Digitale Signatursysteme

Inhaltsverzeichnis

1 Einführung

In den bisherigen Versuchen wurden kryptographische Verfahren zur Geheimhaltung von Nachrichten vorgestellt. Neben der **Geheimhaltung** („=“ Vertraulichkeit „=“ Konzelation) ist die Sicherstellung von **Authentizität** ein zweites zentrales Sicherheitsziel [DiHe_79]. Dadurch soll zweierlei gewährleistet werden:

1. Der Empfänger einer Nachricht kann sicher sein, daß diese von dem angegebenen Sender stammt, und

2. jede Veränderung der Nachricht auf dem Transportweg kann entdeckt werden.

Dieses Sicherheitsziel spielt eine zentrale Rolle beispielsweise beim Austausch von Schlüsseln für symmetrische Konzelationssysteme über unsichere Kanäle: Will ein aktiver Angreifer die Kommunikation zwischen Teilnehmer A und Teilnehmer B abhören, so kann er an beide einen selbstgewählten symmetrischen Schlüssel schicken und sich dabei als A bzw. B ausgeben. Wird er dabei nicht entdeckt, kann er anschließend alle mit diesem Schlüssel konzelierten Nachrichten mitlesen. Eine wichtige Rolle spielt Authentizität auch im digitalen Zahlungsverkehr.

In diesem Versuch werden zwei kryptographische Verfahren zur Gewährleistung von Authentizität vorgestellt. In den Versuchsaufgaben sind anschließend einzelne zentrale Elemente dieser Verfahren zu programmieren. Dabei werden bereits eingeführte Prozeduren und Funktionen der Versuche “Zahlentheoretische Algorithmen” und “Asymmetrische Konzelationssysteme” zur Anwendung kommen. Die implementierten Prozeduren illustrieren die Funktionsweise der vorgestellten Signatursysteme.

2 Digitale Signatursysteme

Kryptographische Verfahren, die den Empfänger einer Nachricht nicht nur den Sender erkennen lassen, sondern auch einen Urheberschaftsnachweis gegenüber (unparteiischen) Dritten ermöglichen, werden in Anlehnung an eigenhändige Unterschriften **digitale Signatursysteme** genannt. Genau besehen unterscheiden sich digitale von eigenhändigen Unterschriften jedoch in verschiedener Hinsicht: Eigenhändig unterschriebene Dokumente können oft mit etwas Geschick unbemerkt verändert oder ergänzt werden (zum Beispiel eine angehängte 0 auf einem Scheck) und viele Namensunterschriften lassen sich mit wenig Übung imitieren.

Diese Schwächen dürfen digitale Signaturen nicht besitzen, wenn sie verläßlichen Nachrichtenaustausch in einem offenen digitalen System gewährleisten sollen. In Streitfällen sollen sie eine, wenn nötig gerichtliche, Rekonstruierung von Kommunikationsabläufen ermöglichen. Die Herkunft und die Unverfälschtheit einer Nachricht müssen also durch das digitale Signatursystem nachgewiesen werden können.

Symmetrische Kryptosysteme (z.B. DES in der Betriebsart CBCauth, siehe [Pfit_98]) sind ungeeignet für die Erfüllung dieses Sicherheitszieles. Zwar kann der Sender seine Nachricht mit dem durch ein symmetrisches Kryptosystem erhaltenen Schlüsseltext „signieren“. Der Empfänger kann dann prüfen, ob sein Schlüssel „paßt“: Die „Unterschrift“ wird mit demselben Schlüssel und Kryptoverfahren (z.B. DES in der Betriebsart CBCauth) erneut erzeugt und mit der erhaltenen verglichen. Auf diese Weise kann er Herkunft und Unverfälschtheit der Nachricht festgestellen. Im Streitfall kann der Empfänger aber weder das eine noch das andere gegenüber Dritten beweisen, da er ja durch die Kenntnis des symmetrischen Schlüssels selbst in der Lage ist, Nachrichten mit derselben Signatur zu versehen.

Die Nachweisbarkeit gegenüber Dritten setzt demnach zweierlei voraus:

1. Allein der Sender ist in der Lage, gültige Unterschriften für seine Nachrichten zu erzeugen.

2. Die Signaturen dürfen nicht nur vom Empfänger auf ihre Gültigkeit geprüft werden können, sondern von jedem Dritten.

Diese Eigenschaften besitzen nun gerade asymmetrische kryptographische Verfahren. Ein digitales Signatursystem auf der Grundlage asymmetrischer Kryptosysteme zerfällt dann im wesentlichen in drei zentrale Komponenten (siehe Bild 1):

1. Einen Generieralgorithmus $\mathbf{G}$, der ein Schlüsselpaar (s, t) liefert; der öffentliche Schlüssel t wird im allgemeinen in ein Verzeichnis eingetragen, der Schlüssel s geheimgehalten.

