Ausarbeitungen

Proseminar Kryptographie

WS 2002-2003

Leiter: Prof. Dr. K. Madlener
Betreuer: Robert Eschbach
Studenten: Timo Neumann, Alexander Petry,
Dominik Schultes, Georg Westenberger,
Christian Zieman

AG GRUNDLAGEN DER INFORMATIK, FB INFORMATIK, UNIVERSITÄT KAISERSLAUTERN,
67653 KAISERSLAUTERN
E-mail address: eschbach@informatik.uni-kl.de

ZUSAMMENFASSUNG. Dieses Dokument enthält die Ausarbeitungen der Teilnehmer des Proseminars 'Kryptographie', welches im WS 2002-2003 von der AG 'Grundlagen der Informatik' unter der Leitung von Prof. Dr. K. Madlener durchgeführt wird.
Stand: 6. Mai 2003

Inhaltsverzeichnis

Kapitel 1. Grundlagen und Protokolle (Christian Ziemann) 5
1. Einleitung 5
2. Geschichtlicher Hintergrund 5
3. Kryptologische Grundlagen 5
4. Grundlegende Protokolle 14
5. Fazit 20
6. Quellen 20

Kapitel 2. Zero-Knowledge-Verfahren (Alexander Petry) 21
1. Einleitung 21
2. Identifikations-Verfahren 23
3. Zero-Knowledge 27
4. Nicht-interaktive Zero-Knowledge Verfahren 34
5. Fazit 36

Kapitel 3. Multiparty Computations und Anonymität (Georg Westenberger) 37
1. Verwendete Notation 37
2. Einleitung 37
3. Multiparty Computations 37
4. Anonymität 41
5. Fazit 46

Kapitel 4. Interaktive Beweise (Timo Neumann) 49
1. Das interaktive Beweissystem 49
2. Die Klasse IP 52
3. Interaktive Beweise und Zero-Knowledge 59
4. Offene Probleme 62
5. Fazit 62

Kapitel 5. Primzahlen (Dominik Schultes) 63
1. Einführung 63
2. Zufallszahlen 64
3. Primzahl-Tests 65
4. Fazit 75

Literaturverzeichnis 77

KAPITEL 1

Grundlagen und Protokolle (Christian Ziemann)

1. Einleitung

In der heutigen Zeit ist die Nachfrage an Sicherheit, im Bezug auf Internet, Netzwerken usw., so hoch wie nie zuvor. Um diese Sicherheit zu gewährleisten, müssen gewisse grundlegende Hilfsmittel benutzt und Regeln beachtet werden. Diese Hilfsmittel, wie z.B. Hashfunktionen, Einwegfunktionen... erläutere ich in dieser Ausarbeitung. Danach werde ich allgemeine Verfahren zur Kommunikation, wie z.B. Challenge-and-Response vorstellen.

2. Geschichtlicher Hintergrund

Schon vor etwa 2500 Jahren versuchten die Spartaner Texte auf eher primitiver Weise zu verschlüsseln. Damals geschah dies mit Hilfe einer *Skytale*, einem langen runden Stab, auf dem Pergament Zeile für Zeile gewickelt wurde. Dieses Pergament wurde dann, um den Geheimtext zu erstellen, auf dem Holzstab beschriftet. Bei diesem Verfahren fungierte der Holzstab als Schlüssel. Sowohl der Sender als auch der Empfänger brauchten einen identischen Stab, um die Nachricht später auch wieder entschlüsseln zu können.
Später (ca. 100-44 v. Chr.) benutzte Julius Caesar die *Verschiebechiffre*, um seine Nachrichten zu versenden. Dabei tauschte er den zu übermittelnden Buchstaben zum Beispiel durch seinen dritten Nachfolger im Alphabet aus:
A B C D E F Klartext
X Y Z A B C Geheimtext
Besser ist das *Vignere-Verfahren* (1523-1596), bei dem ein anderes Alphabet für den Geheimtext benutzt wird. Das Problem hierbei ist die Übermittlung des Schlüssels.

3. Kryptologische Grundlagen

3.1. Symmetrische Verschlüsselung. Eine Möglichkeit zur Geheimhaltung von Nachrichten ist die *symmetrische Verschlüsselung*. Hierbei besitzen Sender und Empfänger einen gemeinsamen Schlüssel, um einen Klartext in Geheimtext beziehungsweise den Geheimtext wieder in Klartext umzuwandeln.
Formal kann man die symmetrische Verschlüsselung als ein 5-Tupel darstellen: $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, wobei
$\mathcal{P}$ die Menge der möglichen Klartexte,
$\mathcal{C}$ die Menge der möglichen Geheimtexte und
$\mathcal{K}$ die Menge der möglichen Schlüssel ist.

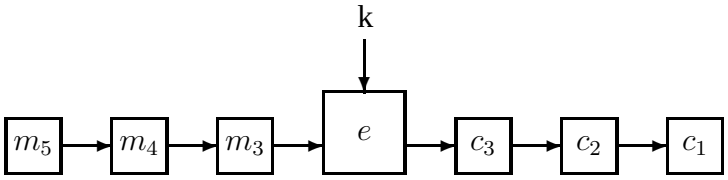
Außerdem gilt: Für jedes K$\in \mathcal{K}$ existiert eine Verschlüsselungsfunktion e:$\mathcal{P} \times \mathcal{K} \to \mathcal{C}$, $e_K \in \mathcal{E}$ und eine dazugehörige Entschlüsselungsfunktion d:$\mathcal{C}\times\mathcal{K} \to \mathcal{P}$,$d_K \in \mathcal{D}$, so dass gilt: $d_K(e_K(x)) = x$ für einen Klartext x.
Hierzu ein Beispiel:
Sender: $c = e_K(m)$ $\longrightarrow$ Empfänger: $d_K(c) = m$,
wobei m$\in \mathcal{P}$, c$\in \mathcal{C}$, K$\in \mathcal{K}$, $d_K \in \mathcal{D}$ und $e_K \in \mathcal{E}$.
In diesem Beispiel wird davon ausgegangen, dass der geheime Schlüssel K vorher auf sicherem Wege übertragen wurde. Nun entsteht der Geheimtext c durch Verschlüsselung des Klartextes m mit Hilfe der Verschlüsselungsfunktion e_K. Der Empfänger wendet dann die Entschlüsselungsfunktion d_K auf den Geheimtext c an und erhält so wieder m.
Bei dieser Variante muss davon ausgegangen werden, dass sowohl Ver- als auch Entschlüsselungsfunktion (e und d) bekannt sind (Kerckhoffsches Prinzip). Die Sicherheit beruht auf der Geheimhaltung des Schlüssels K.

3.2. Angriffsarten. Es gibt vier Arten von Angriffen, die eine Verschlüsselungsfunktion f überstehen muss, um als sicher zu gelten. Dabei ist dem Angreifer beim Ciphertext only attack am wenigsten, und beim Chosen ciphertext attack am meisten bekannt. Die Schwierigkeit (bei der unten aufgeführten Reihenfolge) bei diesen Angriffen die erwünschten Informationen zu erhalten sinkt, da der Angreifer immer mehr und wichtigere Informationen zur Verfügung hat:

- *Ciphertext only attack:* Dem Angreifer sind eine endliche Anzahl von Geheimtexten $c \in \mathcal{C}$ bekannt, und möchte daraus den verwendeten Schlüssel $K \in \mathcal{K}$ und die dazugehörigen Klartexte $p \in \mathcal{P}$ berechnen.
- *Known-plaintext attack:* Dem Angreifer ist eine begrenzte Anzahl von Geheimtexten $c \in \mathcal{C}$ sowie den dazugehörigen Klartexten $p \in \mathcal{P}$ bekannt, und möchte daraus den dazugehörigen Schlüssel $K \in \mathcal{K}$, bzw. den Klartext zu einem noch nicht entschlüsselten Geheimtext c berechnen.
- *Chosen plaintext attack:* Der Angreifer hat sich zu der mit dem Schlüssel k$\in \mathcal{K}$ parametrisierten Verschlüsselungsfunktion e Zugriff verschafft. Er kann den Schlüssel k zwar nicht auslesen, aber bestimmte von ihm ausgewählte Klartexte mit Hilfe von e_k verschlüsseln. Mit Hilfe dieser Informationen möchte er andere Geheimtexte $c \in \mathcal{C}$ entschlüsseln, bzw. den Schlüssel k berechnen.
- *Chosen ciphertext attack:* Der Angreifer hat sich Zugang zu der mit dem Schlüssel $k \in \mathcal{K}$ parametrisierten Entschlüsselungsfunktion d_k verschafft. Er kann den Schlüssel k zwar nicht auslesen, kann aber Geheimtexte entschlüsseln. Mit dieser Information versucht er k zu berechnen.

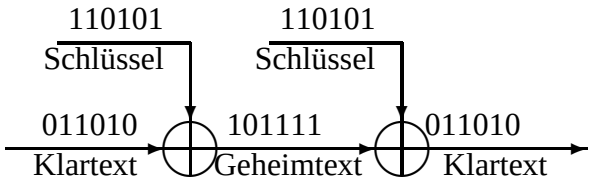
3.3. Blockchiffren und Stromchiffren. Algorithmen zur symmetrischen Verschlüsselung von Daten werden in Block- und Stromchiffren unterteilt.

3.3.1. *Blockchiffren.* Bei einer Blockchiffre wird eine Nachricht in gleich lange Blöcke zerteilt (z.B. 64 Bit) und jeder einzelne Block wird dann mit der Funktion e und dem Schlüssel k verschlüsselt: $c_i = e_k(m_i), i = 1, 2, 3, ...$



Beispiele hierfür sind die Blockchiffren DES (Data Encryption Standard), der aufgrund seines nur 52 Bit langen Schlüssels eher veraltet ist, und der IDEA (International Data Encryption Algorithm) dessen Schlüssel aus 128 Bit besteht. Das modernste Verfahren ist allerdings der AES (Advanced Encryption Standard), welcher erst am 2.10.2000 veröffentlicht wurde. Der Nachfolger des DES benutzt eine Blocklänge von 128, 192 und 256 Bit und eine Schlüssellänge von 128, 192 und 256 Bit.

3.3.2. *Stromchiffren.* Bei einer Stromchiffre wird eine Nachricht zeichenweise verschlüsselt. Der bei diesem Verfahren benutzte Schlüssel muss die gleiche Länge wie der Klartext haben. Jedes Zeichen des Klartextes wird dann mit einem Zeichen des Schlüssels zu einem chiffrierten Zeichen verknüpft. Daraus ergibt sich dann ein chiffrierter Text von gleicher Länge wie der Klartext. Das One-Time-Pad, der Prototyp aller Stromchiffren, benutzt eine Bitfolge als Schlüssel, demzufolge muss auch der Klartext als Bitfolge vorliegen. Bei diesem Verfahren werden Klartext- und Schlüsselbits modulo 2 addiert:
Bsp.:



Der Nachteil der Stromchiffren ist, dass der Schlüssel genauso lang sein muss wie der Klartext. Außerdem ist es ratsam einen Schlüssel nur einmal zu verwenden, denn nur dann ist das Verfahren sicher. Andernfalls könnte der Schlüssel mit einem Known-Plaintext-Angriff errechnet werden.
Hierzu ein Beipiel:
Es wird die Zeichenfolge 101 mittels des One-Time-Pad verschlüsselt, der Schlüssel ist hierbei 010:

$$\begin{array}{ccc} \text{Klartext} & \rightarrow & \text{Geheimtext} \\ 101 & \rightarrow & 111 \end{array}$$

Bei dem Known-Plaintext-Angriff ist dem Angreifer nun der Klartext und der dazugehörige Geheimtext bekannt. Jetzt addiert er einfach den Klartext mit dem Geheimtext, und schon erhält er den verwendeten Schlüssel:

$$\begin{array}{ccccc} \text{Klartext} & + & \text{Geheimtext} & \rightarrow & \text{Schlüssel} \\ 101 & + & 111 & \rightarrow & 010 \end{array}$$

Anwendungen: Besonders eignen sich Stromchiffren bei Online-Chiffrierung von Nachrichtenkanälen sowie bei der Verschlüsselung ganzer Festplatten.

3.4. Asymmetrische Verschlüsselung. Bei der *asymmetrischen Verschlüsselung* (oder auch *Public-Key-Verschlüsselung*) handelt es sich um ein Verschlüsselungsverfahren, bei dem der Sender nicht den geheimen Schlüssel des Empfängers benötigt. 1976 erfanden W. Diffie und M. Hellman das asymmetrische Verschlüsselungsverfahren, bei dem jeder Teilnehmer T eines Systems einen privaten (oder auch geheimen) Schlüssel $d = d_T$ und einen öffentlichen Schlüssel $e = e_T$ zugeordnet bekommt. Hierbei wird in der Theorie davon ausgegangen, dass die verwendeten Schlüssel der Teilnehmer verschieden sind, d.h. kein Teilnehmer des Systems besitzt oder benutzt den selben geheimen Schlüssel eines Anderen. Der geheime Schlüssel ist, wie der Name schon sagt, geheim zu halten. Der öffentliche Schlüssel kann den Teilnehmern des Systems öffentlich zugänglich gemacht werden.
Die Funktion f erstellt nun mit dem öffentlichen Schlüssel e aus dem Klartext m den Geheimtext c:

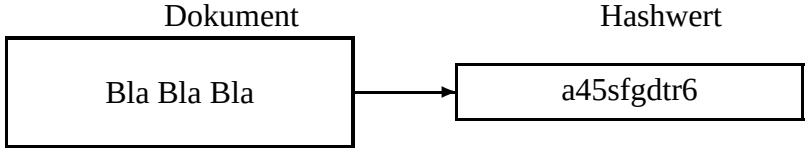
$$c = f_e(m)$$

Jetzt kann nur derjenige die Nachricht entschlüsseln, der den dazugehörigen privaten Schlüssel d besitzt, da nur dieser Schlüssel den öffentlichen Schlüssel e des Teilnehmers invertiert und den Klartext m wieder erzeugt. Die Funktion f ordnet jetzt unter dem privaten Schlüssel d, einem Geheimtext c den dazugehörigen Klartext m zu:

$$m' = f_d(c) \text{ Also gilt: } m' = f_d(c) = f_d(f_e(m)) = m$$

3.5. Einwegfunktionen. Eine *Einwegfunktion* ist eine Funktion, die leicht auszuführen, aber schwer zu invertieren ist. Das bedeutet, der Algorithmus, der die Einwegfunktion invertieren soll, liegt in der Komplexitätsklasse $\mathcal{NP}$, d.h. unter der Annahme, dass: $\mathcal{N} \neq \mathcal{NP}$, kann eine nichtdeterministische Touringmaschine den Algorithmus nicht in polynomialer Zeit berechnen. Ein anschauliches Beispiel hierfür ist ein Telefonbuch: Es ist leicht einem Namen eine Telefonnummer zuzuordnen, allerdings fast unmöglich einer Nummer den Namen zuzuordnen. Eine Einwegfunktion ist also eine Abbildung f von einer Menge X in eine Menge Y, so dass f(x) für jedes Element von X leicht zu berechnen ist, während es für jedes y aus Y extrem schwer ist ein Urbild x (d.h. ein x mit f(x)=y) zu finden. Ist eine Einwegfunktion $f : X \to X$ bijektiv, so spricht man von einer *Einwegpermutation*. Eine Einwegfunktion, für die es in angemessener Zeit unmöglich ist, zwei verschiedene Werte x und x', mit $x \neq x'$ in der Urbildmenge X zu finden, mit $f(x) \neq f(x')$, heißt *kollisionsfrei*.
Noch ist es allerdings unklar, ob es Einwegfunktionen überhaupt gibt. Sobald bewiesen wird, dass $\mathcal{N} \neq \mathcal{NP}$ gilt, ist auch bewiesen, dass Einwegfunktionen existieren.

