln, 37

Kerberos, 85
Key-Substitution-Angriff, 78
Klartext, 1
kleiner Satz von Fermat, 33
known-plaintext, 3
Körper
endlicher, 64
Restklassen-, 64
Körpererweiterung, 64
Kongruenz
quadratische, faktorisieren, 53
Kriterium von Euler, 40
Kryptoanalyse, 3
Kryptographie, 2
Aufgaben, 2
kryptographische Hashfunktion, 10
Kurve
elliptische, 49
elliptische, faktorisieren, 49
elliptische, über Ringen, 51

$L(n)$, 54
Legendre Symbol, 34
Logarithmen, diskrete, 61

Man-in-the-middle-Angriff, 13
McEliece, 23–24
Meet-in-the-middle-Angriff, 10
Merkles Puzzle, 7
Miller, 43
multiplikative Gruppe, endlicher Körper, 66

Nachricht, 1
verschleiern, 100
Needham, 82

Needham-Schroeder, 85
Neuman-Stubblebine, 86
Notar, 6
    Protokoll mit, 6

öffentlicher Schlüssel, 3

$\pi(x)$, 30
$p-1$-Methode von Pollard, 47
passiver Angriff, 7
passiver Betrug, 7
perfekt zero-knowledge, 94
Pohlig-Hellman, 62
Pollard, 47
Pollards $p-1$-Methode, 47
PPT-Algorithmus, 25
primitives Element, 66
Primkörper, 64
Primtest
    nach Solovay und Strassen, 40
    probabilistischer, 40
Primzahl, 29
    Pseudo-, 39
Primzahlsatz, 30
probabilistischer Algorithmus, 40
probabilistischer Primtest
    Rabin und Miller, 43
    Solovay und Strassen, 40
Protokoll, 5
    Angriffe, 7
    Bausteine, 8
    informelles, 5
    Interlock-, 13
    Schlüsselaustausch, 11
    Zero-Knowledge, 92
Protokoll mit Notar, 6
Pseudoprimzahl, 38, 39
    Euler-, 40
Public-Key-Algorithmus, 3
    Merkles Puzzle, 7
Public-Key-Kryptosystem, 26
Public-Key-Signatursystem, 73
Public-Key-Verfahren
    Signatur, 70

quadratische Kongruenz
    faktorisieren, 53
quadratische Reste, 34
quadratisches Reziprozitätsgesetz, 36
quadratisches Sieb, 57
Quadratwurzeln
    zero-knowledge, 95

Rabin, 43
Rabin und Miller, probabilistischer Primtest, 43
Reaction Attack, 24
Rechenregeln, Jacobi Symbol, 37
Reste, quadratische, 34
Restklassenkörper, 64
Reziprozitätsgesetz, quadratisches, 36
RSA, 9, 19–22, 71
RSAES-OAEP, 20, 27–28

Satz
    von Euklid, 29
    von Euler, 33
    von Fermat (kleiner), 33
    von Solovay und Strassen, 41
    von Tschebyscheff, 30
Schlüssel, 2
    geheimer, 3
    öffentlicher, 3
Schlüsselaustausch, 11, 84
    Diffie-Hellman, 8, 88
    mit Signatur, 14
Secure Shell, 91
Secure Socket Layer, 90
Sieb, quadratisches, 57
Signatur
    digitale, 69
    DSA, 76
    EC-DSA, 76
    ElGamal, 75
    Key-Substitution-Angriff, 78
    mit subliminalem Kanal, 80
    Public-Key, 70
    RSA, 71
    Schlüsselaustausch mit, 14

Signatur mit Hashfunktion, 70
Simulator, 94
Solovay, 40
Solovay und Strassen, Satz von, 41
SSH, 91
SSL, 90
Strassen, 40
Stromchiffre, 3
Subliminaler Kanal, 80
Symbol
    Jacobi, 37
    Legendre, 34
symmetrischer Algorithmus, 3

TLS, 90
Transport Layer Security, 90
Tschebyscheff, Satz von, 30

Unabstreitbarkeit, 69
Unterkörper, 64
Unterschrift
    blinde, 100

Verbindlichkeit, 2
Verifizierer, 92
vernachlassigbar, 25
verschleiern einer Nachricht, 100
Verschlüsselung, 1
Vertraulichkeit, 2

Wide-mouth frog, 84

Zahl
    Carmichael, 40
Zerfällungskörper, 65
zero-knowledge
    berechenbar, 94
    Graphenisomorphie, 96
    perfekt, 94
    Quadratwurzeln, 95
Zero-Knowledge-Protokoll, 92

überwältigend, 25