2. Einen Signieralgorithmus $\mathbf{S}$ zum Unterschreiben von Nachrichten, der den geheimen Schlüssel s verwendet.

3. Einen Testalgorithmus $\mathbf{T}$, der die Signatur einer Nachricht anhand des öffentlichen Schlüssels t überprüft.

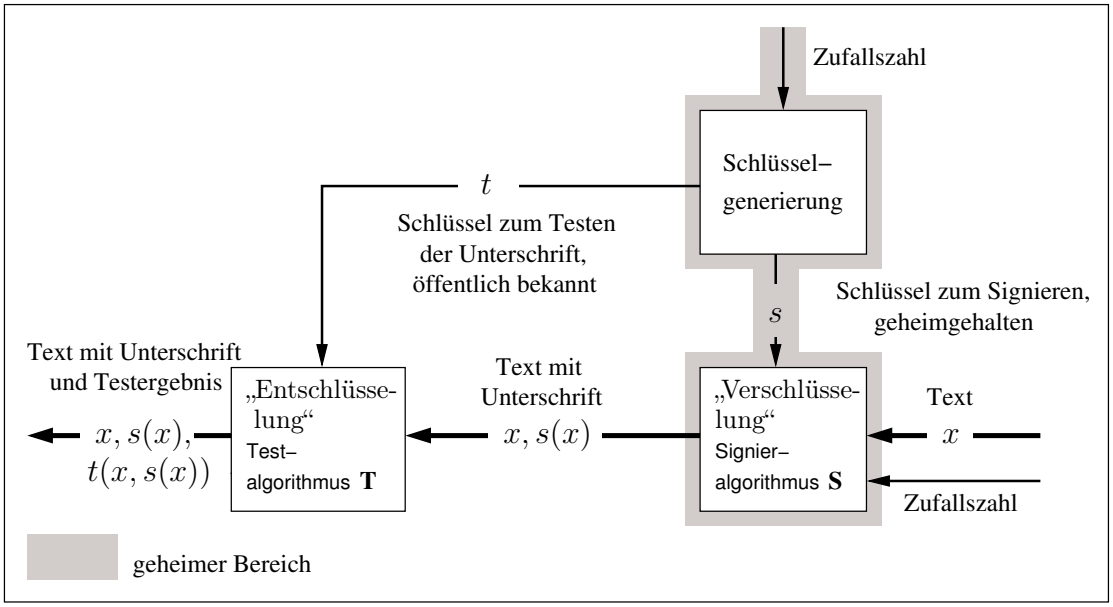

Abbildung 1: Digitales Signatursystem

3 RSA als Digitales Signatursystem

3.1 RSA „pur“

Im Versuch zu asymmetrischen Konzelationssytemen wurde RSA als Beispiel eines asymmetrischen Kryptoverfahrens vorgestellt. Der geheime Schlüssel s besteht dort aus zwei zufällig und unabhängig gewählten großen Primzahlen p und q mit $p \neq q$. Aus diesen wird die erste Komponente des öffentlichen Schlüssels $t = (n, e)$, der Modulus $n = p \cdot q$ berechnet. Die zweite Komponente, der Exponent e, wird so gewählt, daß $\mathrm{ggT}(e, \phi(n)) = 1$.[1] Dann gilt für ein (geheimzuhaltendes) d mit

$$e \cdot d \equiv 1 \pmod{(p-1) \cdot (q-1)}$$

für alle Nachrichten m des Nachrichtenraums [RSA_78]:

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

Den ersten Teil der Gleichung macht man sich zunutze, wenn man das RSA-Verfahren zur Konzelation einsetzen möchte. Jeder Sender einer Nachricht führt eine modulare Exponentiation mit dem öffentlichen Exponenten e bezüglich des Modulus n des Empfängers durch. Nur dieser kann dann mit Hilfe seiner Kenntnis von d oder der Primfaktoren von n (dem geheimen Schlüssel s) die Nachricht entschlüsseln (siehe Versuch “Asymmetrische Konzelationssysteme”).

[1] $\phi(n)$ bezeichnet die Euler'sche ϕ-Funktion. $\phi(n)$ ist definiert als die Anzahl natürlicher Zahlen, die teilerfremd zu und kleiner als n sind. Ist $p \neq q$, dann gilt für $n = p \cdot q$: $\phi(n) = (p-1) \cdot (q-1)$.

Berechnet der Sender einer Nachricht m hingegen $m^d \pmod{n}$ mit seinem eigenen geheimen Schlüssel s und schickt das Ergebnis mit der Nachricht mit, hat er eine digitale Signatur erzeugt, die jeder mit Hilfe des veröffentlichten Schlüsselpaares $t = (e, n)$ überprüfen kann. Der Signieralgorithmus **S** entspricht demnach dem Dechiffrieralgorithmus D (Versuch "Asymmetrische Konzelationssysteme"), der Testalgorithmus **T** kann den Chiffrieralgorithmus C aus dem Versuch "Asymmetrische Konzelationssysteme" verwenden; hinzu kommt ein Vergleich von „chiffrierter Signatur" und erhaltener Nachricht.

Die Signierfunktion läßt sich ebenso wie der Dechiffrieralgorithmus aus dem Versuch "Asymmetrische Konzelationssysteme" beschleunigen, indem

$$\begin{aligned} sig_p &:= m^{d \bmod (p-1)} \pmod{p} \\ sig_q &:= m^{d \bmod (q-1)} \pmod{q} \end{aligned}$$

berechnet und die Teilergebnisse mit Hilfe des **Chinesischen Reste Algorithmus** zum Ergebnis zusammengefügt werden:

$$sig := \mathrm{CRA}(sig_p, sig_q, p, q)$$

In Aufgabe 1 werden die Signier- und die Testfunktion für RSA als Signatursystem implementiert.

Wie im Versuch "Asymmetrische Konzelationssysteme" erläutert, besitzt das Konzelationssystem RSA unterschiedliche Sicherheitsschwächen. Das gilt auch, wenn man RSA als Signatursystem verwendet. So kann RSA durch einen aktiven Angriff gebrochen werden. (Wie bei Konzelationssystemen werden hier aktive und passive Angriffe unterschieden.) Besonders leicht ist das möglich, wenn Signier- und Konzelationsschlüssel übereinstimmen [Denn_84]. Die Sicherheit des RSA-Signatursystems gegen passive Angriffe, d.h. die Äquivalenz von Signaturfälschung und Faktorisierung des Modulus n ist bisher nicht bewiesen.

3.2 Ein aktiver Angriff auf RSA „pur"

Die meisten aktiven Angriffe auf RSA-Signaturen, also Fälschungsversuche, nutzen die **Multiplikativitätseigenschaft** von RSA:

$$x^d \cdot y^d \equiv (x \cdot y)^d \pmod{n}$$

Ein bekannter aktiver Fälschungsangriff,[2] der auf der Multiplikativität von RSA beruht, stammt von Denning [Denn_84]. Er soll hier beispielhaft die Vorgehensweise illustrieren:

1. Der Angreifer wählt eine beliebige Signatur sig_x und eine Nachricht m, deren Unterschrift er fälschen möchte.

2. Dann berechnet er $x := sig_x^e \pmod{n}$ mit dem öffentlichen Schlüssel $t = (e, n)$ des Opfers.

   $$\{ \text{ Es gilt: } sig_x \equiv x^d \pmod{n} \ \}$$

[2] Bei einem aktiven Angriff auf digitale Signatursysteme wird angenommen, daß der Angreifer vom Opfer eine beliebige, endliche Anzahl gültiger Unterschriften zu von ihm gewählten Nachrichten erhält. Der Angriff ist erfolgreich, wenn die Fälschung einer weiteren Signatur gelingt.