3.6. Kryptographische Hashfunktionen. Eine *Einweg-Hashfunktion* (oder auch *kryptographische Hashfunktion*) ist eine kollisionsfreie Einwegfunktion, die Nachrichten beliebiger Länge auf einen Hashwert fester Länge komprimiert (typischer Wert 128 Bit). Man spricht dabei auch von einem 'Fingerabdruck', da ein Fingerabdruck eine Person eindeutig identifiziert, es aber unmöglich ist, von einem Fingerabdruck auf die dazugehörige Person zu schließen.
In der Praxis ist es außerordentlich schwierig gute Hashfunktionen zu finden. Prüfsummen zum Beispiel sind gänzlich ungeeignet, da sie zu manipulierbar sind. Deshalb benutzt man heute eher Hashfunktionen wie: MD4, MD5, RIPEMD und SHA.

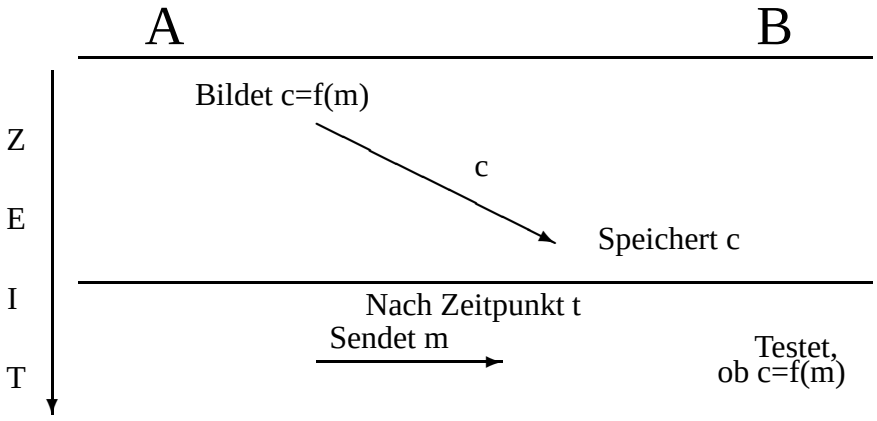


3.7. Trapdoor-Einwegfunktionen. Der Nachteil von Einwegfunktionen ist der, dass ihr Einsatzbereich sich auf Berechnungen beschränkt, bei denen alle Beteiligte die Berechnungen durchführen dürfen. Eine *Trapdoor-Einwegfunktion* ist eine Einwegfunktion, die nur mit Hilfe einer Geheiminformation invertierbar ist.
Ein Beispiel hierfür ist:

$$\begin{array}{rccl} f: & \mathbb{Z}/n\mathbb{Z} & \to & \mathbb{Z}/n\mathbb{Z} \\ & x & \mapsto & x^2 \bmod n \end{array}$$

Für n=pq ist dies eine Trapdooreinwegfunktion, denn ohne Kenntnis der Faktorisierung von n ist es praktisch unmöglich diese Funktion zu invertieren. Eine ausführliche Betrachtung dieser Problematik folgt beim RSA-Verfahren.

3.8. Commitment und Bit-Commitment. Das *Commitment* lässt sich am Besten anhand eines Beispiels erklären: 3 verschiedene Firmen wollen für eine Ausschreibung ihr Angebot abgeben. Dabei sollen die jeweiligen Angebote nicht von anderen Firmen gelesen werden und bis zu einem gewissen Zeitpunkt soll auch die Firma, die das Angebot gemacht hat, ihr Angebot nicht verändern können. Dieses Problem wird mit Hilfe kollisionsfreier Einwegfunktionen gelößt. Dabei wird die Nachricht m der Einwegfunktion f unterworfen und dieser Wert $c = f(m)$ an den Empfänger gesendet. Die Einwegeigenschaft garantiert, dass kein m' mit $c = f(m')$ existiert, d.h. im obigen Beispiel kann der Empfänger sein Gebot nicht ändern. Wenn der Sender nun dem Empfänger das Urbild m mitteilt, hat der Sender das Commitment geöffnet, vorher ist es für den Empfänger unmöglich m zu berechnen.

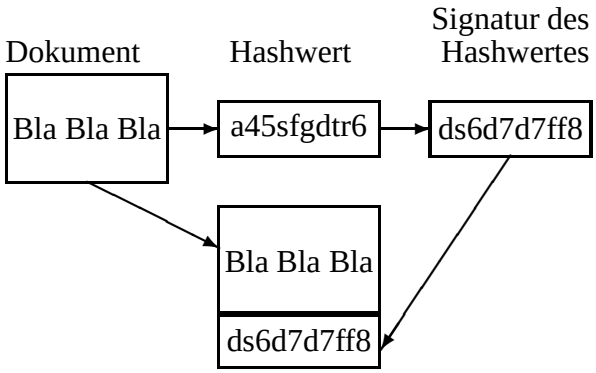


Ein *Bit-Commitment* besteht nun, wenn m nur ein Bit lang ist. Da keine Einwegfunktion auf $\{0,1\}$ existiert, wird dieses Problem nun wie folgt gelöst: Man benötigt eine hinreichend große Zufallszahl r, das Bit b und eine Einwegfunktion $f : \{0,1\} \times X \to Y$, die Kollisionsfreiheit auf das erste Argument garantiert, d.h. für alle Zufallszahlen r, s muss gelten:
$f(0,r) \neq f(1,s)$
Ein Beispiel hierfür ist folgende Funktion: $f(b,r) = y^b r^2 mod\ n$ wobei $n = pq$ das Produkt zweier großer Primzahlen und y ein fester quadratischer Nichtrest $modulo\ n$ mit Jakobisymbol $+1$ ist.

3.9. Digitale Signatur. Das Ziel der digitalen Signatur (oder elektronische Unterschrift) ist es, die wesentlichen Eigenschaften der handschriftlichen Unterschrift in elektronischer Form zu realisieren:

- *Echtheitseigenschaft:*
  Stellt sicher, dass das Dokument wirklich vom Unterschreibenden stammt. Es wird gefordert, dass ein enger Zusammenhang zwischen dem Dokument und der Unterschrift besteht, wie etwa dass sie auf dem selben Blatt stehen.
- *Identitätseigenschaft:*
  Jede digitale Signatur ist persönlich, d.h. sie kann nur von einem einzigen Menschen ausgestellt werden.
- *Abschlusseigenschaft:*
  Diese signalisiert die Vollendung der Erklärung. Dies wird dadurch ausgedrückt, dass die Unterschrift am Ende des Dokuments zu finden ist.
- *Warneigenschaft:*
  Diese soll den Unterzeichnenden vor einer Übereilung bewahren.
- *Verifikationseigenschaft:*
  Jeder Empfänger einer unterschriebenen Erklärung kann die Unterschrift verifizieren, etwa durch einen Unterschriftenvergleich.

Diese Eigenschaften (bis auf die Warneigenschaft) lassen sich nun mit Hilfe eines Signaturschemas mittels kryptographischer Mechanismen übertragen:
Jedem Teilnehmer T eines Systems wird eine geheime Signaturfunktion s_T und eine öffentliche Verifikationsfunktion v_T zugeteilt. In der Theorie wird davon ausgegangen, dass die Verifikationsfunktionen und die Signaturfunktionen der einzelnen Teilnehmer verschieden sind, d.h. es existieren keine identischen Verifikations- oder Signaturfunktionen zwischen zwei unterschiedlichen Teilnehmern. Aus v_T lässt sich dabei unmöglich s_T berechnen. Unterschreibt ein Teilnehmer T nun eine Nachricht m, so muss er auf m seine Signaturfunktion s_T anwenden: $sig = s_T(m)$. Der Empfänger erhält dann m wie auch sig, und ist nun in der Lage, die Echtheit der Signatur mit Hilfe der Verifikationsfunktion v_T zu überprüfen. Es gibt eine einfache Klasse von Signaturschematas (zu denen auch das RSA-Verfahren gehört), bei denen die Verifikationsfunktion die Umkehrung der Signaturfunktion ist, also: $v_T(sig) = m$.
Ein (zentrales) Problem der Signaturschemata ist, dass sich lange Dokumente nicht in angemessener Zeit signieren lassen und die Signatur stets so lang ist wie das unterschriebene Dokument selbst. Um eine kurze Signatur fester Länge zu erhalten, wird in der Praxis ein Dokument erst mithilfe einer öffentlichen Hashfunktion h auf einen Wert $h(m)$ fester Länge komprimiert. Danach wird die Signaturfunktion auf diesen Wert angewendet: $sig = s_T(h(m))$



Bei der Verifikation bildet man ebenfalls zunächst $h(m)$, und überzeugt sich dann, dass sig die zu $h(m)$ gehörende Signatur ist.

3.10. Der RSA-Algorithmus. Der *RSA-Algorithmus* wurde 1978 von R. Rivest, A. Shamir und L. Adleman erfunden, als sie eigentlich beweisen wollten, dass Public-Key-Kryptographie unmöglich sei.
Der RSA-Algorithmus begründet sich auf folgender mathematischer Tatsache: Sei $n = p \cdot q$ das Produkt zweier verschiedener Primzahlen p und q, dann gilt: Für alle $m \leq n;\ m, n, k \in \mathbb{N}$:

$$m^{k(p-1)(q-1)+1}\ mod\ n = m$$

Für den RSA-Algorithmus müssen nun ein e und ein d mit:

$$e \cdot d = k(p-1)(q-1) + 1$$

gefunden werden. Es gilt:

$$(m^e)^d \; mod \, n = m \text{ und}$$

$$(m^d)^e \; mod \, n = m$$

In der Praxis geht man nun wie folgt vor: Jedem Teilnehmer eines Systems werden zwei verschieden große Primzahlen p und q zugeordnet, wobei:

$$n = p \cdot q \text{ und } \varphi(n) = (p-1)(q-1),$$

dann wählt man $e \in \mathbb{N}$ das teilerfremd zu φ(n) ist und berechnet d mit:

$$e \cdot d \; mod \, n \; \varphi(n) = 1, \text{ also:}$$
$$e \cdot d = 1 + k(p-1)(q-1)$$

für eine natürliche Zahl k. Der private Schlüssel dabei ist d und der öffentliche Schlüssel bildet e zusammen mit n. Dabei sind p, q und $\varphi(n)$ ebenfalls geheimzuhalten, können aber genausogut vernichtet werden, da sie nicht mehr benötigt werden.

3.10.1. *Der RSA als asymmetrisches Verschlüsselungsverfahren.* Wenn der RSA-Algorithmus nun als asymmetrisches Verschlüsselungsverfahren benutzt wird, so wird das Potenzieren mit e als Verschlüsseln, und das Potenzieren mit d als Entschlüsseln angesehen. Nach dem Satz von Euler (siehe [**Sti95**]) wird hierbei korrekt entschlüsselt;

$$\text{Verschlüsseln:=} f_e(m) := m^e \; mod \, n$$
$$\text{Entschlüsseln:=} \; f_d(c) := c^d \; mod \, n$$
$$\text{Korrektheit: } f_d(f_e(m)) = (m^e)^d \; mod \, n = m.$$

Es handelt sich hierbei um einen Public-Key, da wenn man n kennt, und nicht die Faktoren p, q oder $\varphi(n)$, kann man aus e nicht d berechnen.

3.10.2. *Der RSA-Algorithmus als Trapdoor-Einwegfunktion.* Der RSA-Algorithmus kann auch als Trapdoor-Einwegfunktion angesehen werden. So ist das Potenzieren mit $e \; modulo \; n$ die Einwegfunktion. Als Trapdoorinformation sind d, n und $\varphi \; (n)$ gleichermaßen geeignet.

3.10.3. *Der RSA-Algorithmus als Signaturverfahren.* Wenn man den RSA-Algorithmus nun als Signaturverfahren benutzen möchte, so gilt als Signaturfunktion: $sig = s_T(m) := m^d \; mod \, n$. Hierbei wird einfach auf die Nachricht m der geheime Schlüssel d des Teilnehmers T angewandt. Als Verifikationsfunktion wird $v_T(sig) := sig^e \; mod \, n = m$ benutzt. Hierbei wird der öffentliche Schlüssel e von T auf sig angewandt, und dann mit m verglichen.

3.10.4. *Die Sicherheit des RSA-Algorithmus.* Die Sicherheit des RSA-Algotithmus beruht auf der Schwierigkeit, die Umkehrfunktionen der beiden Hauptbestandteile des RSA-Algotithmus zu bilden:

Faktorisierung von n:

Um die Umkehrfunktion von z.B. $f(x) = x^2 \; mod \, n$, wobei $n = p \cdot q$ und p unq q Primzahlen sind zu berechnen, muss man n zunächst in die Primfaktoren zerlegen. Die Algorithmen (z.B. Number-Field-Sieve, Quadratic-Sieve...), die dieses leisten, liegen in der Klasse $\mathcal{NP}$, d.h. unter

der Annahme, dass: $\mathcal{N} \neq \mathcal{NP}$, kann eine nichtdeterministische Touringmaschine den Algorithmus nicht in polynomialer Zeit berechnen.

Problem des diskreten Logarithmus:
Eine andere Möglichkeit, den RSA-Algorithmus zu knacken, ist die Berechnung der Umkehrfunktion der Exponentialfunktion $f(x) = g^k mod\, p$, wobei p eine Primzahl, und g eine natürliche Zahl mit $g \leq p-1$ ist. Die Umkehrfunktion (diskreter Logarithmus genannt), wird nun ebenfalls mit Algorithmen (z.B. Baby-step-Giant-step,...), die in der Klasse $\mathcal{NP}$ liegen, berechnet. Also kann auch hier kein Angriff erfolgen, der in akzeptabler Zeit erfolgreich wäre.

3.11. Attacken auf den RSA-Algorithmus. Der RSA-Algorithmus gilt immer noch als ungebrochen. Entweder die Attacken zielen auf die Implementierung des RSA-Algorithmus, oder, wie unten aufgeführt, auf Spezialfälle.

- *Die Homomorphieeigenschaft vom RSA-Algorithmus:*
  Mit dem öffentlichen Schlüssel (e, n) eines RSA-Systems gilt:
  $m_1^e m_2^e \; mod\; n = (m_1 m_2)^e \; mod\; n$
  $\Longrightarrow$ Man kann $m_1 m_2$ verschlüsseln, ohne m_1 und m_2 zu kennen.
  Dieser Angriff trifft auch auf das RSA-Signaturverfahren zu, man erkennt ihn daran, dass die Nachricht m_1 und m_2 Sinn ergeben, $m_1 m_2$ aber nicht. Diesen Angriff kann man vermeiden (oder erkennen), indem man Redundanz einführt.
- *Generieren eines gültigen Nachrichten Signatur-Paares:*
  Man kann leicht aus einer Signatur eine Nachricht erzeugen, die diese Signatur besitzt:
  Man bildet dazu: $m := sig^e \; mod\; n$
  Die Überprüfung ergibt dann: $m^d = sig^{ed} = sig \; mod\; n$
  wird also als korrekt angesehen. Dieser Angriff lässt sich mit Hilfe von Redundanz vermeiden.
- *Verwendung kleiner Exponenten:*
  Aus Effizienzgründen wird oft $e = 3$ als Exponent des öffentlichen Schlüssels verwendet. Wenn dies der Fall ist, und drei Teilnehmer eines Systems die gleiche Nachricht verschlüsseln, kann man mit Hilfe des chinesischen Restsatzes die eindeutige Lösung bestimmen:
  $y_1 = m^3 \; mod\; n_1$
  $y_2 = m^3 \; mod\; n_2$
  $y_3 = m^3 \; mod\; n_3$
  $\Longrightarrow y = m^3 \; mod\; n_1 n_2 n_3$
  Da $m < n_1, m < n_2, m < n_3$
  $\Longrightarrow m^3 < n_1 n_2 n_3$
  $y = m^3 \; mod\; n_1 n_2 n_3 = m^3$
  $\longrightarrow m = \sqrt[3]{y}$

Es gibt noch eine Vielzahl von anderen Attacken, allerdings möchte ich an dieser Stelle nicht auf diese eingehen, da wie auch in den Beispielen zuvor, diese Attacken nur auf Spezialfälle angewendet werden können.

4. Grundlegende Protokolle

Wenn mehrere Personen gemeinsam ein Ziel verfolgen, und miteinander kommunizieren, müssen sie gewisse Regeln einhalten. Die Gesamtheit dieser Regeln heißt *Protokolle*. Damit ein kryptographisches Protokoll sinnvoll eingesetzt werden kann, sollte es zwei Merkmale besitzen, und daraufhin überprüft werden:

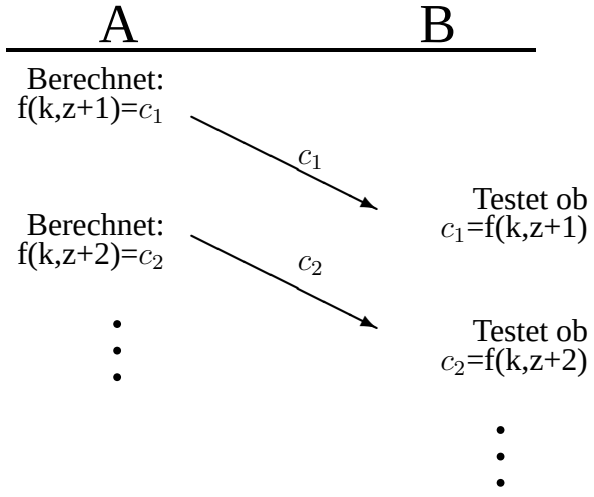
- *Durchführbarkeit (completeness):*
  Das Protokoll muss mit beliebig hoher Wahrscheinlichkeit das richtige Ergebnis liefern, solange sich alle Beteiligten gemäß den Spezifikationen des Protokolls verhalten.
- *Korrektheit:*
  Falls ein Teilnehmer eines Systems betrügt, so wird dies mit beliebig hoher Wahrscheinlichkeit erkannt.

Wie man diese beiden Eigenschaften beweist, werde ich allerdings erst später anhand der Diffie-Hellmann-Schlüsselvereinbarung zeigen.
Wie wir uns vorher schon klargemacht haben, ist eine Möglichkeit die Identität einer Person zu beweisen, indem die Person zeigt, dass sie ein Geheimnis kennt, das sonst niemand kennt. Der einzige Unterschied der einzelnen Verfahren zur Teilnehmerauthentikation, besteht darin, wie dieser Beweis erfolgt.

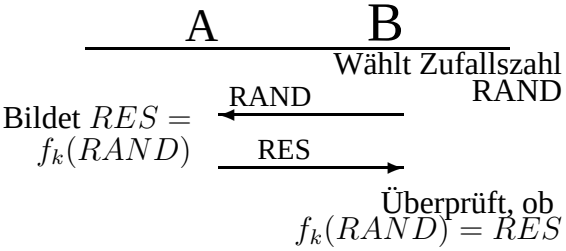
4.1. Passwortverfahren. Die grundlegendsten Verfahren sind die *Passwortverfahren*. Hierbei übermittelt eine Person neben ihrer Identität ein Passwort (oder PIN) an eine zentrale Stelle, die neben der Identität auch das dazugehörige Passwort besitzt.
Bei Passwortverfahren treten grundlegende Schwächen auf, die allerdings gelöst werden können. Einige Beispiele sind:

- *Schlechte Passwörter*
  Hierbei werden Wörter genommen, die von Angreifern erraten werden können.
  Lösung: Verwendung einer sinnlosen Zeichenfolge
- *Zentrale Speicherung des Passwortes*
  Bei der zentralen Stelle sind alle Passwörter gespeichert. Hat sich nun jemand Zugriff zu diese Datei verschafft, kennt dieser alle Geheimnisse.
  Lösung: Das Passwort wird einer Einwegfunktion f unterworfen, dann wird nicht mehr das Passwort, sondern f(Passwort) gespeichert. Möchte sich nun jemand authentifizieren, so sendet er/sie der zentralen Stelle das Passwort zu, diese bildet f(Passwort) und vergleicht diesen Wert mit dem Gespeicherten (siehe Unix etc.).
- *Die Übertragung des Passwortes*
  Hat der Angreifer einmal das Passwort abgehört, so kann er dieses so lange benutzen, so lange es gültig ist.
  Lösung: Die Gültigkeitsdauer muss so gering wie möglich gehalten werden. Im Extremfall darf ein Passwort nur einmal verwendet werden. Dieses Verfahren wird Beispielsweise beim Homebanking verwendet, wo sogenannte TAN's (Transaktionsnummern) benutzt werden.

4.2. Wechselcodeverfahren. Um das letzte Problem zu lösen (dass das gesendete Passwort immer gleich ist), verwendet man sogenannte *Wechselcodeverfahren*. Bei diesem Authentikationsvorgang berechnet der Teilnehmer nach einem genau festgelegten Verfahren aus seinem (konstanten) Geheimnis und einem anderen, veränderlichen Wert seinen Authentikationscode, den er zusammen mit seiner Identität an die Zentrale übermittelt. Diese kennt ebenfalls das Geheimnis und den veränderlichen Wert, kann das Verfahren also wiederholen und die Ergebnisse vergleichen. Wechselcodeverfahren sind genau wie Passwortverfahren unidirektional, d.h. dass für die Authentikation relevanten Informationen nur in eine Richtung übermittelt werden. Ein Bsp. hierfür:
Man benötigt: Verschlüsselungsfunktion f, gemeinsamer Schlüssel k, Anfangszähler $z \in \mathbb{N}$
A will sich gegenüber B authentisieren. Beim ersten Authentisierungsvorgang sendet A den Wert $f(k, z+1)$, beim zweiten $f(k, z+2)$...
Allgemein:
A bildet $c_i = f(k, z+i) \longrightarrow$ B überprüft, ob $f(k, z+i) = c_i$ ist
Problem: Falls sich ein Angreifer A gegenüber als B ausgibt und ihn dazu bringt, dass dieser ihm $c_{i+1} = f(k, z+i+1)$ sendet, kann sich der Angreifer B gegenüber als A ausgeben.
Lösung: Anstelle von z sollte man die aktuelle Zeit verwenden und festlegen, dass eine Authentikationsnachricht nur kurz gültig ist.
Mithilfe des Wechselcodeverfahrens kann man auch ein Signaturverfahren realisieren: Hierbei braucht B nicht das Geheimnis von A zu kennen. Dies funktioniert dann wie folgt: A und B benötigen eine Einwegfunktion f, A benötigt außerdem einen geheimen Startwert z_0 und schätzt ab, wie oft er sich bei B gegenüber authentifizieren möchte, z.B. $n = 10000$. A berechnet nacheinander $z_{i+1} = f(z_i)$ für alle $i = 0, 1, 2, ..., 9999$. Danach wird der Wert $z_n = z_{10000}$ an B übermittelt und A teilt B außerdem noch mit, dass er auch noch die Werte $z_0 ... z_{9999}$ kennt. Wenn die beiden nun kommunizieren wollen, sendet A an B zusätzlich zu seiner Identifikation z_{9999} B testet, ob $f(z_{9999}) = z_{10000}$ ist, falls ja sitzt am anderen Ende A. Bei der nächsten Kommunikation sendet A an B z_{9998} und B, der z_{9999} kennt, testet ob $f(z_{9998}) = z_{9999}$ ist usw.

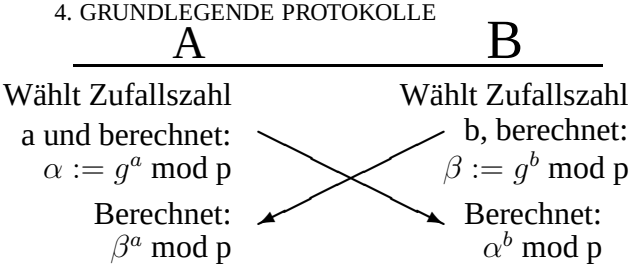


4.3. Challenge-and-Response. Im Gegensatz zu unidirektionalen Verfahren, fließen bei den bidirektionalen Verfahren die Nachrichten sowohl von A nach B, als auch umgekehrt. Das *Challenge-and-Response* Verfahren funktioniert dann so: B stellt eine unvorhersagbare Frage an A, der mit Hilfe seines geheimen Wissens die Antwort berechnen kann, und an B sendet. Der Vorteil an diesem Verfahren besteht darin, dass die Nachricht, die von B erwartet wird, jeweils von A frisch berechnet werden muss, d.h. ein Vorproduzieren der Authentikationsnachrichten ist nicht mehr möglich.
Die technische Anwendung geht wie folgt vor: Beide Seiten kennen eine Einwegfunktion f_k, die vom Schlüssel k abhängt. B wählt nun eine Zufallszahl, RAND (die 'challenge') und sendet diese an A, der wiederum wendet f_k auf RAND an, und bildet so $RES = f_k(RAND)$ (RES='response') und sendet diese an B, dieser überprüft, ob $f_k(RAND) = RES$.



Anstelle einer symmetrischen Funktion f kann man natürlich auch ein Signaturverfahren verwenden, bei dem B das Geheimnis von A nicht zu kennen braucht.

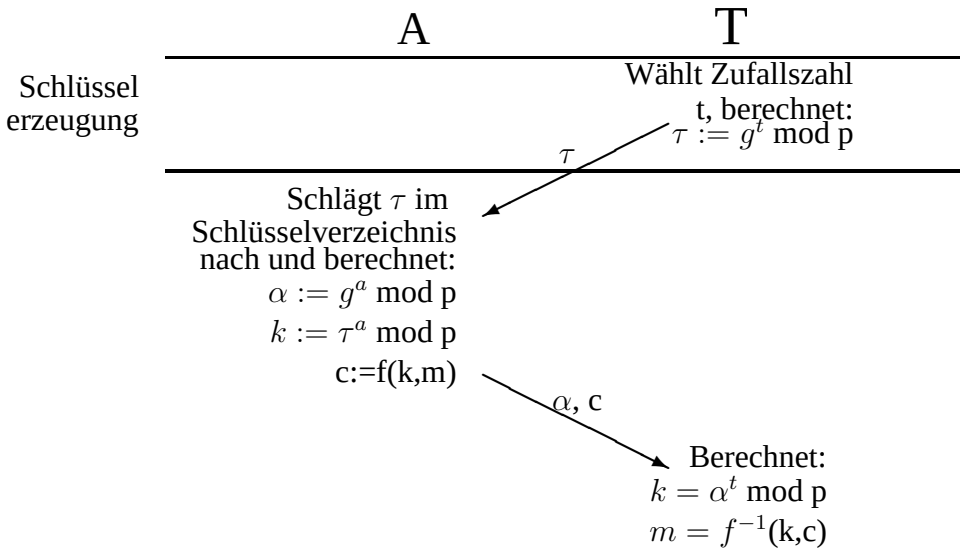
4.4. Diffie-Hellmann-Schlüsselvereinbarung. Anders als bei den vorherigen Protokollen dient die *Diffie-Hellmann-Schlüsselvereinbarung* nicht dem Schlüsselaustausch, sondern, wie der Name schon sagt, der Schlüsselvereinbarung. In der klassischen Kryptographie wird vorausgesetzt, dass vorher über einen geheimen Kanal ein Schlüssel ausgetauscht wurde. Dieser Kanal musste sicher gegenüber Angreifer sein. Bei der Diffie-Hellmann-Schlüsselvereinbarung ist dieser geheime Kanal nicht mehr notwendig, der Schlüssel kann nun in einem öffentlichen Kanal übertragen werden. Die Sicherheit liegt bei diesem Verfahren eng mit der Schwierigkeit der Berechnung des diskreten Logarithmus $modulo\ p$ zusammen.
Das Verfahren basiert auf der Funktion:
$a \longmapsto \alpha := g^a \ mod\ p\ (a \in \mathbb{N})$, die außer ihrer Einwegeigenschaft die Kommutativität ihrer Exponenten mitbringt. Das Protokoll lautet nun wie folgt:
Die Teilnehmer benötigen eine Primzahl p und eine natürliche Zahl g (beide Zahlen müssen nicht geheim sein). Nun wählt Person A ein geheimes a und B ein geheimes b.
A bildet: $\alpha := g^a \ mod\ p$ und B bildet $\beta := g^b \ mod\ p$.
Nun tauschen sie ihre beiden Ergebnisse miteinander aus.
A berechnet: $\beta^a \ mod\ p$
B berechnet:$\alpha^b \ mod\ p$



Daraus haben A und B einen gemeinsamen, geheimen Wert k erhalten.
Wie schon am Anfang von Kapitel 3 versprochen, beweise ich bei diesem Verfahren die Durchführbarkeit und die Korrektheit:

- Das Protokoll ist durchführbar, da A und B ein gemeinsames k erhalten haben:
  $\beta^a \; mod \; p = (g^b)^a \; mod \; p = g^{ba} \; mod \; p = g^{ab} \; mod \; p = (g^a)^b \; mod \; p = \alpha^b \; mod \; p = k$
- Das Protokoll ist korrekt, da ein Angreifer, der p, g und durch mithören α und β kennt, nicht auf k schließen kann.

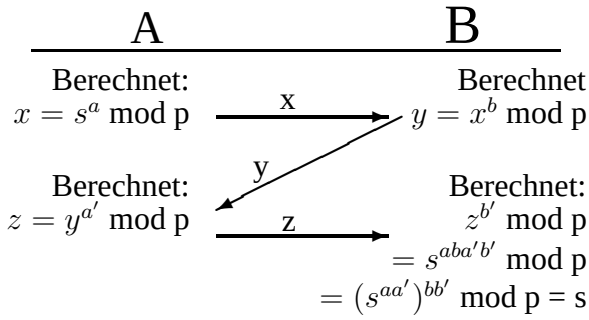
4.5. Das ElGamal-Verschlüsselungsverfahren. Dank eines Herrn namens Taher ElGamal ist man heute in der Lage, durch leichte Variation des Diffie-Hellmann-Schlüsselvereinbarungsprotokolls einen asymmetrischen Verschlüsselungsalgorithmus zu nutzen:
Man benötigt eine Primzahl p und eine natürliche Zahl g, die öffentlich sind. Jeder Teilnehmer besitzt eine privaten Schlüssel t und einen öffentlichen Schlüssel $\tau := g^t \; mod \; p$. Um an den Teilnehmer T eine verschlüsselte Nachricht zu senden geht der Sender wie folgt vor:
Er wählt $a \in \mathbb{N}$ und berechnet:
$\alpha := g^a \; mod \; p$ und $k := \tau^a \; mod \; p$
Dann verschlüsselt er die Nachricht m unter dem Schlüssel k mit Hilfe einer beiderseits bekannten symmetrischen Funktion f und sendet $c = f(k, m)$ zusammen mit α an T. Der Teilnehmer potenziert α nun mit seinem Schlüssel t und erhält dadurch k, damit kann er dann c entschlüsseln.
Der Unterschied zum Diffie-Hellmann-Protokoll liegt nun darin, dass das Senden von α und β entkoppelt ist, es wird nur der öffentliche Schlüssel τ generiert und nie wieder verändert (α muss bei Diffie-Hellmann immer wieder neu generiert werden).



4.6. Das ElGamal Signaturverfahren. Durch eine komplexe Operation ist es beim *ElGamal Signaturverfahren* nicht möglich von einer Signatur auf die eigentliche Nachricht zu schließen. Bei diesem Verfahren wird wieder der geheime Schlüssel t und der öffentliche Schlüssel $\tau = g^t \ mod \ p$ benötigt. Um eine Unterschrift für eine Nachricht m zu erzeugen, geht der Teilnehmer T wie folgt vor: Er wählt r, die zu $p-1$ teilerfremd ist und bildet:
$k = g^r \ mod \ p$
dann berechnet er eine Lösung s der Kongruenz $t \cdot k + r \cdot s \equiv m (mod \ p)$
dies geschieht zum Beispiel, indem er mit Hilfe des erweiterten euklidischen Algorithmus das multiplikative Inverse r^{-1} zu $r \ mod \ p-1$ berechnet und dann $s = ((m - t \cdot k) \cdot r^{-1}) mod \ p-1$ bildet. Die digitale Unterschrift besteht nun aus dem Paar (k, s). Der Empfänger, der die signierte Nachricht $(m(k, s))$ auf die Unterschrift hin prüfen möchte, bildet $g^m \ mod \ p$ und $\tau^k \cdot k^s \ mod \ p$ und vergleicht, ob die Werte identisch sind.