3. Nun läßt er das Opfer die Nachricht $(x \cdot m)$ unterschreiben, erhält $sig_{x \cdot m} := (x \cdot m)^d \pmod{n}$.

4. Mithilfe des Euklidschen Algorithmus (Versuch "Zahlentheoretische Algorithmen") berechnet der Angreifer $sig_x^{-1} \pmod{n}$.

$$\{ \text{ Es gilt: } sig_x^{-1} \equiv (x^d)^{-1} \pmod{n} \ \}$$

5. Jetzt kann er die Signatur sig_m seiner gewählten Nachricht m bestimmen:

$$sig_x^{-1} \cdot sig_{x \cdot m} \equiv x^{-d} \cdot (x \cdot m)^d \equiv m^d \equiv sig_m \pmod{n}$$

Auf den ersten Blick mag dieser Angriff eigenartig erscheinen: Wenn das Opfer Nachrichten gewissermaßen „blind" unterschreibt, warum dann nicht auch die gewünschte Nachricht m? Anschaulich kann man sich das Opfer beispielsweise als öffentlichen, digitalen Notar vorstellen, der alles unterschreibt, was das aktuelle Datum oder eine Laufnummer in einem bestimmten Intervall enthält. Will man nun eine „zurückdatierte" Nachricht beglaubigt haben, wird der Notar sich weigern. Mit dem erläuterten Angriff kann man dann die gewünschte Signatur fälschen.

Wohlgemerkt: Durch diesen Angriff wird RSA nicht vollständig gebrochen - den Modulus kann man auf diese Weise nicht faktorisieren. Man spricht bei Signatursystemen in diesem Zusammenhang vom „existentiellen Brechen", wenn eine Signatur zu einer Nachricht eigener Wahl gefälscht wird [GoMR_88].

Der vorgestellte Angriff ist in Aufgabe 2 zu programmieren.

3.3 Verwendung kollisionsresistenter Hashfunktionen

Für lange Nachrichten kann die Wirkung dieser Multiplikativitätseigenschaft abgeschwächt werden, indem wie in Versuch "'Asymmetrische Konzelationssysteme" beschrieben eine Block-Betriebsart (ECB selbstverständlich ausgeschlossen) eingesetzt wird. Einfache aktive Angriffe – wie der oben beschriebene – lassen sich auf diese Weise vereiteln. Mit der Abwehr dieser Protokollschwäche des RSA-Verfahrens nimmt die Sicherheit des RSA-Signatursystems als solchem jedoch nicht (beweisbar) zu.

Verwendet man RSA als digitales Signatursystem, gibt es noch eine vielversprechende andere Gegenmaßnahme. Da die Nachricht bei Signatursystemen im Unterschied zu Konzelationssystemen vom Empfänger nicht aus den erhaltenen Daten zurückgewonnen werden muß (sie wird ja separat mitgeschickt, siehe Bild 1), kann die Nachricht vor der Signierung durch eine **Hashfunktion** h auf einen Hashwert abgebildet werden, um den im vorangegangenen Abschnitt vorgestellten aktiven Angriff zu vereiteln.

Dadurch kann die algebraische Struktur des Verfahrens (z.B. Multiplikativität) zerstört werden. Der Signieralgorithmus **S** wendet vor der Erzeugung der Signatur die Hashfunktion h auf die Nachricht an; signiert wird dann statt der Nachricht m deren Hashwert $\mathrm{h}(m)$. Der Empfänger muß in seinem Testalgorithmus **T** auch zunächst den Hashwert $\mathrm{h}(m)$ berechnen.

Die verwendete Hashfunktion h sollte, möglichst bewiesenermaßen, die Eigenschaft besitzen, **kollisionsresistent** zu sein, d.h. es sollte praktisch unmöglich (= ebenso schwierig wie

Faktorisieren) sein, zu einem Bild h(m) mehrere zugehörige Urbilder (Nachrichten) m zu finden [Denn_84]. Einwegfunktionen, wie sie im Versuch "Asymmetrische Konzelationssysteme" vorgestellt wurden, genügen dieser Bedingung nicht: Ist es möglich, ein zweites Urbild zu einem Funktionswert zu finden, so kann bereits eine Fälschung stattfinden; dazu ist im allgemeinen eine Umkehrung der Funktion (Berechnung aller Urbilder) nicht notwendigerweise erforderlich.

4 Hashfunktionen aus kollisionsresistenten Permutationen

Verwendet man als Hashfunktion beispielsweise DES, wie von Davies und Price vorgeschlagen wurde, kann die Geschwindigkeit des Signaturverfahrens für lange Nachrichten zusätzlich erheblich gesteigert werden [DaPr_80]. Allerdings ist die Kollisionsresistenz von DES bis heute nicht nachgewiesen. Auch der Einsatz von Modulararithmetik (z.B. modulares Quadrieren verketteter Nachrichtenblöcke [DaPr_85, Jung_87]), hat diesen Nachteil; einzelne vorgeschlagene Hashfunktionen wurden bereits gebrochen [Gira_87].

Von Damgård stammt der Vorschlag, die Hashfunktion h aus kollisionsresistenten Permutationen zusammenzusetzen [Damg_88]. **Kollisionsresistent** heißt eine Menge von Permutationen dann, wenn es praktisch unmöglich ist, für wenigstens zwei Permutationen der Menge jeweils ein Urbild so zu finden, daß diese von den Permutationen auf denselben Funktionswert abgebildet werden (siehe Bild 2).

Die Idee geht auf den Ansatz von Goldwasser, Micali und Rivest zurück, der dem im nächsten Abschnitt beschriebenen GMR-Signatursystem zugrundeliegt [GoMR_84, GoMR_88]. Von Damgård wurden konkrete Permutationenmengen vorgeschlagen, für die das Finden einer Kollision bewiesenermaßen äquivalent dem Faktorisieren eines großen Modulus ist [Damg_88, Fox_91].

Dazu wird es zunächst ein wenig formaler: Eine Nachricht m aus dem Nachrichtenraum N sei dargestellt als ein endliches Wort über dem Alphabet[3] $\Sigma = \{0, 1, \ldots, s-1\}$ der Kardinalität s. Sei weiter eine s-elementige Menge $\mathcal{M}$ paarweise **kollisionsresistenter** Permutationen über einem Definitionsbereich D gegeben:

Definition: Kollisionsresistent
Eine Menge von Permutationen $\mathcal{M} := \{f_i \,|\, 0 \leq i < s \in \mathbb{N}\}$ heißt **kollisionsresistent**, wenn

1. alle $f_i \in \mathcal{M}$ auf demselben Definitionsbereich D definiert sind,
2. für alle $x \in \mathrm{D}$ und $f_i \in \mathcal{M}$ gilt: $f_i(x)$ ist leicht zu berechnen, und
3. es praktisch unmöglich ist, eine **Kollision** zu finden, das ist ein Quadrupel (f_i, f_j, x, y) mit $x, y \in \mathrm{D}, x \neq y$ und $f_i, f_j \in \mathcal{M}$ $(0 \leq i, j < s)$ so daß gilt: $f_i(x) = f_j(y)$.

[3] Dieses Alphabet hat nichts zu tun mit dem Alphabet, das von den kollisionsresistenten Permutationen permutiert wird und in aller Regel deutlich mehr Elemente haben muß.

Anschaulich hat beispielsweise die Kollision (f_0, f_1, x, y) die folgende Gestalt (dabei sei, wie gefordert, $x, y \in \mathrm{D}, x \neq y$ und $f_i, f_j \in \mathcal{M}$ $(0 \leq i, j < s)$):

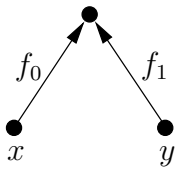

Abbildung 2: Kollision (f_0, f_1, x, y)

Für die Konstruktion kollisionsresistenter Hashfunktionen wird noch eine auf dem Nachrichtenraum N definierte sogenannte **präfixfreie Abbildung** präf : $N \to N$ benötigt (warum, wird im weiteren deutlich):

Definition: Präfixfrei
Eine Abbildung präf : $N \to N$ heißt **präfixfrei** genau dann, wenn für alle $a, b \in \mathrm{N}, a \neq b$ kein Wort $c \in \Sigma^*$ existiert mit präf$(a)c =$ präf(b).

Anschaulicher erklärt bedeutet dies, dass kein gültiges Codewort ein Präfix eines anderen gültigen Codeworts sein darf. Wenn also an ein gültiges (präfixfrei codiertes) Codewort weitere Zeichen angehängt werden, führt dies immer zu einem ungültigen Codewort.

Ist die Nachricht m beispielsweise ein Wort über $\Sigma = \{0, 1\}$ $(s := 2)$, dann kann als präfixfreie Abbildung eine Codierung verwendet werden, die jede 0 durch 00 und jede 1 durch 11 ersetzt und die codierte Nachricht präf(m) mit 01 abschließt [Damg_88, GoMR_88]. Bei dieser Abbildung verdoppelt sich allerdings die Länge des Wortes. Bildet man statt dessen m durch das Voranstellen einer Längenangabe[4] präfixfrei ab, wächst das Wort nur logarithmisch in der Anzahl der Zeichen [Fox_90, FoPf_91].

Jetzt lassen sich die gewünschten Hashfunktionen konstruieren:

Definition: Damgård-Hashfunktion
Eine **Damgård-Hashfunktion** h(m) ist für $m \in N$, präf : $N \to N$ präfixfrei, $f_i \in \mathcal{M}$, präf$(m) := m_1 m_2 \ldots m_r$ mit $m_i \in \Sigma$ und ein zufällig gewähltes $R \in \mathrm{D}$ folgendermaßen definiert:

$$\mathrm{h}(m) := F_{\text{präf}(m)}(R) := f_{m_1}(f_{m_2}(\ldots f_{m_r}(R)\ldots))$$

Die Nachricht m wird also zeichenweise ausgewertet: Jedes Zeichen $m_i \in \Sigma$ wählt die zugehörige Permutation $f_{m_i} \in \mathcal{M}$ aus. Die Permutationen bilden nicht die Nachricht, sondern eine

[4] Die Längenangabe hat eine feste Länge t, gegebenenfalls wird ein Erweiterungsbit gesetzt, falls die Längenangabe nicht in einer Zahl der Länge t möglich ist.

zufällig gewählte Referenz R ab. Diese Referenz wird fest gewählt und mit dem öffentlichen Schlüssel des Signatursytems und der Permutationenmenge $\mathcal{M}$ bekanntgegeben.

Jetzt wird auch deutlich, warum die Nachricht zunächst präfixfrei abzubilden ist: Anderenfalls erhielte ein passiver Angreifer mit der Referenz R und der Nachricht auch die Hashwerte[5] aller Anfangsabschnitte der signierten Nachricht. Alle Zwischenergebnisse des Hashvorgangs dürfen daher keine Hashwerte gültiger Nachrichten sein. Das stellt die präfixfreie Abbildung sicher.

Für die auf diese Weise konstruierten Hashfunktionen gilt [Damg_88]:

Theorem 1: (Kollisionsresistent)

DAMGÅRD-Hashfunktionen h sind **kollisionsresistent**, wenn die Permutationenmengen $\mathcal{M}$ kollisionsresistent sind.

Der Reduktionsbeweis dieses Theorems wird von DAMGÅRD geführt. Gesucht sind nun noch konkrete Hashfunktionen h, die aus bewiesenermaßen kollisionsresistenten Permutationenmengen konstruiert sind. In Anlehnung an die in [GoMR_88] verwendeten kollisionsresistenten Permutationenpaare schlägt DAMGÅRD die folgenden Permutationenmengen vor [Damg_88]:

Theorem 2

$\mathcal{M} = \{f_i | f_i := a_i \cdot x^2 \pmod n, \mathrm{Def}(f_i) = \mathbf{QR}_n, i \neq j \Leftrightarrow a_i \neq a_j, 0 \leq i, j < s\}$,
dabei seien $a_i \in \mathbf{QR}_n$ und $n \in \mathrm{Blum}_k$ gleichverteilt gewählt.

Die Permutationenmenge $\mathcal{M}$ wird also durch die zufällige Wahl des Modulus n sowie der s Koeffizienten $(a_0, \ldots, a_{s-1}) =: a$ bestimmt. Eine solche konkrete Permutationenmenge wird im folgenden mit $\mathcal{M}_{n,a}$ bezeichnet. Ihre Kollisionsresistenz beruht auf dem **Faktorisierungsproblem** (siehe Versuch "Zahlentheoretische Algorithmen").

$\mathbf{QR}_n$ steht dabei für die Menge aller **Quadratischen Reste** modulo n.[6] Blum_k bezeichnet die Menge aller **BLUM-Zahlen** der Länge k (bezüglich der Basis b; siehe auch Versuch "Zahlentheoretische Algorithmen", [Blum_82]):[7]

Definition: Blum_k

$$\mathrm{Blum}_k := \{n \mid n = p \cdot q,\ p, q \text{ prim},\ \log_b(p) \approx \log_b(q) \approx \frac{1}{2}k,\ p \equiv_4 3,\ q \equiv_4 3\}$$

Die Koeffizienten $a_i \in \mathbf{QR}_n$ müssen gleichverteilt gewählt werden. Das kann effizient geschehen, indem $b_i \in \{1, \ldots, n-1\}$, $\mathrm{ggT}(b_i, n) = 1$ gewählt und anschließend modulo n quadriert werden. Diese Zufälligkeit ist für den Sicherheitsbeweis von Bedeutung. Dabei ist, informell formuliert, folgendes zu zeigen [Damg_88]:[8]

[5] Ein Hashwert ist das Ergebnis der Anwendung einer Hashfunktion auf eine Nachricht.