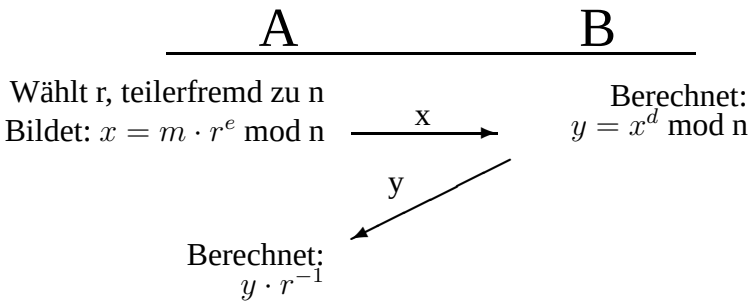
4.7. Shamirs No-Key-Protokoll. Bei *Shamirs No-Key-Protokoll* handelt es sich um ein Verfahren, bei dem zwei Teilnehmer ein Geheimnis miteinander austauschen wollen, ohne dass ein gemeinsamer Schlüssel existiert. Hierbei verschlüsselt A die Nachricht mit seinem Schlüssel a, schickt dieses Ergebnis dann an B, der das Geheimnis mit seinem Schlüssel b verschlüsselt und es wieder an A schickt. Der entschlüsselt es mit seinem a'. Wenn B jetzt dieses Resultat mit seinem b' entschlüsselt, so erhält er die eigentliche Nachricht.
Mathematisch geschieht dies unter Anwendung der diskreten Exponentialfunktion: Beide Teilnehmer einigen sich auf eine große Primzahl p. A erzeugt ein Paar von Zahlen (a, a') mit $aa' = 1 (mod \ p-1)$, B erzeugt (b, b') mit $bb' = 1 \ (mod \ p-1)$. Dabei sind die Zahlen a' bzw. b' die multiplikativen Inversen von a bzw. $b \ modulo \ p-1$. Die Zahlen a und b entsprechen den Schlössern und a' und b' den entsprechenden Schlüsseln. Das Potenzieren mit a bzw. b einer Nachricht s entspricht dem Verschließen, das Potenzieren mit a' bzw. b' das Entfernen des Schlosses, also:
$s = s^{aa'} \ mod \ p$ und $s = s^{bb'} \ mod \ p$

Nun wird beim Senden der Nachricht s wie folgt vorgegangen:



Dieses Verfahren ist asymmetrisch, weil man aus a und der öffentlich bekannten Primzahl p leicht den anderen Schlüssel a' berechnen kann. Die zugrundeliegende Eigenschaft des No-Key-Verfahrens ist:
$f_a(f_b(x)) = f_b(f_a(x))$
Das No-Key Protokoll ist höchstens so sicher, wie das Problem des diskreten Logarithmus, denn kann ein Angreifer den diskreten Logarithmus $dl_y(x)$ von $x = s^a$ zur Basis $y = s^{ab}$ berechnen, so kann er das Geheimnis s berechnen:
$dl_y(x) = dl(x, y) = dl(s^a, s^{ab}) = dl((s^{ab})^{b'}, s^{ab}) = b' \Longrightarrow z^{b'} = s^{bb'} = s$

4.8. Blinde Signaturen. Von einer *blinden Signatur* spricht man dann, wenn ein Teilnehmer B von Teilnehmer A eine verschlüsselte Nachricht zugesandt wird, diese jedoch nicht entschlüsseln kann, allerdings seine Signatur darunter setzen soll. Mathematisch passiert dies aufgrund des RSA-Verfahren:
Ein Dokument m wird von A verschlüsselt:
Sei n der Modul, e der öffentliche und d der geheime Schlüssel des Unterzeichners B. A wählt r, die teilerfremd zu n ist, potenziert diese mit e und schickt dann diesen Wert x an B: $x = x^d \; mod \; n$. B unterzeichnet dieses Dokument, indem er $y = x^d \; mod \; n$ berechnet und zu A schickt. A kann nun die Nachricht m wieder entschlüsseln, indem er den erhaltenen Wert y mit dem Inversen r^{-1} von r multipliziert und erhält so zusätzlich noch die Unterschrift unter dem Dokument:
$y \cdot r^{-1} = x^d \cdot r^{-1} = (m \cdot r^e)^d \cdot r^{-1} = m^d \cdot r^{ed} \cdot r^{-1} = m^d \cdot r \cdot r^{-1} = m^d \; mod \; n.$
Die so erzielte Signatur ist blind, da B nicht weiß was er eigentlich unterschreibt.



5. Fazit

Zuerst habe ich die symmetrische Verschlüsselung eingeführt, die auch ein Bestandteil der asymmetrischen Verschlüsselung ist. Diese hingegen, oft in der heutigen Zeit, gerade im Internet, Netzwerke etc., eingesetzt, habe ich dann mithilfe von Hilfsmitteln (wie z.B. Hashfunktionen und Einwegfunktionen) beschrieben. Danach erläuterte ich dann noch gewisse Regeln zur Kommunikation, die so genannte Protokolle (wie z.B. Challenge-and-Response).
Wie viele Leute (unter anderem auch Ich) vorher angenommen haben, beruht die heutige Kryptographie nicht auf der Unmöglichkeit Codes zu knacken, sondern auf der Tatsache, dass die legalen Teilnehmer ihre Berechnungen in akzeptabler Zeit durchführen können. Die Berechnungen sind also in polynomialer Zeit berechenbar, liegen somit in der Klasse $\mathcal{P}$. Ein Angreifer hingegen muss sich mit Algorithmen befassen, deren Berechnung nicht mehr in polynomialer Zeit liegen, diese liegen also in der Klasse $\mathcal{NP}$, sind somit nicht unknackbar, allerdings mit hinreichend großen Zahlen nicht in akzeptabler Zeit lösbar. Die heutige Forschung konzentriert sich immer noch auf Protokolle, ist also immer noch nicht abgeschlossen. Es bestehen immer noch gravierende Fehler in als sicher geltende Protokolle. Ein Beispiel hierfür wäre das oft verwendete SSH, das im Prinzip zwar sicher ist, jedoch in der Implementierung immer mal wieder Schwachstellen aufweist.

6. Quellen

Meine Arbeit beruht auf dem Buch von A. Beutelspacher, J. Schwenk und K.-D. Wolfenstetter.[**BSW98**]
Zusätzlich habe ich Elemente aus den nachfolgenden Quellen benutzt:
Vorlesung Mathematik für Informatiker 1, WS 01/02 (Dr. B. Kreussler) [**Kre02**]
Cryptography - Theory and Practice (Douglas R. Stinson) INF 525/187 [**Sti95**]

KAPITEL 2

Zero-Knowledge-Verfahren (Alexander Petry)

ZUSAMMENFASSUNG (ABSTRACT)
Dieses Kapitel beschäftigt sich mit den so genannten „Zero-Knowledge-Verfahren“[1]. Bei diesen Verfahren handelt es sich um eine Form von interaktiven Beweisen[2], bei denen *kein Wissen* übertragen wird. Dies ist für die Kryptographie von außerordentlicher Bedeutung, da meist schon normale Passwort-Verfahren das Geheimnis, nämlich das Passwort, übertragen – in verschlüsselter oder unverschlüsselter Form. Ein potentieller Angreifer, das kann dabei auch der Server sein, auf dem man sich einloggen will, könnte sich diesen Umstand zu Nutze machen und in Besitz des Geheimnisses geraten. Das hätte zur Folge, dass sich der Angreifer als jemand anderes ausgeben kann. Diese Schwäche ist natürlich nicht gerade wünschenswert und genau da kommen die „Zero-Knowledge-Verfahren“ ins Spiel.

„Man möchte jemanden davon überzeugen, dass man ein Geheimnis kennt, ihm aber unter keinen Umständen dieses Geheimnis verraten.“

1. Einleitung

1.1. Motivation. Betrachten wir einen kleinen Ausschnitt aus einer Unterhaltung zwischen den Personen Alice und Bob. Alice kennt ein paar sehr vertrauliche Geheimnisse, die sie nicht weitergeben darf. Alice möchte aber unbedingt Bob davon überzeugen, dass sie die Geheimnisse kennt. Eine typische Situation, in der man ein Zero-Knowledge-Verfahren anwenden könnte[3]:

Alice: „Ich kenn’ das Passwort für einen Computer der Bundesbank
und die genaue Zusammensetzung von Cola.“
Bob: „Nein, kennst Du nicht.“
Alice: „Kenn’ ich doch.“
Bob: „Gut, dann beweis’ es!“
Alice: „O.k., ich verrat’s Dir“ [sie flüstert Bob ins Ohr]
Bob: „Super, jetzt weiß ich es auch – da wird sich der *Spiegel* aber freuen!“
Alice: „Verdammt!“

[1] zero (engl.): null, knowledge (engl.): Wissen
[2] näheres dazu ist in dem gleichnamigen Kapitel von Timo Neumann zu finden
[3] Beispiel abgewandelt aus [**Sch96**]

Das Problem hierbei war, dass Alice versuchen wollte, Bob allein durch ihre „Überzeugungskraft“ zu überzeugen – das scheitert natürlich, wenn der Gegenspieler sich nicht so einfach überzeugen lässt. Alice ist in ihrer Situation nichts besseres eingefallen, als Bob ihr Geheimnis zu verraten und ihn auf diesem Wege zufrieden zustellen, leider konnte sie nicht ahnen, dass Bob damit sofort zu einem Redakteur einer bekannten Zeitschrift geht und das Geheimnis veröffentlicht. Alice' Wunsch, Bob zu überzeugen wurde zwar erfüllt, dabei wurde aber ihre Privatsphäre verletzt, da sie ihr Geheimnis verraten musste.
Wir suchen also nach einer Möglichkeit, andere davon zu überzeugen, dass wir im Besitze eines Geheimnisses oder eines Identifikationsmerkmals sind, ohne auch nur das geringste über dieses Geheimnis preisgeben zu müssen. Wir können dadurch verhindern, dass die eigene Identität, die zum Beispiel für den Zugang zu einem Computer nachgewiesen werden muss, missbraucht werden kann.

1.2. Tartaglia und Fior. Im Folgenden betrachten wir ein historisches Beispiel für ein Zero-Knowledge-Verfahren.
Im 16. Jhd. fand der italienische Mathematiker *Nicolò Tartaglia* eine Lösungsformel für kubische Gleichungen. Eine solche Formel war bis dahin noch unbekannt. Da Tartaglia aufgrund seiner Herkunft und seiner Armut keinen akademischen Ruf besaß, hielt er seine Formel geheim. Um aber den anerkannten Rechenmeister *Antonio Maria Fior* von der Existenz seiner Formel zu überzeugen, musste Tartaglia sich etwas einfallen lassen. Fior sollte sich einige kubische Gleichungen ausdenken und Tartaglia zum Bestimmen der Nullstellen geben. Tartaglia bestimmte daraufhin anhand seiner Formel die Lösungen und gab sie Fior zum Vergleichen:

Tartaglia		Fior
		wählt Zahlen r, s und t und berechnet: $(x-r)(x-s)(x-t) =: x^3 + ax^2 + bx + c$
	$\xleftarrow{x^3+ax^2+bx+c}$	
Bestimmt r', s' und t'		
	$\xrightarrow{r',s',t'}$	
		Überprüft, ob $r = r', s = s'$ und $t = t'$

ABBILDUNG 1. Tartaglia und Fior

Durch diese Idee konnte Tartaglia jeden davon überzeugen, dass er eine Formel zur Lösung kubischer Gleichungen gefunden hat, ohne jedoch diese Formel – sein Geheimnis – preiszugeben. Es konnte niemand anhand der übergebenen Informationen eine andere Lösungsformel erstellen, da

ja nur Gleichung und eventuelle Lösungen übergeben wurden und anhand derer kann niemand auf die Formel, die Tartaglia benutzt hat, kommen. Es wird also *kein Wissen* übertragen. Das einzige, was man aus der Unterhaltung mit Tartaglia lernen kann, ist, dass er eine Lösungsformel besitzen muss und damit seine Behauptung beweisen konnte, nicht mehr.

1.3. Forderungen an kryptographische Protokolle. Die folgenden Definitionen sind aus [**BSW98**] entnommen:

DEFINITION 1 (Durchführbarkeit). „Wenn sich die am Protokoll beteiligten Instanzen alle gemäß den Spezifikationen des Protokolls verhalten, muss das Protokoll auch immer (bzw. mit beliebig hoher Wahrscheinlichkeit) das gewünschte Ergebnis liefern.“[4]

DEFINITION 2 (Korrektheit). „Versucht einer der Teilnehmer in einem Protokoll zu betrügen, so kann dieser Betrugsversuch mit beliebig hoher Wahrscheinlichkeit erkannt werden. Das bedeutet, dass die Wahrscheinlichkeit, dass ein Teilnehmer erfolgreich betrügen kann, vernachlässigbar klein ist.“

Das Beispiel „Tartaglia und Fior“ (Abbildung 1) erfüllt diese Eigenschaften:

- Das Protokoll ist *durchführbar*, da Tartaglia zu jeder ihm vorgelegten Gleichung eine Lösung bestimmen kann und jeder Herausforderer kann die bestimmte Lösung überprüfen, indem er sie einfach mit seinen Zahlen vergleicht.
- Es ist ebenfalls *korrekt*, da jemand, der keine Lösungsformel für kubische Gleichungen besitzt, eine Lösung höchstens erraten kann. Aber die Wahrscheinlichkeit, bei vielen Gleichungen immer richtig zu raten, ist vernachlässigbar klein, somit kann auch niemand betrügen.

Das Beispiel erfüllt aber zusätzlich noch eine weitere wichtige Eigenschaft, die nicht alle Protokolle erfüllen: Tartaglia konnte seine Behauptung, eine Lösungsformel für kubische Gleichungen zu besitzen, beweisen, ohne die Formel zu verraten. Diese Eigenschaft nennt man *„Zero-Knowledge-Eigenschaft“*. In den nächsten Abschnitten werden wir diese Eigenschaft genauer betrachten, insbesondere, wie man sie formal definieren und überprüfen kann.

2. Identifikations-Verfahren

In sehr vielen Situationen ist es wichtig, eine Person genau identifizieren zu können. Zum Beispiel bei der Anmeldung an einem Computer, der wichtige Daten speichert und nicht jedem Zugang gewährt, oder an einem Geldautomat – man möchte ja, dass nur der rechtmäßige Inhaber der Kreditkarte Geld abheben kann. Für all solche Situationen benötigt man Identifikations-Verfahren, gewöhnlich werden dabei Passwörter oder PINs[5] verwendet.

[4] im Englischen wird diese Eigenschaft auch „completeness“ genannt

[5] **P**ersonal **I**dentifi cation **N**umbers

Obwohl diese Verfahren schon seit längerer Zeit eingesetzt werden, bergen sie auch Gefahren in sich. Jeder, dem ich mein Passwort oder meine PIN mitteilen muss, kann diese Information dazu benutzen vorzugeben, „ich“ zu sein. Das kann genauso gut auch der Administrator eines Computers, zu dem man Zugang benötigt, sein. Der Administrator kann beispielsweise das Passwort während der Eingabe durch ein Programm auslesen lassen. Wenn ich dieses Passwort noch an anderen Stellen verwende, erhält der Administrator ebenfalls Zugang dazu.

Die Zero-Knowledge Identifikations-Verfahren bieten eine Möglichkeit, diese Probleme zu vermeiden. Wie in dem „Tartaglia und Fior“ Beispiel werden bei einem solchen System zwei Instanzen anzutreffen sein – die, die sich autorisieren lassen möchte und diejenige, die die Autorisierung vornimmt, wobei hier das Passwort aber nicht verraten wird oder berechnet werden kann.

2.1. Interaktive Beweissysteme. Identifikations-Verfahren bauen auf Interaktion und Kommunikation auf, deshalb betrachten wir in diesem Abschnitt kurz die so genannten „interaktiven Beweissysteme“[6]. In einem späteren Kapitel wird noch näher auf solche Beweissysteme und ihre formale Definition eingegangen werden.