[6] Ein Element a einer Restklasse (mod n) heißt **Quadratischer Rest** genau dann, wenn $x^2 \equiv_n a$ lösbar ist, (siehe Versuch "Zahlentheoretische Algorithmen"). Diese Eigenschaft besitzt etwa ein Viertel aller Zahlen aus $\mathbb{Z}_n^*$.

[7] Die Primfaktoren p und q werden in der gleichen Größenordnung (etwa 350 Bit) gewählt, um Faktorisierungsangriffe zu erschweren.

[8] Die Umkehrung gilt, wie man sich leicht überlegen kann, trivialerweise.

Behauptung

Wenn die Faktorisierung langer Zahlen $n \in \text{Blum}_k$ für ausreichend große k kryptographisch schwer ist[9], dann ist jede Permutationenmenge der folgenden Form kollisionsresistent:

$$\mathcal{M}_{n,a} = \{f_i | f_i := a_i \cdot x^2 \mod n, \text{Def}(f_i) = \mathbf{QR}_n, i \neq j \Leftrightarrow a_i \neq a_j, 0 \leq i, j < s\}$$

dabei seien $a_i \in \mathbf{QR}_n$ und $n \in \text{Blum}_k$ gleichverteilt gewählt.

Der formale Beweis dieser Behauptung wird in [Fox_91] geführt.

Die DAMGÅRD-Hashfunktionen $\text{h}(m)$ sind in Aufgabe 3 zu implementieren. Die Referenz R wird mit dem Modulus $n \in \text{Blum}_k$ bei der Schlüsselgenerierung als Teil des authentisch zu veröffentlichenden Schlüssels $t := (n, R)$ gewählt. Kombiniert man DAMGÅRD-Hashfunktionen mit RSA als Signatursystem, ist die Nachricht m zunächst mit $\text{h}(m)$ abzubilden; der Hashwert wird anschließend signiert. Beim Testen wird die Nachricht bezüglich demselben öffentlichen Schlüssel t „komprimiert“[10] und anschließend die Signatur dieses Hashwertes überprüft.

5 Das Digitale Signatursystem GMR

Erstmals 1984, dann in einer ausgearbeiteten Fassung 1988 stellten GOLDWASSER, MICALI und RIVEST ein digitales Signatursystem (im weiteren **GMR-System** genannt) vor, dessen Sicherheit selbst bei aktiven Angriffen äquivalent der Existenz **kollisionsresistenter Permutationenpaare** (f_0, f_1) ist. Der Sicherheitsbeweis, der von den Autoren geführt wird, gilt für alle Familien kollisionsresistenter Permutationenpaare, gleich auf welchem (mathematischen) Problem sich deren Kollisionsresistenz gründet (z.B. Faktorisierungsproblem, diskreter Logarithmus).

Die im vorausgegangenen Abschnitt eingeführten Mengen kollisionsresistenter Permutationen sind eine Verallgemeinerung der **kollisionsresistenten Permutationenpaare** des GMR-Signatursystems. Deren Konstruktion ähnelt der Darstellung aus Abschnitt 5.4 für $s := 2$ mit festen Koeffizienten a_0 und a_1 [GoMR_88]:

Satz: (kollisionsresistent)

Ein Permutationenpaar (f_0, f_1) heißt **kollisionsresistent**, wenn

1. f_0, f_1 auf demselben Definitionsbereich D definiert sind,
2. für alle $x \in \text{D}$ und $i \in \{0, 1\}$ gilt: $f_i(x)$ ist leicht zu berechnen, und
3. es praktisch unmöglich ist, eine **Kollision** zu finden, das ist ein Quadrupel (f_0, f_1, x, y) mit $x, y \in \text{D}$ so daß gilt: $f_0(x) = f_1(y)$.

[9] D.h. sie ist unter der Annahme realistisch begrenzter Rechenleistung in vernünftiger Zeit praktisch unmöglich.

[10] Das Verb „komprimieren“ ist in diesem Zusammenhang möglicherweise leicht irreführend: Schließlich geht im Unterschied zu Datenkompressionen bei der Abbildung $\text{h}(m)$ Information verloren (ein Hashwert hat unendlich viele Urbilder). Dennoch ist das Wort schöner als die Bezeichnung „hashen“ ...

Goldwasser, Micali und Rivest schlagen konkrete Permutationen f_0 und f_1 vor, deren Kollisionsresistenz auf dem in Versuch "Asymmetrische Konzelationssysteme" erläuterten Faktorisierungsproblem beruht:

$$f_0(x) := \begin{cases} x^2 \pmod n, \text{ falls } (x^2 \bmod n) < \frac{n}{2} \\ -x^2 \pmod n \text{ sonst.} \end{cases}$$

$$f_1(x) := \begin{cases} 4 \cdot x^2 \pmod n, \text{ falls } (4 \cdot x^2 \bmod n) < \frac{n}{2} \\ -4 \cdot x^2 \pmod n \text{ sonst.} \end{cases}$$

Dabei ist $n \in \text{BlumGMR}_k := \{n \mid n = p \cdot q,\ p, q \text{ prim},\ \log_b(p) \approx \log_b(q) \approx \frac{1}{2}k, p \equiv_8 7, q \equiv_8 3\}$ und $\text{D} := \{x \in \mathbb{Z}_n^* \mid \left(\frac{x}{n}\right) = 1,\ x < \frac{n}{2}\}$. $\left(\frac{x}{n}\right)$ bezeichnet hier das **Jacobi-Symbol** von x bezüglich n (siehe Versuch "Zahlentheoretische Algorithmen"). Der Definitionsbereich der Permutationen und die Wahl des Modulus weicht von den Permutationen zur Konstruktion der Damgård-Hashfunktionen in Abschnitt 4 ab. Das hat insbesondere beweistechnische Gründe: Für die (nur einmalig verwendbaren, s.u.) Referenzen R muß vom Empfänger $R \in \text{D}$ getestet werden können. Das geht jedoch nicht, wenn $\text{D} = \mathbf{QR}_n$, da nur der Signierer die Primfaktoren von n kennt.