In interaktiven Beweissystemen gibt es zwei Teilnehmer, den Prover und den Verifier. Im Folgenden bezeichnet *„Peggy“* den Prover und *„Vic“* den Verifier. Peggy besitzt ein nur ihr bekanntes Geheimnis[7]. Sie versucht dann mittels des Beweissystems, Vic davon zu überzeugen, dass sie dieses Geheimnis kennt[8].

- In unserem Anfangsbeispiel hatte Alice die Rolle der Peggy inne und diverse Geheiminformationen, sie konnte Bob, der den Vic spielte, dadurch überzeugen, dass sie ihm diese Geheimnisse verraten hat, dadurch allerdings kannte auch Bob die Geheimnisse.
- Tartaglia (Peggy) hingegen konnte Fior (Vic) durch ein ausgeklügeltes Aufgaben-Antwort Spiel von seinem Geheimnis überzeugen. Er musste sein Geheimnis dabei nicht verraten.

Es gibt also zahlreiche Beispiele für interaktive Beweissysteme, die am häufigsten anzutreffende praktische Anwendung wird wohl die des Nachweises seiner Identität einem anderen gegenüber sein. Auf andere Anwendungen wird in einem späteren Kapitel eingegangen – zum Beispiel gibt es auch interaktive Beweise, die einen mathematischen Beweis ersetzen (Nicht-Isomorphismus von Graphen, Existenz von Hamiltonzyklen etc.).

Ein solcher interaktiver Beweis läuft normalerweise wie folgt ab, wobei meist Peggy anfängt und Vic den Ablauf beendet:

(1) Eine Nachricht des anderen empfangen.
(2) Diverse Rechnungen durchführen.
(3) Eine Nachricht an den anderen versenden.

[6] eine andere, eventuell bessere Bezeichnung wäre: „interactive proof of knowledge“ – interaktiver Wissensbeweis

[7] beispielsweise einen geheimen Schlüssel k oder einen Isomorphismus zwischen zwei großen Graphen

[8] Es wird nichts über die Art und Weise ausgesagt, wie der Beweis erbracht wird

Diese Schritte werden im allgemeinen mehrmals wiederholt. Vic akzeptiert Peggy's Beweis nur, wenn Peggy auf alle Fragen („Challenges“) richtig antwortet.

Ein Prover bzw. Verifier wird als *„ehrlich“* bezeichnet, wenn er alle Aktionen so durchführt, wie es das Protokoll vorsieht. Ein Prover wird *„betrügerisch“* genannt, wenn er das Geheimnis nicht kennt aber dennoch versucht, den Verifier davon zu überzeugen. Ebenso kann es auch einen betrügerischen Verifier geben, der anhand seiner Fragen versucht, an das Geheimnis des Provers zu kommen. Folgt einer der Beteiligten nicht der Syntax des Protokolls, fällt dies sofort auf und die Kommunikation wird abgebrochen. Folglich können betrügerische Teilnehmer nur bei ihren internen Berechnungen schummeln.

2.2. Die Magische Tür. In diesem Abschnitt möchte ich ein sehr bekanntes Beispiel vorstellen. Es handelt sich hierbei ebenfalls um einen interaktiven Beweis, der aber zusätzlich noch die Zero-Knowledge-Eigenschaft besitzt.

Peggy möchte Vic gegenüber den Nachweis erbringen, dass sie die geheime Zahlenkombination zu einer magischen Tür kennt. Die magische Tür ist für dieses Protokoll in ein Haus eingebaut:

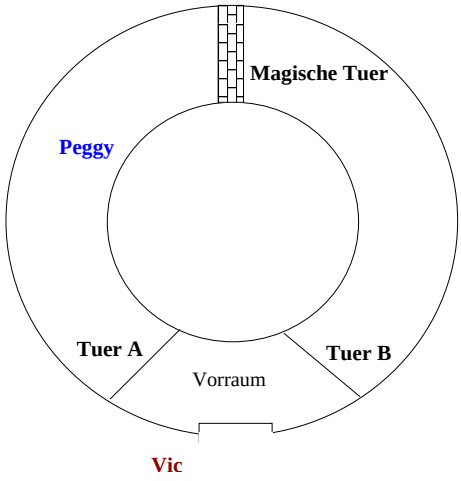


ABBILDUNG 2. Grundriss des Gebäudes mit der magischen Tür

Als erstes betritt Peggy (allein) den Vorraum und schließt hinter sich die Tür. Dann sucht sie sich eine der Türen „A“ oder „B“ aus, durch die sie hindurchgeht und hinter sich wieder schließt.

Jetzt erst darf Vic den Vorraum betreten. Er kann nicht feststellen, durch welche von den beiden Türen Peggy gegangen ist – er sucht sich also willkürlich eine davon aus und erwartet, dass Peggy aus genau dieser herauskommt. Wenn Peggy zuvor durch eben diese Tür gegangen ist, hat sie keine Probleme, aus dieser wieder herauszukommen – ist sie jedoch durch die andere Tür gegangen, muss sie ihr Geheimnis einsetzen, um die magische Tür öffnen und aus der richtigen Tür herauskommen zu können.

Dieser Ablauf wird nun mehrmals wiederholt, sagen wir n mal. Wenn Peggy bei jedem Durchlauf aus der richtigen Tür herauskommt, ist Vic davon überzeugt, dass Peggy das Geheimnis kennt.

Wenn Peggy das Geheimnis kennt, kann sie immer aus der richtigen Tür herauskommen und Vic immer überzeugen, das Verfahren ist also *durchführbar*. Warum aber ist Vic sich so sicher? Nehmen wir einmal an, dass wir das Verfahren nocheinmal mit einer Eve statt Peggy durchführen, wobei Eve das Geheimnis allerdings nicht kennt. Jedesmal, wenn Eve sich eine der Türen aussucht, kann sie nur mit einer Wahrscheinlichkeit von $(\frac{1}{2})$ die richtige Entscheidung treffen – nämlich genau die Tür auszusuchen, die sich Vic später wünschen wird. Da das Verfahren n mal wiederholt wird, verringern sich Eve's Chancen auf $(\frac{1}{2})^n$ – das sind bei 20 Durchläufen bereits weniger als 10^{-6}. Auf diesem Weg kann sie Vic niemals (oder jedenfalls nur mit einer vernachlässigbar kleinen Wahrscheinlichkeit) davon überzeugen, das Geheimnis zu kennen. Das Verfahren ist also auch *korrekt*.

Es bleibt aber noch eine Frage ungeklärt: *„Wie lässt sich nachweisen, ob dieses Protokoll die Zero-Knowledge-Eigenschaft besitzt?“*

Die Zero-Knowledge-Eigenschaft besagt ja, dass der Verifier keine Möglichkeit hat, Informationen über das Geheimnis oder sonstiges Wissen zu erhalten. Vorerst möchte ich aber die Grundsteine für eine formale Definition legen. Wenn man sich die Kommunikation zwischen Peggy und Vic ansieht, erkennt man, dass man den Ablauf auf die übertragenen Daten reduzieren kann. In unserem Beispiel besteht ein Durchlauf aus folgendem Tupel: (v_i, p_i), wobei $v_i, p_i \in \{A, B\}$. Das bedeutet im Durchlauf i hat sich Vic dafür entschieden, Peggy aus Tür v_i kommen zu sehen. Kennt Peggy das Geheimnis, so ist $v_i = p_i$ ansonsten muss die Gleichheit nicht immer gelten[9]. Das heißt Vic akzeptiert dann und nur dann, wenn $v_i = p_i, i = 1, \ldots, n$ gilt.

Die Menge[10] aller Tupel ist genau das, was Vic oder ein beliebiger Angreifer aus der Interaktion zwischen Vic und Peggy gewinnen kann. Wenn nun jeder, insbesondere Vic, sich genauso eine Menge erzeugen kann, ohne jemals mit Peggy kommuniziert zu haben, können wir die Zero-Knowledge-Eigenschaft bereits nachweisen! Man kann die beiden Mengen nicht von einander unterscheiden, das bedeutet, im nachhinein kann niemand mehr sagen, welche der beiden Mengen aus der „echten“ Kommunikation mit Peggy entstanden ist und welche nicht. Es kann also kein Wissen übertragen werden, denn wie soll man etwas Unbekanntes aus Kommunikationsdaten berechnen können, wenn man sich die Daten selbst erzeugen kann – es können in selbsterzeugten Daten nicht plötzlich mehr Informationen enthalten sein, als man an Informationen in die Berechnung der Daten hat einfließen lassen.

Um wieder auf unser Beispiel zurück zu kommen: *„Wie kann sich nun Vic solche Daten selbst erzeugen?“* Stellen wir uns dazu einfach vor, Vic würde eine Videoaufzeichnung des gesamten Geschehens, das er beobachten kann, erstellen. Wir würden auf dem Videoband das Haus sehen, dann wie eine Frau das Haus betritt und die Tür hinter sich schließt. Als nächstes sehen wir dann Vic, wie er das Gebäude betritt und die Bezeichnung einer der Türen in das Bauwerk hineinruft – wir sehen dann, wie die zuvor in das Haus gegangene Frau seinem Wunsch nachkommt und durch die gewünschte Tür tritt. So sieht ein Ablauf aus, der mit der echten Peggy aufgezeichnet

[9] s.a. Definition 2 (Korrektheit), auf Seite 23

[10] die Mengen sind geordnete Multimengen

wurde. Als nächstes versuchen wir genauso einen Ablauf zu erzeugen, ohne Peggy zu benötigen. Wir benötigen eine neue Instanz, den *„Simulator S“* – dieser wird quasi die Rolle der Peggy übernehmen. Außerdem brauchen wir für dieses Beispiel noch eine weibliche Person, nennen wir sie Eve, die Peggy auf dem Video darstellen soll.

Die Erzeugung des neuen Videobandes kann man wie folgt bewerkstelligen. Der Simulator versucht Vic's Wunsch zu erraten, also welche Tür Vic wählen wird. Das Ergebnis teilt er Eve mit, die sich daraufhin vom Vorraum aus durch diese geratene Tür begibt. Wenn S richtig geraten hat und die Türen tatsächlich dieselben waren, kann Eve aus der Tür herauskommen. Wir haben somit einen korrekten Durchlauf erzeugt und können diese Szene auf dem Video belassen. Hat der Simulator hingegen die falsche Tür geraten, bleibt Eve nichts anderes übrig, als aus der falschen Tür herauszukommen – sie kennt ja das Geheimnis, das die magische Tür öffnen würde, nicht und kann in diesem Fall auch gar nicht aus der richtigen Tür herauskommen. Der Simulator „verwirft“ diese Szene, da sie ja inkorrekt ist und beginnt eine neue Szene aufzuzeichnen. Im Normalfall wird der Simulator $2n$ Versuche benötigen, um n „gute“ Szenen aufzuzeichnen.

Wenn man jetzt die beiden Videoaufzeichnungen vergleicht, stellt man fest, dass die statistische Verteilung der Tupel auf beiden Bändern gleich ist – diese hängt nur von Vic ab[11]. Aus diesem Grund ist es unmöglich, die beiden Bänder von einander zu unterscheiden. Diese Abhängigkeit der Verteilung war auch der Grund dafür, dass wir einen Simulator eingeführt haben, da wir Vic als eigenständige Instanz benötigen, damit er seine Strategie – hier die Türen auszuwählen – anwenden kann.

3. Zero-Knowledge

In diesem Abschnitt werden wir die *„Zero-Knowledge Eigenschaft“* formal definieren und noch weitere Beispiele kennenlernen, insbesondere auch Beispiele, die man in der Praxis einsetzen kann.

Von jetzt an betrachten wir den Prover und den Verifier als in Polynomzeit laufende Turingmaschinen. Die durch die Turingmaschinen beschriebenen Algorithmen bezeichnen wir mit P für den Prover und mit V für den Verifier, einen eventuell betrügerischen Verifier bezeichnen wir mit V^*. Ein interaktives Beweissystem inklusive Kommunikation bezeichnen wir mit (P, V). Die Turingmaschinen verfügen jeweils über ein Arbeitsband und ein Eingabeband, ein Kommunikationsband, das beide Turingmaschinen nutzen können und haben jeweils Zugriff auf einen „idealen“ Zufallszahlengenerator – dieser wird dafür benutzt, zufällige Entscheidungen zu treffen, zum Beispiel *„Tür A oder Tür B“* oder, um *„r, s und t“* zu bestimmen. Peggy kennt ein Geheimnis zu einem Objekt x – zum Beispiel einen geheimen Schlüssel oder einen Isomorphismus zwischen zwei Graphen. Dieses Objekt ist die gemeinsame Eingabe an P und V. Rechnungen, die durch die Turingmaschinen durchgeführt werden, sind probabilistisch, da zufällige Entscheidungen getroffen werden können.

[11] er kann seine Wahl entweder zufällig treffen oder nach einer gewissen Strategie – zum Beispiel könnte er beabsichtigen durch seine Wahl, mehr Informationen über das Geheimnis zu erfahren

Das nachfolgende Bild[12] skizziert den Aufbau eines solchen Systems:

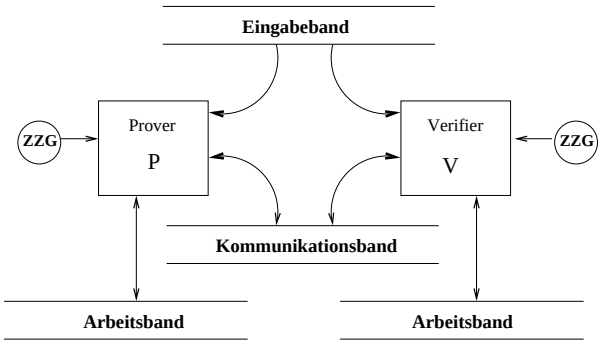


ABBILDUNG 3. Aufbau eines interaktiven Beweissystems (P, V)

Wie bereits weiter oben erwähnt, kann man einen interaktiven Beweis auch durch die übertragenen Daten beschreiben. Diesen Sachverhalt halten wir in folgender Definition[13] fest:

DEFINITION 3 (Transkript). Sei x die gemeinsame Eingabe an ein interaktives Beweissystem (P, V). In jedem Schritt wird eine Nachricht gesendet. Wir nehmen an, dass ein Ablauf n Schritte umfasst. Desweiteren nehmen wir an, dass die erste Nachricht durch den Prover geschickt wird. m_i bezeichnet dann die Nachricht, die im i-ten Schritt gesendet wird. Die Nachrichten $m_1, m_3, \ldots$ werden vom Prover zum Verifier geschickt und die Nachrichten $m_2, m_4, \ldots$ vom Verifier zum Prover. Das Transkript der gemeinsamen Berechnung von P und V^* wird durch

$$tr_{P,V^*}(x) := (m_1, \ldots, m_n)$$

definiert. Wobei $tr_{P,V^*}(x)$ ein ***akzeptierendes Transkript*** ist, wenn V^* nach der letzten Nachricht akzeptiert.

Das Transkript hängt nur von den Zufallszahlen ab, die von den Algorithmen P und V^* während ihrer Berechnungen benutzt werden, nicht jedoch von der Eingabe x.

DEFINITION 4 (Zero-Knowledge Eigenschaft). Ein interaktives Beweissystem (P, V) ist ***(perfekt) zero-knowledge***, wenn es einen in Polynomzeit laufenden *„Simulator $S(V^*, x)$“* gibt, der für jeden Verifier V^* – also ehrlich oder nicht – bei Eingabe x ein akzeptierendes Transkript t von P und V^* ausgibt, so dass diese simulierten Transkripte die gleiche Wahrscheinlichkeitsverteilung besitzen, als ob sie von einer echten Kommunikation zwischen P und V^* stammten.

Diese Definition beinhaltet *alle* Verifier, also auch die betrügerischen. Daraus folgt unmittelbar, dass „zero-knowledge“ eine Eigenschaft des Provers P sein muss. Diese Eigenschaft schützt den Prover davor, dass ein Verifier *„Wissen“* anhand der Kommunikation mit ihm erlangen kann.