Die Funktionen F_m werden, wie im vorausgegangenen Abschnitt beschrieben, auf sehr einfache Art und Weise konstruiert: Aus einem kollisionsresistenten Permutationenpaar (f_0, f_1) wird F_m konkateniert, indem f_1 für jedes gesetzte Bit aus m, f_0 für jedes gelöschte angewendet wird. Die Nachricht m wird dabei beginnend mit dem niederwertigsten Bit ausgewertet.

Die Konstruktion der Umkehrfunktionen erfolgt analog durch Konkatenation der Umkehrpermutationen f_0^{-1} und f_1^{-1}; die Nachricht m wird dabei bitweise in umgekehrter Richtung ausgewertet. Deren Berechnung ist etwas komplizierter: Sollen die (mit dem im Versuch "Zahlentheoretische Algorithmen" eingeführten Algorithmus **SQRTMOD*** leicht bestimmbaren) modularen Quadratwurzeln wieder im Definitionsbereich liegen, d.h. Jacobi-Symbol 1 besitzen, müssen die „richtigen" Legendre-Symbole erzwungen werden. Genauer sind für die Berechnung von $y := f_0^{-1}(x)$ die folgenden Schritte durchzuführen:

1. Teste, ob $x \in \mathbf{QR}_n$. Wenn nicht, dann setze $y := -x \pmod n$, anderenfalls $y = x$ {Jetzt gilt sicher $y \in \mathbf{QR}_n$ }
2. Berechne $y_p := y^{(p+1)/4} \pmod p$ und $y_q := y^{(q+1)/4} \pmod q$.
3. Füge die Teilergebnisse mit dem Chinesischen Restalgorithmus zusammen: $y := \text{CRA}(y_p, y_q, p, q)$. { Jetzt gilt sicher $y \in \mathbf{QR}_n$ }
4. Teste, ob $y < \frac{n}{2}$. Wenn nicht, dann setze $y := -y \pmod n$. { Jetzt gilt sicher $y \in \text{D}$ }

Für die Berechnung von $y := f_1^{-1}(x)$ ist nach dem 1. Schritt zusätzlich eine modulare Division durch 4, d.h. eine Multiplikation mit der multiplikativen Inversen von 4 (mod n) (bzw. (mod p) und (mod q)) durchzuführen.

Zum Signieren einer Nachricht m berechnet ein Sender nun $sig := F_m^{-1}(R)$ mittels seines geheimen Signierschlüssels (p, q) und verschickt die Signatur sig mit der Nachricht. Die zufällig gewählte Referenz $R \in \text{D}$ darf hier im Unterschied zu den Hashfunktionen aus Abschnitt

4 nur einmal verwendet werden.[11] Auch sie muß jedoch authentisch bekannt sein, damit der Empfänger anhand des öffentlichen Testschlüssels n die Signatur sig prüfen kann. Dazu bestimmt er $F_m(sig)$ und testet, ob das Ergebnis mit R übereinstimmt.

Im Unterschied zum RSA-Signatursystem wird also nicht die Nachricht selbst von der Signaturfunktion abgebildet, sondern wie bei den DAMGÅRD-Hashfunktionen aus Abschnitt 4 eine zufällig gewählte, authentisch bekannte Referenz R. Die Nachricht m legt die Signierfunktion fest: Jeder Nachricht m aus dem Nachrichtenraum N ist genau eine Funktion F_m zugeordnet. Die Funktionen F_m setzen sich aus Paaren kollisionsresistenter Permutationen zusammen und sind daher selbst kollisionsresistent, d.h. es ist praktisch unmöglich, ein Paar (x, y) zu finden mit $F_m(x) = F_{m'}(y)$, falls $m \neq m', m, m' \in \mathrm{N}$.

Bei dieser Konstruktion treten jedoch zwei Probleme auf:

1. Jedes Zwischenergebnis (nach einer Anwendung von f_0 oder f_1) des Testvorgangs ist eine gültige Signatur der jeweils um ein Bit kürzeren Nachricht (vgl. Abschnitt 4).

2. Die zufällig gewählten Referenzen müssen nachprüfbar authentisiert sein und dürfen nur einmalig verwendet werden.

Die erste Schwierigkeit läßt sich leicht umgehen, indem die Nachricht vor Berechnung der Signatur **präfixfrei** abgebildet wird. Dabei erhält sie am Nachrichtenende einen „Gültigkeitsvermerk“, der dafür sorgt, daß nach der ersten Anwendung von f_0 bzw. f_1 der Zwischenwert keine Signatur einer anderen, präfixfrei abgebildeten Nachricht darstellt (siehe Abschnitt 4).

In Aufgabe 4 ist die Signierfunktion für präfixfrei abzubildende Nachrichten zu implementieren. Der Programmierrahmen stellt einen effizienten Testalgorithmus zur Verfügung.

Das zweite Problem ist nicht ganz so einfach zu lösen. Die Autoren schlagen vor, einen Binärbaum der Tiefe b aus Hilfsreferenzen zufällig zu generieren, die Wurzel r_ε als Teil des öffentlichen Schlüssels bekanntzugeben und an die Blätter des Baumes die für die Signaturen benötigten Referenzen $R_0, \ldots, R_{2^b-1}$ anzuhängen, vgl. Bild 3.

Dieser gesamte Baum ist nun Knoten für Knoten vom Sender zu authentisieren. Dazu werden jeweils die beiden Folge-Hilfsreferenzen r_{j0} und r_{j1} eines Knotens mit der Hilfsreferenz r_j hintereinandergehängt und als zu signierende Nachricht interpretiert, also

$$sig_{\text{präf}(r_{j0}r_{j1})} := F^{-1}_{\text{präf}(r_{j0}r_{j1})}(r_j)$$

berechnet. Diese Signatur wird zur Unterscheidung K-Signatur (K-Sig, K=Knoten) genannt. Verschickt wird dann außer der Nachricht m und deren (Nachrichten-)Signatur N-Sig := $F^{-1}_{\text{präf}(m)}(R_i)$ der von r_ε nach R_i führende Ast des Hilfsreferenzenbaumes, bestehend aus den jeweiligen Hilfsreferenzen r_j (an den Knoten) und deren K-Signaturen (K-Sig) und der Signatur der Referenz R_i (R-Sig, siehe Bild 4).

Der Empfänger kann nun die Authentizität der Referenz R_i überprüfen, indem er die Authentisierungskette ausgehend von R_i bis zum öffentlich bekannten r_ε überprüft. Die Korrektheit der Signaturen jedes dieser Knoten kann er durch Berechnung von $F_{\text{präf}(r_{j0}r_{j1})}(sig_{r_{j0}r_{j1}})$

[11] Mehrfachverwendung einer Referenz R für unterschiedliche Nachrichten ist gleichbedeutend mit dem Erzeugen einer Kollision, was wiederum dazu führt, daß der geheime Schlüssel (die Faktoren p, q mit $p \cdot q = n$) berechnet werden kann.

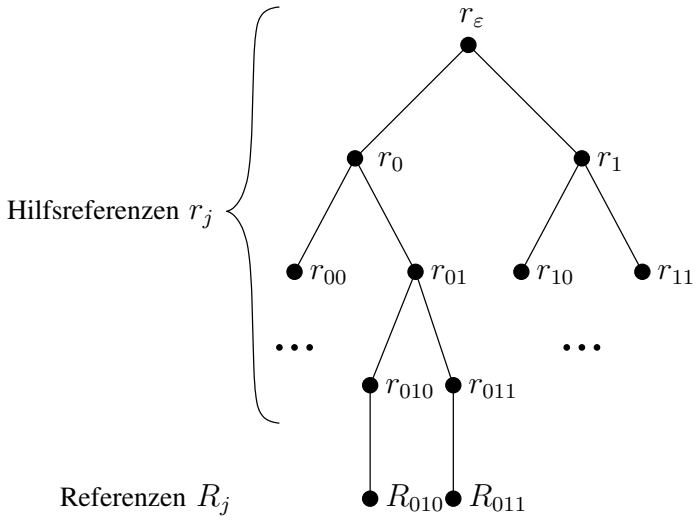

Abbildung 3: Referenzenbaum

bzw. $F_{\text{präf}(R_i)}(sig_{R_i})$ und einen Vergleich mit den mitgeschickten Hilfsreferenzen r_{j0} und r_{j1} bzw. R_i leicht testen.

Eine Berechnung von $F_{\text{präf}(m)}$ erfordert $\log_2(\text{präf}(m))$ modulare Quadrierungen; die Multiplikation mit 4 in f_1, die durchschnittlich für jedes zweite Bit anfällt, kann durch eine Verschiebeoperation um ein Bit vor der Quadrierung ersetzt werden und fällt vergleichsweise nicht ins Gewicht.

Werden die Knoten des Hilfsreferenzenbaumes erst nach Bedarf authentisiert, ist je Signatur einer Nachricht durchschnittlich lediglich eine Signatur einer einzelnen Hilfsreferenz r_j erforderlich. Hinzu kommt die Signatur unter die Referenz R_i.

6 Aufgaben

Die folgenden Aufgaben haben steigenden Schwierigkeitsgrad und bauen aufeinander auf. Bearbeiten Sie sie daher bitte in der angegebenen Reihenfolge.

Für die Programmieraufgaben wird jeweils ein Projekt zur Verfügung gestellt, das entsprechend der Aufgabe benannt ist. Die Klassen, in denen Methoden zu implementieren sind, werden in der jeweiligen Aufgabenstellung genannt. Die benötigten Datenstrukturen und Methoden sind bereits definiert worden. Diese Definitionen sollten nicht geändert werden, um das Testen der Lösungen zu erleichtern. Als Langzahlarithmetik werden die Methoden der Klasse `java.lang.BigInteger` benutzt. Diese Klasse ist in der üblichen JDK-Dokumentation ausführlich beschrieben. Außerdem können noch die Algorithmen CRA, LEGENDRE und SQRTMOD benutzt werden, die aus dem Versuch "Zahlentheoretische Algorithmen" bekannt sind. Dokumentation dazu ist Teil der Versuchssoftware (Datei `doc/index.html`)

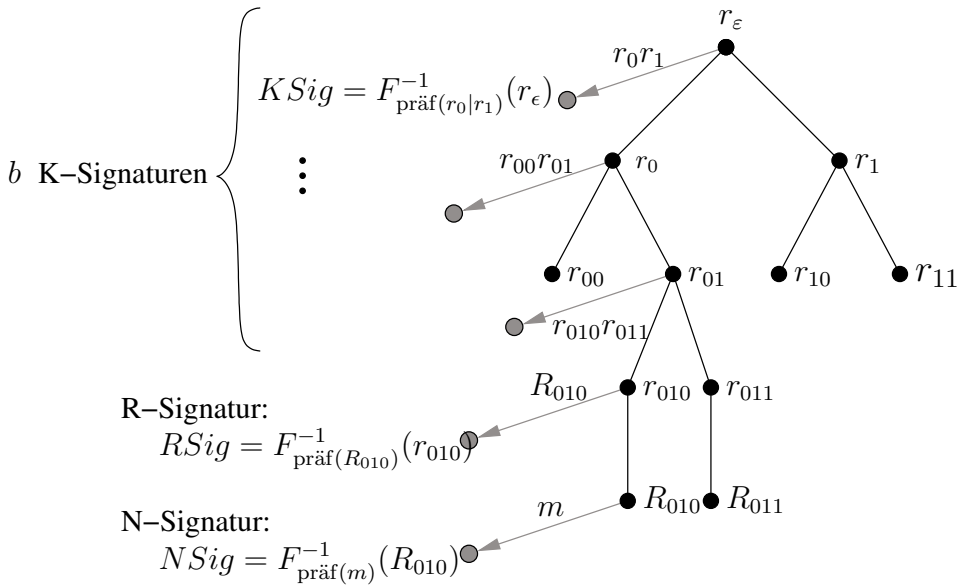

Abbildung 4: Signatur unter die Nachricht m bezüglich der Referenz R_{01}

Aufgaben

1. Implementieren Sie die für das Signieren von Nachrichten mit RSA und das Testen einer Signatur erforderlichen Methoden in der Klasse `RSA.java`. Sie können vereinfachend annehmen, daß die Nachrichten als Langzahlen (`BigInteger`) vorliegen und kleiner sind als der Modulus n des öffentlichen RSA-Schlüssels (Betriebsarten sind also nicht erforderlich). Zum Testen der Methoden starten Sie das Projekt `Aufgabe1`.