Um die Definition zu verstehen, müssen wir verstehen, welche Rolle der Simulator spielt. S ist ein Algorithmus, der zu (P, V^*) ein akzeptierendes Transkript liefert, ohne mit dem echten

[12] entnommen aus [**Weg96**, Seite 290]

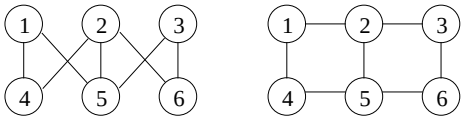
[13] zu fi nden in [**DK02**, Kaptitel 4.2.3, Seite 71]

Proover zu kommunizieren. Um ein solches Transkript zu erzeugen, schlüpft S in die Rolle des Provers und versucht V^*'s Fragen zu beantworten. Da S keinen Zugriff auf P's Geheimnis hat, ist die Wahrscheinlichkeit, ein akzeptierendes Transkript zu erzeugen deutlich kleiner als 1 – P benutzt ja das Geheimnis, um auf alle Fragen korrekt zu antworten. Angenommen S könnte mit einer größeren Wahrscheinlichkeit ein akzeptierendes Transkript erzeugen, dann kann S den Verifier fälschlicherweise davon überzeugen, dass er das Geheimnis kennt, obwohl das natürlich nicht so ist – das Protokoll wäre nicht mehr korrekt. Aus diesem Grund schlagen viele Versuche, ein akzeptierendes Transkript zu erzeugen, fehl. Die Eigenschaft, solche Transkripte erzeugen zu können reicht noch nicht aus, um „zero-knowledge“ zu sein. Aufgrund des probabilistischen Anteils, besitzen die Transkripte (echt oder simuliert) eine Wahrscheinlichkeitsverteilung. Der letzte Teil der Definition sagt aus, dass die Verteilung der Transkripte, die von dem echten Prover und V^* erzeugt werden, gleich der Verteilung sein muss, die der Simulator S mit V^* erzeugt.

In den nächsten Abschnitten werden zwei Beispiele für Zero-Knowledge Identifikationsbeweise vorgestellt. Diese Beweise kann man auch in der Praxis verwenden, um zum Beispiel Passwortverfahren zu ersetzen.

3.1. Isomorphismen von Graphen. Bei diesem Beispiel geht es um die Isomorphie von zwei großen Graphen. Im allgemeinen ist es sehr schwer zu entscheiden, ob zwei beliebige, große Graphen isomorph zu einander sind oder nicht, ebenso schwer ist die Berechnung eines Isomorphismus – im Gegensatz dazu ist es effizient entscheidbar, ob eine gegebene Permutation π ein Isomorphismus der beiden Graphen darstellt oder nicht[14].

Ein kleines Beispiel zum Graphenisomorphieproblem:



G1 G2

ABBILDUNG 4. Zwei isomorphe Graphen G_1 und G_2

Ein möglicher Isomorphismus ist:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 5 & 3 & 4 & 2 & 6 \end{pmatrix}$$

Im folgenden Protokoll identifiziert sich Peggy Vic gegenüber dadurch, dass sie beweist, einen Isomorphismus zwischen zwei großen Graphen zu kennen.

[14] vergleiche [**BSW98**, Kapitel 8.5]

Peggy erzeugt sich einen großen Graphen G_0 und wählt eine zufällige Permutation π auf dem Graphen. Mit dieser Permutation erzeugt sie sich einen weiteren Graphen G_1, der durch Anwendung der Permutation auf G_0 entstanden ist. Das Tupel (G_0, G_1) veröffentlicht Peggy[15]. Die Permutation π bleibt allerdings ihr Geheimnis. Das Tupel steht somit jedem zur freien Verfügung, da es aber sehr schwer ist, anhand der Graphen einen Isomorphismus zu berechnen, kann auch niemand Peggy's Geheimnis einfach so berechnen. Der Isomorphismus spielt hier die Rolle Tartaglia's Lösungsformel für kubische Gleichungen (Seite 22) oder des geheimen Zahlencodes aus dem Beispiel „Die magische Tür" (Seite 25).

Der Vorteil eines solchen mathematischen Ansatzes ist natürlich, dass jetzt jede Person ein individuelles Geheimnis haben kann, ohne das Verfahren ändern zu müssen. Es gibt unendlich viele „Instanzen" des Graphenisomorphieproblems, aber zum Beispiel nur endlich viele Lösungsformeln für kubische Gleichungen. Der Nachteil bei der magischen Tür war, dass man erst ein komplexes Gebäude errichten muss, um den Beweis durchführen zu können – hier kann diese Aufgabe von Computern übernommen werden.

Der Ablauf einer Runde des Protokolls wird in Abbildung 5 dargestellt:

Ablauf

Peggy		Vic
wählt zufällige Permutation σ und $j \in \{0,1\}$ bildet $H := \sigma(G_j)$	$\xrightarrow{H}$	
		wählt zufälliges $b \in \{0,1\}$
	$\xleftarrow{b}$	
$\tau := \sigma\pi^{j-b}$	$\xrightarrow{\tau}$	
		akzeptiert, wenn $H = \tau(G_b)$

ABBILDUNG 5. Autorisierung durch isomorphe Graphen

Dieser Ablauf wird n mal wiederholt. Peggy erzeugt sich als erstes einen weiteren Graphen, der durch die Anwendung einer zufälligen Permutation σ auf den Graphen G_j entsteht[16], wobei $j \in \{0,1\}$ zufällig gewählt wurde. Dieser neue Graph ist zu den beiden Graphen G_0 und G_1 isomorph, nur kann dies Vic genauso schwer überprüfen, wie die Isomorphie zwischen G_0 und G_1. Der Graph H wird nun an Vic verschickt, der sich daraufhin etwas wünschen darf. Vic möchte jetzt entweder den Isomorphismus zwischen H und G_0 oder den zwischen H und G_1

[15] zum Beispiel kann sie es in einer öffentlichen Datenbank ablegen (vgl. Public-Key-Server)

[16] hier kommt der Zufallszahlengenerator zum Tragen

sehen. Peggy kann diese Frage immer mit Hilfe der geheimen Permutation π lösen („Durchführbarkeit“). Vic kann sich leicht davon überzeugen, ob die empfangene Permutation τ wirklich einen Isomorphismus beschreibt.

Nehmen wir an, eine Eve möchte Vic davon überzeugen, das Geheimnis zu besitzen, obwohl dies nicht der Fall ist. Eve kann in jeder Runde nur eine der Fragen beantworten, wir können also annehmen, dass die Wahrscheinlichkeit, dass Eve auf die Frage richtig antworten kann höchstens $\frac{1}{2}$ beträgt. Desweiteren können wir annehmen, dass die Wahrscheinlichkeit auch mindestens $\frac{1}{2}$ beträgt, da Eve im Voraus raten kann, welche Frage Vic stellen wird und dementsprechend die Permutation τ festlegt. Es folgt, dass die Betrugswahrscheinlichkeit in einer Runde genau $\frac{1}{2}$ betägt, bei n Runden, liegt sie folglich nur noch bei $(\frac{1}{2})^n$. Somit ist das Protokoll korrekt.

Nachweis der Zero-Knowledge Eigenschaft. Um die Zero-Knowledge Eigenschaft nachzuweisen, müssen wir einen Simulator S angeben, der uns korrekte Transkripte erzeugen kann. Ein Transkript besteht hier aus dem Tripel (H, b, τ), wobei ein akzeptierendes Transkript vorliegt, genau dann wenn $H = \tau(G_b)$ gilt. Die erzeugten Transkripte müssen dabei die gleiche Wahrscheinlichkeitsverteilung besitzen, wie die Transkripte, die bei der realen Kommunikation von Vic und Peggy entstehen würden.

Simulator		Vic
wählt zufällige Permutation σ und rät $\tilde{b} \in \{0, 1\}$ bildet $H := \sigma(G_{\tilde{b}})$		
	$\xrightarrow{H}$	
		wählt zufälliges $b \in \{0, 1\}$
	$\xleftarrow{b}$	
$\tilde{b} = b$: $\tau := \sigma$ und liefere Transkript (H, b, τ) zurück $\tilde{b} \neq b$: verwerfen		

ABBILDUNG 6. Simulator zum Nachweis der Zero-Knowledge Eigenschaft

Sowohl in einem echten Dialog als auch in einem simulierten Dialog, taucht die gleiche Anzahl an (zufälligen) Tripeln (H, b, τ) auf, so dass

$$H = \tau(G_b)$$

gilt. Das bedeutet, dass ein Außenstehender die beiden Dialoge unmöglich unterscheiden kann. *Es folgt, dass das Verfahren die Zero-Knowledge Eigenschaft besitzt.*

3.2. Das Fiat-Shamir Identifikationsverfahren. Das folgende Protokoll besitzt einen großen praktischen Anreiz – es wird in der ein oder anderen Form zum Beispiel bei Chipkarten verwendet. Die Algorithmen sind effizient berechenbar und der Speicherplatzbedarf ist akzeptabel – das ist ein Argument, warum man Graphen in solchen Situationen nicht benutzen würde.

Fast alle kryptographischen Protokolle bestehen aus einer *„Schlüsselerzeugungsphase“* und der anschließenden Verwendung der Schlüssel während des Protokollablaufs. Peggy erzeugt zunächst zwei große verschiedene Primzahlen[17] p und q und berechnet daraus das Produkt $n := pq$. Als nächstes berechnet sich Peggy eine Quadratwurzel y eines Elementes $x \in \mathbb{Z}_n^*$. Sie wählt dazu ein beliebiges Element $y \in \mathbb{Z}_n^*$ und definiert $x := y^2$. Das Paar (n, x) *wird als Public-Key veröffentlicht* – zum Beispiel wieder in einem öffentlichen Register – *die Zahl y bleibt Peggy's Geheimnis*. Das Verfahren beruht darauf, dass es sehr schwer ist, also nicht in polynomialer Zeit, eine Quadratwurzel eines Elements aus $\mathbb{Z}_n^*$[18] zu ziehen. Die genauen Definitionen und Beweise dazu kann man unter [**DK02**, Kapitel A.5 und folgende] nachlesen.

Peggy weist ihre Identität dann dadurch nach, dass sie durch einen interaktiven Beweis Vic davon überzeugt, eine Quadratwurzel von x zu kennen.

Peggy		Vic
wählt zufälliges $r \in \mathbb{Z}_n^*$ und berechnet $a := r^2 \ mod \ n$	$\xrightarrow{a}$	
		wählt ein zufälliges $b \in \{0, 1\}$
	$\xleftarrow{b}$	
berechnet $z := ry^b \ mod \ n$	$\xrightarrow{z}$	
		akzeptiert $\Leftrightarrow z^2 \equiv ax^b \ mod \ n$ gilt

ABBILDUNG 7. Fiat-Shamir Identifikationsverfahren

Peggy berechnet anfangs eine Quadratwurzel r eines Elements a aus $\mathbb{Z}_n^*$, damit kann sie Vic gegenüber beweisen, dass sie zu a eine Quadratwurzel, nämlich r, kennt. Dieses Objekt übernimmt dieselbe Funktion wie der Graph H aus dem vorhergehenden Beispiel. Vic bekommt dieses Element übermittelt und kann jetzt bestimmen, ob er die Quadratwurzel von a sehen möchte oder die Quadratwurzel von ax. Dadurch, dass Peggy die geheime Quadratwurzel y von a kennt, kann sie jede Frage, die Vic stellen wird, korrekt beantworten – das Protokoll ist somit durchführbar.

Die Korrektheit nachzuweisen ist bei diesem Protokoll etwas schwieriger. Wir hatten gesagt, dass ein Betrüger allerhöchstens während seiner eigenen Berechnungen schummeln kann, diesen Fakt nutzen wir jetzt aus, damit eine Eve mit Wahrscheinlichkeit $\frac{1}{2}$ betrügen kann. Eve versucht zu

[17] Wie das genau funktioniert, kann man in Kapitel 5 (Primzahlen) ab Seite 63 nachlesen

[18] genauer: eines Elementes $x \in QR_n$, also aus dem quadratischen Restklassenring modulo n.

erraten, für welche Möglichkeit Vic sich entscheiden wird, erhält dadurch ein Bit $\tilde{b}$ und präpariert dementsprechend das übermittelte a, indem sie für a einfach $r^2x^{-\tilde{b}}$ berechnet. Als z verschickt sie r. Wenn sie richtig geraten hat, wird Vic akzeptieren, da die Gleichung $z^2 \equiv ax^b \; mod \; n$ erfüllt ist. Es ergibt sich, dass Eve in jeder Runde mit mindestens einer Wahrscheinlichkeit von $\frac{1}{2}$ betrügen kann. Sie kann auch nur höchstens mit einer Wahrscheinlichkeit $\frac{1}{2}$ betrügen: Angenommen Eve könnte mit einer höheren Wahrscheinlichkeit betrügen, dann kennt sie ein a, zu dem sie beide Fragen beantworten kann. Das heißt, sie kann $z_1^2 = a$ und $z_2^2 = ax$ berechnen, sie kann also auch die Quadratwurzel y aus x berechnen ($y = \frac{z_2}{z_1}$). Eve besitzt demnach einen Algorithmus $\mathcal{A}$, der bei Eingabe x die Quadratwurzel y von x ausgibt. Nach unserer Voraussetzung, dass dies nicht in Polynomzeit berechenbar wäre muss unsere Annahme falsch gewesen sein, Eve kann also auch nur höchstens mit Wahrscheinlichkeit $\frac{1}{2}$ betrügen. Insgesamt folgt, dass Eve pro Runde genau mit Wahrscheinlichkeit $\frac{1}{2}$ betrügen kann, bei t Runden, verringert sich die Wahrscheinlichkeit, immer zu betrügen, auf $(\frac{1}{2})^t$. Das Protokoll ist damit korrekt.

Nachweis der Zero-Knowledge Eigenschaft. Die Menge der Transkripte ist bei diesem Protokoll als

$$\mathcal{T}(x) := \{(a, b, z) \in QR_n \times \{0, 1\} \times \mathbb{Z}_n^* | z^2 \equiv ax^b \; mod \; n\}$$

definiert. Wir geben wieder einen Simulator an, der unter Mithilfe Vic's solche Transkripte erzeugt.

Simulator S		Vic
versucht $\tilde{b} \in \{0, 1\}$ "zu raten" wählt zufälliges $r \in \mathbb{Z}_n^*$ und berechnet $a := r^2 * x^{-\tilde{b}} \; mod \; n$	$\xrightarrow{a}$	
		wählt ein zufälliges $b \in \{0, 1\}$
	$\xleftarrow{b}$	
wenn $\tilde{b} = b$, dann $z := r$ und Ausgabe (a, b, z), sonst Abbruch		
	$\xrightarrow{z}$	
		akzeptiert, da: $z^2 \equiv r^2 \equiv r^2 * x^{-b} * x^b \equiv ax^b \; mod \; n$

ABBILDUNG 8. Simulator für das Fiat-Shamir Identifikationsverfahren

Der Simulator S versucht Vic's Wunsch zu erraten, dies gelingt ihm mit einer Wahrscheinlichkeit von $\frac{1}{2}$. Der Simulator muss mit Vic kommunizieren, genauso wie bei der „magischen Tür“ oder bei dem Graphenisomorphieproblem, da Vic seinen Wunsch nicht unbedingt zufällig wählen

muss. Vic kann auch eine ganz gewisse Strategie verfolgen, um an Informationen zu gelangen – ein zero-knowledge interaktives Beweissystem war ja mit V^* definiert[19].

Die Erwartung ist nun, dass S bei jedem zweiten Versuch ein akzeptierendes Transkript $(a', \tilde{b}, z')$ $\in \mathcal{T}$ zurückliefert. Dieses Transkript kann man nicht von einem echten Transkript $(a, b, z) \in \mathcal{T}$, das von (P, V^*) produziert wurde, unterscheiden. Das Protokoll ist *„zero-knowledge"*.

3.3. Schwächen von Zero-Knowledge Verfahren. Bisher sind wir bei den vorgestellten Verfahren auf keine Probleme gestoßen. Die folgende Schwäche ist aber leider nicht trivial und trifft auf alle Zero-Knowledge Verfahren zu. Es handelt sich hier um eine Art „man-in-the-middle attack", da sich der Betrüger zwischen zwei Instanzen befindet.

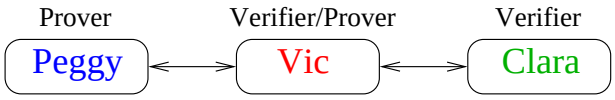


ABBILDUNG 9. Verschränkte Kommunikation zwischen Peggy, Vic und Clara

Durch diese Konstellation ist es Vic möglich, sich Clara gegenüber als Peggy auszugeben. Das funktioniert sogar recht leicht. Am Beispiel des Fiat-Shamir Verfahrens, sieht das folgendermaßen aus:

(1) Peggy schickt Vic die Zufallszahl a, dieser leitet diese Zahl direkt an Clara weiter
(2) Clara schickt Vic das zufällige Bit b, dieses wird sofort an Peggy weitergeleitet
(3) Peggy's Antwort z wird an Clara weitergeleitet
(4) nach mehreren erfolgreichen Runden ist Clara davon überzeugt, dass Vic Peggy ist

Das ist natürlich ein extrem schwerwiegender Fehler. Eine mögliche Lösung wurde von Thomas Beth und Yvo Desmedt vorgeschlagen und in Ansätzen in [**Sch96**, Kapitel 5.2, Seite 131] vorgestellt. Die Lösung arbeitet mit exakten Uhren und versucht so zu verhindern, dass Vic während der Kommunikation mit Peggy noch Zeit hat, mit Clara zu kommunizieren. Wenn jeder Schritt des Protokolls zu genau definierten Zeitpunkten stattfinden muss, dann kann man das Ausnutzen dieser Schwäche verhindern.

4. Nicht-interaktive Zero-Knowledge Verfahren

Bisher war das herausragendste an Zero-Knowledge Beweisen, dass es Beweise waren, die auf *Interaktivität* beruhen. *Wie kann man also interaktive Beweise plötzlich wieder „nicht-interaktiv" machen?* Mit dieser Frage beschäftigt sich dieser Abschnitt.

Wenn Peggy Vic gegenüber ihre Identität nachweist, kann eine weitere Person, Carol zum Beispiel, noch lange nicht überzeugt werden, dass Peggy ein Geheimnis weiß, sie muss dazu selbst

[19] siehe auch Seite 28

mit Peggy kommunizieren. Um diesen Missstand auszugleichen, benötigen wir nicht-interaktive Zero-Knowledge Beweise.

Nicht jedes Protokoll eignet sich dafür, bei einem nicht-interaktiven zero-knowledge Beweis eingesetzt zu werden. Das Fiat-Shamir Verfahren erlaubt es zum Beispiel[20]. Peggy könnte einen solchen Beweis veröffentlichen und so jedem beweisen, dass sie ein Geheimnis kennt, ohne jedoch irgendetwas über das Geheimnis zu verraten. Es ist allerdings ausgesprochen erstaunlich, dass man Daten veröffentlichen kann, die einem beweisen, dass jemand ein Geheimnis besitzt, aber absolut nichts über das Geheimnis an sich aussagen.

Die Kommunikation mit Vic war das entscheidendste, da er dafür gesorgt hat, dass Peggy nicht vorhersehen konnte, was sie als nächstes beweisen muss[21]. Wir müssen also Vic durch eine Folge von *„Zufallszahlen“* ersetzen, die aber Peggy absolut nicht vorhersehen kann. Das nächste Protokoll beschreibt den allgemeinen Ablauf eines nicht-interaktiven Beweises:

(1) Peggy benutzt ihr Geheimnis, und n Zufallszahlen, um das mathematische Problem, welches mit dem Geheimnis verbunden ist, in n isomorphe neue Probleme zu transformieren[22]. Diese neuen Probleme kann man genauso schwer lösen, wie das ursprüngliche Problem.
(2) Peggy legt sich auf die Lösung der n neuen Probleme fest, das kann sie, da sie die Lösung zu dem ursprünglichen isomorphen Problem besitzt.
(3) sie fasst nun alle Festlegungen zu einem Bitvektor zusammen und wendet darauf eine Einweg-Hashfunktion an. Sie speichert nun die ersten n Ausgabebits der Hashfunktion.
(4) Peggy verwendet nun die n generierten Bits fogendermaßen. Für jedes neue Problem i nimmt sie das i-te Bit und
    (a) wenn es gleich 0 ist, dann legt sie einen Isomorphismus zwischen dem alten Problem und dem i-ten Problem offen, oder
    (b) wenn es gleich 1 ist, dann legt sie die in Schritt (2) berechnete Lösung des i-ten Problems offen und zeigt, dass es sich dabei tatsächlich um eine Lösung handelt.
(5) Peggy veröffentlicht alle Festlegungen aus Schritt (2) und die Lösungen aus Schritt (4).
(6) Vic und Carol können nun die Echtheit des Beweises überprüfen, indem sie die Schritte (1) bis (5) einzeln durchgehen und nachsehen, ob Peggy alles korrekt durchgeführt hat.

Das Protokoll funktioniert, da die Einweg-Hashfunktion als ein „ausgewogener Zufallszahlengenerator“ eingesetzt wird. Wenn Peggy betrügen will, müsste sie die durch die Hashfunktion zufällig generierten Bits vorhersagen können – dies kann sie aber nicht. Da die Hashfunktion festlegt, wie Peggy im Schritt (4) antworten muss, ersetzt sie Vic’s Rolle. Kennt Peggy das Geheimnis nicht, kann sie nicht auf beide Fragen in Schritt (4) antworten und der Betrug würde sofort auffallen. Man muss allerdings die Anzahl der durchgeführten Iterationen erhöhen – bei interaktiven Beweisen haben ca. 10 Iterationen genügt. Das reicht bei einem nicht-interaktiven

[20] Beweise dazu kann man in [**DK02**, Kapitel 4.2.5, Seite 75] finden

[21] beispielsweise: den Isomorphismus zwischen H und G_0 oder zwischen H und G_1, vergleiche Abbildung 5 auf Seite 30

Beweis nicht mehr aus, da eine Betrügerin ständig von vorne anfangen kann und somit den Wert der Hashfunktion „nach ihrem Geschmack“ beeinflussen[23].

5. Fazit

Als abschließende Bemerkungen möchte ich noch einmal verdeutlichen, wie wichtig Zero-Knowledge Verfahren in der Kryptographie sind. Viele Autorisierungsverfahren lassen es zu, dass ein Angreifer Informationen über das verwendete Geheimnis erhalten kann, das geht bei Zero-Knowledge Verfahren nicht. Ein weiterer wichtiger Vorteil ist, dass man von Zero-Knowledge Beweisen *„beweisen“* kann, dass sie sicher sind, von anderen Protokollen wird dies nur vermutet.

Wir haben dabei Verfahren kennengelernt, welche die Zero-Knowledge Eigenschaft besitzen. Diese Verfahren beruhen dabei auf mathematisch schwer berechenbaren Problemen, wie zum Beispiel dem Graphenisomorphieproblem oder dem Quadratwurzelziehen in QR_n. Aufgrund des mathematischen Hintergrundes, gibt es *„unendlich“* viele Instanzen dieser Beweise – diese Feststellung ist in dahingehend wichtig, dass sie Zero-Knowledge Beweise nicht nur theoretisch wichtig macht, sondern insbesondere auch in praktischer Hinsicht. Das Fiat-Shamir Verfahren wird zum Beispiel bei Kreditkarten oder auch SmartCards[24] eingesetzt, hier ist es absolut wichtig, dass man auf die Sicherheit der Kunden achtet. Das Unternehmen, welches die SmartCards vertreibt, das kann beispielsweise eine Bank sein, muss unbedingt auf seinen Ruf achten, sonst verliert es Kunden.

Wie man leicht einsieht, kann man Zero-Knowledge Verfahren in vielen Gebieten – sowohl theoretisch, als auch praktisch – einsetzen und in Zukunft wird noch viel auf diesem Gebiet geforscht werden. Insbesondere sollte man sich meiner Meinung nach auf die effektive Vermeidung der Schwächen konzentrieren und auf diesem Wege beweisbar perfekte kryptographische Protokolle kreieren, die in jeder Hinsicht sicher sind.

[23] sie kann so lange probieren, passende Probleme zu finden, bis die Hashfunktion „angenehme“ Werte zurückliefert.

[24] vgl. [**Die93**]

KAPITEL 3

Multiparty Computations und Anonymität (Georg Westenberger)

ZUSAMMENFASSUNG
Gegenstand dieses Kapitels sind zum einen *Multiparty Computations*, d.h. Berechnungen, die von mehreren Parteien unter Erfüllung bestimmter Sicherheitsbedingungen (z.B. Geheimhaltung der Eingabedaten) gemeinsam ausgeführt werden, sowie Secret Sharing Schemes (Verfahren, um Geheimnisse "aufzuteilen"); zum anderen *Anonymität*, d.h. Protokolle, welche die Identität bestimmter Protokollteilnehmer verschleiern. Dies wird an Beispielen wie z.B. "Elektronisches Geld" erläutert.

1. Verwendete Notation

- $A \xrightarrow{m} B$: A schickt die Nachricht m an B.
- $A \xrightarrow{m} *$: A schickt m an alle (Broadcast).
- A :: *Berechnung* : A führt *Berechnung* durch.
- d_A : Entschlüsselungs- oder Validierungsfunktion von A.
- e_A : Verschlüsselungs- oder Signaturfunktion von A.
- $a \oplus b$: XOR bzw. Addition modulo 2, wobei a, b Bits/Bitvektoren sind

2. Einleitung

Bei der Zusammenarbeit mehrerer Parteien stellt sich häufig die Frage: Wem kann ich wieviel anvertrauen? Die Kryptographie ermöglicht Lösungen für folgende Szenarien:

(1) Niemand ist vertrauenswürdig, aber alle halten sich an das Protokoll. Die Teilnehmer wollen kooperieren, ohne dabei direkt ihre Geheimnisse preiszugeben.
(2) Keine einzelne Partei ist vertrauenswürdig, jedoch bestimmte Mengen von Parteien sind es.

Auf dieses Thema geht Abschnitt 3 ein.

Abschnitt 4 befasst sich mit dem Thema Anonymität. Die Motivation für Protokolle, die diese gewährleisten, ist klar, gibt es doch vielerlei Anwendungen wie z.B. Wahlen, elektronisches Geld, anonyme Nachrichten.... Ich werde auf ein Protokoll, das die Anonymität des Senders einer Nachricht garantiert, eingehen (Dining-Cryptographers-Protokoll). Außerdem werde ich 2 Protokolle für elektronische Bezahlung vorstellen.

3. Multiparty Computations

Multiparty Computations sind Protokolle für verteilte Berechnungen, die eine sinnvolle Kooperation mehrerer Parteien ermöglichen, die sich untereinander nicht notwendigerweise vertrauen.

Sie ermöglichen z.B. das gemeinsame Berechnen bestimmter Funktionen, wobei die Eingabedaten der Teilnehmer oder die Zwischenergebnisse geheim bleiben.

3.1. Ein einfaches Beispiel: Durchschnittsgehalt. Alice, Bob und Carol arbeiten alle in derselben Abteilung eines Betriebs. Eines Tages kommen sie auf die Idee, ihr Durchschnittsgehalt zu berechnen, so dass jeder von ihnen sein Gehalt damit vergleichen und evtl. um eine Gehaltserhöhung bitten kann. Keiner von ihnen will jedoch ein Gehalt den anderen beiden preisgeben. Dennoch ist die Berechnung möglich.

Zunächst meldet sich einer der drei freiwillig als Sprecher, z.B. Alice. Sie denkt sich eine Zufallszahl r aus, addiert sie zu ihrem Gehalt und flüstert die Summe Bob ins Ohr (hier durch Verschlüsselung mit Bobs öffentlichem Schlüssel dargestellt):

$A :: s_A := e_B(a + r)$
$A \xrightarrow{s_A} B$

Die beiden anderen addieren ihr Gehalt und teilen die Summen vertraulich der jeweils nächsten Person mit:

$B :: s_{AB} := e_C(d_B(s_A) + b)$
$B \xrightarrow{s_{AB}} C$
$C :: s_{ABC} := e_A(d_C(s_{AB}) + c)$
$C \xrightarrow{s_{ABC}} A$

Alice kann nun ihre Zufallszahl von $d_A(s_{ABC})$ abziehen und erhält die Gehaltssumme, und nach Division durch 3 das Durchschnittsgehalt. Das Flüstern/die Verschlüsselung ist notwendig, da sonst z.B. Carol zusätzlich zu $r + a + b$ auch $r + a$ erfahren könnte und somit Bobs Gehalt wüsste.

3.2. Secret Sharing Schemes. Secret Sharing Schemes sind Verfahren, um ein Geheimnis auf intelligente Weise so zu zerteilen, dass dieses nur aus bestimmten Konfigurationen der Stücke wiederhergestellt werden kann. Diese Verfahren werden z.B. angewandt, um Verantwortung auf mehrere Parteien $A_1, \ldots, A_n$ zu verteilen, wenn keiner der A_i allein vertrauenswürdig genug ist, das Geheimnis für sich zu besitzen.

Im allgemeinen sind nicht alle Stücke notwendig, um das Geheimnis wiederherzustellen; deswegen kann ein Teilgeheimnis verlorengehen, ohne dass das Geheimnis verlorengeht.

Beispiele für mögliche Geheimnisse sind Codes für kritische Operationen wie das Öffnen eines Banksafes oder der Abschuss von Atomraketen; hierbei wird man jedoch nur einen Hashwert des Geheimnisses im Gerät speichern, um eine Rekonstruktion durch Analyse des Geräts unmöglich zu machen.

Am einfachsten aufgebaut sind sogenannte *Threshold-Verfahren*. Ein (k,n)-Threshold-Verfahren zerstückelt ein Geheimnis in n Teile, von denen je k ausreichen, um das Geheimnis widerzuerlangen. Es ist also kein Stück “wertvoller” als ein anderes; Sharing Schemes mit dieser Eigenschaft nennt man deswegen auch *demokratisch*. Im folgenden werde ich auf 2 Beispiele für Threshold-Verfahren eingehen: Das *Secret Splitting* und das *Shamir-Verfahren*.

Secret Splitting. Bei diesem sehr einfachen Verfahren sind alle Teilgeheimnisse für den Rekonstruktionsvorgang notwendig. Es handelt sich also um ein (n, n)-Thresholdverfahren.

Das zu zerteilende Geheimnis wird zunächst als Bitvektor $s \in \{0, 1\}^k$ kodiert. Nun bestimmt man $n - 1$ Bitvektoren $v_1, \ldots, v_{n-1}$ mit gleicher Länge wie s. Den letzten Vektor wählen wir als $v_n = \sum_{i=1}^{n-1} v_i + s$. $v1, \ldots, v_n$ werden an die Protokollteilnehemer verteilt.

Da Addition und Subtraktion modulo 2 identisch sind, können die Geheimnisträger nun s rekonstruieren, indem sie $v_1, \ldots, v_n$ aufsummieren. Bei diesem Verfahren ist die Kenntnis von weniger als n Teilgeheimnissen völlig nutzlos, da vor Addition des letzten Teils jeder beliebiger Bitvektor als Summe in Frage kommt.

Interpolations-Verfahren nach Shamir. Das Geheimnis wird hier üblicherweise als Element eines großen endlichen Körpers, z.B. eines Restklassenkörpers modulo p dargestellt. Die Aufteilung geschieht wie folgt:

Es wird ein Polynom $P(x) := \sum_{i=1}^{k-1} a_i x^i + s$ erzeugt, wobei s das Geheimnis ist und die a_i zufällig ausgewählte Elemente des Körpers. Nun werden n weitere paarweise verschiedene Elemente $b_1, \ldots, b_n$ zufällig ausgewählt. Jetzt werden an die n Parteien die Teilgeheimnisse $(b_i, P(b_i))$ verteilt.