2. Vollziehen Sie den in Abschnitt 3.2 beschriebenen aktiven Angriff von DENNING nach. Schreiben Sie ein kurzes Demonstrationsprogramm (Projekt `Aufgabe2`, Klasse `Aufgabe2`), das den Angriff für kurze Nachrichten simuliert. Dabei sollen die von Ihrem Programm vorgenommenen Aktivitäten durch Bildschirmausgaben dokumentiert werden. Vereinfachend können Sie sowohl die Aktivitäten des Angreifers als auch die des Opfers durch dasselbe Programm durchführen lassen; eine Kommunikation von Angreifer und Opfer ist nicht zu programmieren.

3. Vervollständigen Sie die Methode `hash` in der Klasse `Damgard`. Diese Methode bildet eine Nachricht m zunächst präfixfrei ab und „komprimiert“ sie anschließend mit einer DAMGÅRD-Hashfunktion. Dabei können Sie vereinfachend $s := 2, a_0 := 1$ und $a_1 := 4$ fest wählen.

   In der Klasse `Damgard` steht bereits die Methode `praefixfreiMessage(BigInteger)` für die präfixfreie Abbildung einer Nachricht bereit. (Hinweis: Die Nachrichtenbits müssen **beginnend vom niederwertigsten Bit** bearbeitet werden, d.h. das niederwertigste Bit ist der „Anfang“ der Nachricht.)

4. Vervollständigen sie die Methode `sign` in der Klasse `GMR`. Diese Methode bildet eine

Nachricht m zunächst präfixfrei ab und signiert sie dann mit GMR. (siehe Abschnitt 5). Eine Methode für die präfixfreie Abbildung einer Nachricht steht bereit. Außerdem steht eine Methode bereit, die ermittelt, ob eine Zahl x quadratischer Rest bezüglich einer Zahl $n = p \cdot q$ ist. Die Klasse `GMR` enthält bereits eine Implementierung der GMR-Testfunktion. Diese wird beim Test der Implementierung im Projekt `Aufgabe4` benutzt.

Hinweise:

- Genauso wie bei der Implementierung der DAMGÅRD-Hashfunktion ist auch hier das niederwertigste Bit der „Anfang“ der Nachricht. Dementsprechend muß die Signierfunktion die Nachricht von „hinten“ nach „vorn“, also vom höchstwertigsten zum niederwertigsten Bit verarbeiten.
- Achten Sie besonders auf eine **effiziente Implementierung**: Das Signieren wird mit Hilfe einer Konkatenation der Umkehrpermutationen $y := f_0^{-1}(x)$ bzw. $y := f_1^{-1}(x)$ durchgeführt. Überlegen sie zunächst, welche Schritte der Berechnung von $y := f_0^{-1}(x)$ bzw. $y := f_1^{-1}(x)$ bei jeder Ausführung, und welche nur vor der ersten bzw. nach der letzten Ausführung dieser Umkehrpermutationen durchgeführt werden müssen.)

Literatur

[Blum_82] Manuel Blum: Coin flipping by telephone - a protocol for solving impossible problems. In: Proc. IEEE Spring CompCon, San Francisco (1982), 133–137.

[Damg_88] Ivan Bjerre Damgård: Collision Free Hash Functions and Public Key Signature Schemes. In: Eurocrypt 1987, Band 304 von *Lecture Notes in Computer Science*, Springer, Berlin (1988), 203–216.

[DaPr_80] Donald W. Davies, Wyn L. Price: The application of digital signatures based in public key cryptosystems. In: Proc. of 5th Int. CompCom (1980), 525–530.

[DaPr_85] Donald W. Davies, Wyn L. Price: Digital signatures - an update. In: Proc. Int. Conf. on CompCom (1985), 843–847.

[DaPr_89] Donald W. Davies, Wyn L. Price: Security for computer networks. 2. Auflage, John Wiley & Sons, Chichester (1989).

[Denn_82] Dorothy Denning: Cryptography and Data Security. Addison-Wesley Publishing Company, Reading (1982), reprinted with corrections, January 1983.

[Denn_84] Dorothy E. Denning: Digital Signatures with RSA and other Public-Key-Cryptosystems. Communications of the ACM, Band 27(4) (1984), 388–392.

[DiHe_79] Whitfield Diffie, Martin E. Hellman: Privacy and Authentication: An Introduction to Cryptography. Proc. of the IEEE (1979).

[Fox_90] Dirk Fox: Implementierung eines sicheren digitalen Signatursystems (5 1990), Studienarbeit, Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe.

[Fox_91] Dirk Fox: Effiziente Softwareimplementierung asymmetrischer Kryptosysteme und der zugrundeliegenden modularen Langzahlarithmetik. Diplomarbeit, Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe (5 1991).

[FoPf_91] Dirk Fox, Birgit Pfitzmann: Effiziente Softwareimplementierung des GMR-Signatursystems. In: Proc. of VIS '91, Verläßliche Informationssysteme, Band 271 von *Informatik Fachberichte*, Springer (1991), 329–345.

[Gira_87] Marc Girault: Hash-Funktions using Modulo-n Operations. Eurocrypt 87, Amsterdam (April 1987), extended abstract.

[GoMR_84] Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: A 'Paradoxial' Solution to the Signature Problem. In: 25th Symposium on FOCS 1984, IEEE Computer Society (1984), 441–448.

[GoMR_88] Shafi Goldwasser, Silvio Micali, Ronald R. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. SIAM Jurnal on Computing, Band 17(2) (1988), 281–308.

[Jung_87] Achim Jung: Implementing the RSA Cryptosystem. Computers & Security, Band 6(4) (1987), 342–350.

[Knut_81] Donald Ervin Knuth: Seminumerical Algorithms, Band 2 von *The Art of Computer Programming*. 2. Auflage, Addison Wesley, Reading (1981).

[Pfit_90] Andreas Pfitzmann: Diensteintegrierende Kommunikationsnetze mit teilnehmerüberprüfbarem Datenschutz, Band 234 von *Informatik-Fachberichte*. Springer-Verlag, Heidelberg (1990).

[Pfit_98] Andreas Pfitzmann: Sicherheit in Rechnernetzen: Mehrseitige Sicherheit in verteilten und durch verteilte Systeme (Oktober 1998), TU Dresden; Fakultät Informatik, Script.

[RSA_78] R. L. Rivest, A. Shamir, L. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, Band 21(2) (1978), 120–126, reprinted: 26/1 (1983) 96-99.

[Salo_90] Arto Salomaa: Public-Key Cryptography, Band 23 von *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg (1990).