Je k Parteien $A_{m_1}, \ldots, A_{m_k}$ können nun das Polynom P aus ihren b_{m_i} und $P(b_{m_i})$ mit Hilfe der Lagrangeschen Interpolation eindeutig rekonstruieren. Zunächst erzeugen sie k Hilfspolynome

$$q_i(x) := \prod_{\substack{1 \le j \le k \\ j \ne i}} \frac{x - b_{m_j}}{b_{m_i} - b_{m_j}}, i = 1, \ldots, k$$

Aus diesen lässt sich das ursprüngliche Polynom leicht erzeugen:

$$P(x) = \sum_{i=1}^{k} q_i(x) P(b_{m_i})$$

Durch Einsetzen von 0 in das Polynom erhalten sie das Geheimnis $P(0) = s$.

Nichtdemokratische Verfahren.

Wenn die betreffenden Parteien nicht alle gleich vertrauenswürdig sind, also nicht jedes Teilgeheimnis gleichen Wert haben soll, wird man kein demokratisches Verfahren anwenden wollen. Eine einfache Lösung besteht z.B. darin, von den durch ein demokratisches Verfahren erzeugten Teilgeheimnissen unterschiedlich viele an die beteiligten Parteien auszugeben – je vertrauenswürdiger eine Partei ist, um so mehr Teile erhält sie.

3.3. Skatspielen per Telefon. Anders als physische Gegenstände sind in Rechnern gespeicherte Informationen beliebig kopier- und reproduzierbar. In der physischen Welt funktionierende Verfahren, die eben die Nichtreproduzierbarkeit gewisser Gegenstände (z.B. Schlüssel) verlangen, funktionieren also in der virtuellen Welt nicht ohne weiteres. Ein Beispiel hierfür ist das Skatspiel: Es ist hier zwar auch beim physischen Kartenspielen möglich zu betrügen, der Aufwand ist jedoch relativ hoch. Es ist viel einfacher, die Information “Herz-As” durch einen

Kommunikationskanal zu senden, als das physische Pendant aus dem Ärmel zu schütteln. In der Praxis wird man eher einer vertrauenswürdigen Partei die Verwaltung der Spieldaten und Überprüfung der Zulässigkeit der jeweiligen Züge überlassen; hier soll ein in [**BSW98**] dargelegtes Verfahren vorgestellt werden, mit dem eine solche Betrugsüberprüfung auch ohne weitere Partei möglich ist.

Es besteht aus den folgenden vier Phasen:

(1) Mischen der Karten
(2) Austeilen der Karten
(3) Das eigentliche Spiel
(4) Betrugsprüfung

Das Verfahren verwendet Shamirs No-Key-Verfahren, denkbar sind jedoch auch andere Implementierungen, die kommutative Verschlüsselungsmethoden benutzen. Zunächst einigt sich unsere Skatrunde, bestehend aus Alice, Bob und Carol (A, B, und C) auf einen großen Primzahlmodulus p, in dem fortan gerechnet werden soll. Sie berechnen je zwei Zahlen $a, a'; b, b'; c, c' \in \mathbb{F}_p$ mit
$aa' \equiv bb' \equiv cc' \equiv 1\,(mod\, p-1)$).
Wir definieren nun $e_A(x) := x^a$ und $d_A(x) := x^{a'}$ (entsprechend für B und C) als Schlüsselfunktionen. Die Verkettung dieser Funktionen ist kommutativ, da es sich um Exponentiationen handelt; ebenso gilt $e_P \circ d_P = id$ für jeden Spieler P, wobei id die identische Abbildung auf $\mathbb{F}_p$ ist.

Die Werte der Karten werden als Zahlen modulo p dargestellt. Man sollte hier bei Verwendung des No-Key-Verfahrens keine quadratischen Reste modulo p verwenden, denn wenn $y \equiv x^2 (mod\, p)$, so ist $y^k \equiv (x^k)^2 (mod\, p)$; sieht ein Spieler eine verschlüsselte Karte, die kein quadratischer Rest ist, weiß er, dass der Kartenwert kein solcher ist, und gewinnt so evtl. einen Vorteil. Der Kartenstapel wird codiert als Funktion $K : \{1, \ldots, 32\} \to \mathbb{F}_p$, wobei K auf einem Computer als Array dargestellt wird.

3.3.1. *Mischen:* Alice wählt eine Permutation $\alpha : \{1, \ldots, 32\} \to \{1, \ldots, 32\}$, mischt K damit durch, verschlüsselt die Karten mit e_A und schickt sie an Bob:

$A :: K' := e_A \circ K \circ \alpha$
$A \xrightarrow{K'} B$

Bob und Carol setzen den Vorgang mit ihren eigenen Schlüsseln und Permutationen fort:

$B :: K'' := e_B \circ K' \circ \beta$
$B \xrightarrow{K''} C$
$C :: K''' := e_C \circ K'' \circ \gamma$
$C \xrightarrow{K'''} A$
$A \xrightarrow{K'''} *$

Natürlich werden auch hier in Wirklichkeit nur Arrays von Zahlen übertragen und die Permutationen von den jeweiligen Spielern als Umsortierung des Arrays durchgeführt.

3.3.2. *Austeilen:* Nun, da der gemischte und verdeckte "Kartenstapel" allen bekannt ist, kann das Austeilen beginnen. Wir definieren $\pi := \alpha \circ \beta \circ \gamma$.Will z.B. Alice eine Karte $K'''(i)$ verdeckt an Carol austeilen, so entfernen jeweils Alice, Bob und Carol in dieser Reihenfolge ihre "Schlösser":

$A :: k := K'''(i) = e_C(e_B(e_A(K(\pi(i)))))$
$A :: k' := d_A(k) = e_C(e_B(K(\pi(i))))$
$A \xrightarrow{k'} B$
$B :: k'' := d_B(k') = e_C(K(\pi(i)))$
$B \xrightarrow{k''} C$
$C :: k''' := d_C(k'') = K(\pi(i))$

3.3.3. *Spiel:* Nachdem alle Karten in dieser Reihenfolge ausgeteilt wurden, kann das eigentliche Spiel beginnen: Die Spieler spielen ihre Karten aus, indem sie einfach den Namen der Karte über die Telefonleitung bekannt geben.

3.3.4. *Betrugsprüfung:* Zwar kann ein Spieler versuchen, eine Karte auszuspielen, die er nicht hat; dieser Betrug wird jedoch in der nächsten Phase aufgedeckt: Hier geben nun alle Spieler ihre Schlüssel (e_X, d_X) und Permutationen (α, β, γ) bekannt und prüfen den Spielverlauf (den mindestens einer der Spieler aufgezeichnet hat) nach.

Eine Abwandlung des Verfahrens kann zur Schlüsselverteilung eingesetzt werden, wenn die Benutzer ihre Schlüssel nicht selbst erzeugen dürfen – die Schlüssel entsprechen dann den Karten. Näheres hierzu in [**Sch96**], Abschnitt 4.11.

4. Anonymität

In einer Kommunikationsbeziehung sind nicht nur die Inhalte von Nachrichten von Interesse, sondern auch deren Sender und Empfänger; man kann bereits die Tatsache, *dass* Bob eine Nachricht an Alice (und nicht etwa an Carol) schickt, als Kommunikation auffassen, ganz unabhängig davon, welchen Inhalt die Nachricht hat. Auch aus dem Zeitpunkt der Nachricht lassen sich möglicherweise Schlüsse ziehen. Deswegen liegt es nicht fern, auch diese Art von Informationen mit den Mitteln der Kryptographie geheimzuhalten.

Anwendungen gibt es viele: Ein Spion, der eine Nachricht an seinen Auftraggeber schickt, ist daran interessiert,nicht zurückverfolgt zu werden, selbst wenn die Nachricht abgefangen wird; ein Wähler legt Wert darauf, dass man ihn seiner Stimme bei der Bundestagswahl nicht zuordnen kann. In diesem Abschnitt werden einige Protokolle, die Anonymität gewährleisten, dargestellt.

4.1. Dining Cryptographers. Das Dining-Cryptographers-Protokoll wurde 1988 von David Chaum entwickelt. Es ermöglicht (höchstens) einem von mehreren Teilnehmern, eine binär codierte Nachricht an alle Teilnehmer zu schicken – ohne, dass einer der anderen Teilnehmer den Absender ausfindig machen kann. Es ist nach dem folgenden Beispiel benannt, anhand dessen das Protokoll meist erklärt wird:

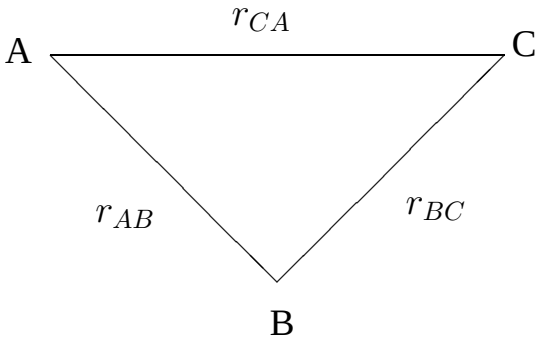
Drei Kryptographen (Alice, Bob, Carol) treffen sich in einem Lokal zum Essen. Als es Zeit wird, zu bezahlen, teilt ihnen der Kellner mit, es sei vereinbart worden, dass entweder der Arbeitgeber der Kryptographen (der BND), oder einer der Kryptographen das Essen bezahlen wolle; er

selbst wisse jedoch nicht, welche von diesen beiden Möglichkeiten zutreffe. Die Kryptographen wollen nun herausfinden, ob der BND oder einer von ihnen bezahlen will, ohne eventuell den spendablen Kryptographen unter ihnen in Verlegenheit zu bringen.

Zunächst einigen sich je zwei der Kryptographen auf ein Bit, das dem jeweiligen dritten Kryptographen verborgen bleibt, z.B. indem sie hinter ihren Speisekarten Münzen werfen:

$A, B :: r_{AB} := random(\{0, 1\})$
$B, C :: r_{BC} := random(\{0, 1\})$
$C, A :: r_{CA} := random(\{0, 1\})$

Wir können die Kryptographen und ihre r_{XY} als Graph darstellen, wobei die Knoten die Teilnehmer und die Kanten die Geheimnisse zwischen den von den durch sie verbundenen Knoten symbolisierten Teilnehmern darstellen:



Jeder der Kryptographen addiert nun die Bits, die er mit seinen Tischnachbarn vereinbart hat, modulo 2:

$A :: s_A := r_{AB} \oplus r_{CA}$
$B :: s_B := r_{BC} \oplus r_{AB}$
$C :: s_C := r_{BC} \oplus r_{CA}$

Jeder der Kryptographen gibt nun sein s_X bekannt – falls jedoch einer von ihnen bezahlen möchte, gibt er das Komplement seines s_X bekannt. Die Kryptographen addieren jetzt ihre bekanntgegebenen Werte modulo 2.
Falls nun der BND bezahlt, so ist die Summe:

$2s_A \oplus 2s_B \oplus 2s_C = 0$

Bezahlt einer der Kryptographen, ist die Summe:

$2s_A \oplus 2s_B \oplus 2s_C \oplus 1 = 1$

Es wird also genau 1 Bit an Information anonym übertragen, denn will z.B. Alice feststellen, ob Bob über die Summe von r_{AB}, r_{BC} "gelogen" hat, fehlt ihr dazu r_{BC}.

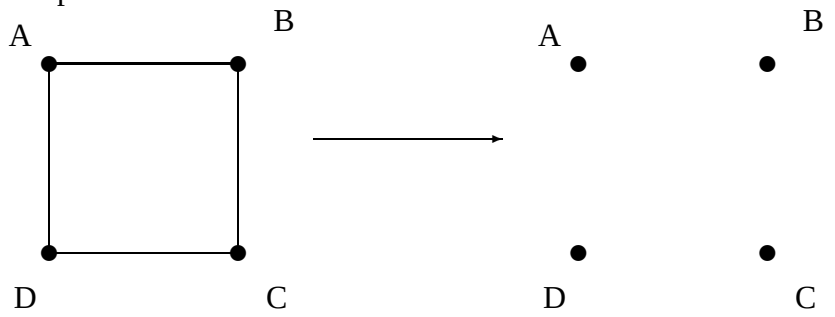
Es gibt nun verschiedene Möglichkeiten, diese Protokoll zu verallgemeinern. Statt z.B. einzelne Bits als Geheimnisse und Nachrichten zu verwenden, ist dies problemlos mit Bitvektoren beliebiger Länge möglich – das hat darüberhinaus den Vorteil, dass man Redundanz (in Form

von fehlererkennenden Codes) in die Nachrichten einbauen kann; so lassen sich z.B. Übertragungsfehler oder das versehentliche gleichzeitige Senden von Nachrichten entdecken.

Auch die Erweiterung des Protokolls auf mehrere Teilnehmer ist möglich. Dann gibt es mehrere Möglichkeiten, einen zusammenhängenden Graphen wie oben (ein sogenanntes *DC-Netz*) zu erstellen. Wichtig ist hierbei, dass jeder Knoten mindestens zwei Nachbarn besitzt, denn sonst wüsste sein Nachbar sofort um eine von ihm gesendete Nachricht. Am einfachsten ist der Zusammenschluss der Teilnehmer in einem Kreis.

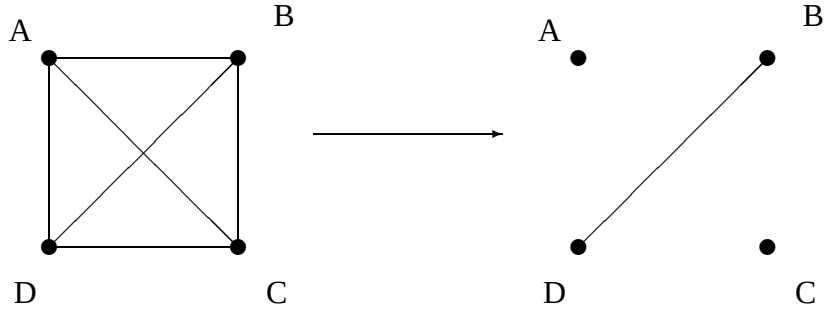
Die Wahl der Netztopologie hat direkten Einfluss auf die Zurückverfolgbarkeit. Chaum zeigte [**Cha88**], dass für einen Beobachter, dem alle s_X und eine Menge S von Geheimnissen ("Kanten") bekannt sind, eine Nachricht sich bis zu den Komponenten des Graphen (und nicht weiter) zurückverfolgen lässt, die durch Hinwegnahme der Kanten in S aus dem ursprünglichen Graphen entstehen.

Beispiel 1:



Nehmen wir an, dass Alice und Carol ihre Schlüssel teilen. Ihnen sind dann alle Schlüssel des Netzes bekannt, und sie können von Bob und Dave gesendete Nachrichten unterscheiden.

Beispiel 2:



Falls Alice und Carol in diesem System ihre Schlüssel teilen, sind ihnen alle Schlüssel bis auf r_{BD} bekannt, d.h. Bob und Dave bleiben in einer Komponente; Alice und Bob können also Nachrichten von Bob und Dave nicht unterscheiden.

Probleme gibt es, wenn mehrere der Teilnehmer anonym Nachrichten senden wollen. Hier geht man folgendermaßen vor:

- Das Protokoll wird wiederholt durchgeführt (natürlich müssen in jeder Runde neue Schlüssel vereinbart werden).
- Die Nachrichten werden als Bitvektoren mit einem fehlererkennenden Code codiert.
- Wenn zwei oder mehr Teilnehmer Nachrichten geschickt und der Fehler erkannt wird, wartet jeder vom ihnen eine zufällige Anzahl von Runden und sendet seine Nachricht erneut (Dies entspricht dem CSMA/CD-Protokoll (Carrier Sense Multiple Access/Collision Detect), das beim *Ethernet* verwandt wird).

4.2. Elektronisches Geld. Wie auch beim Skatspielen (s.o.) liegt das Problem beim bargeldlosen Bezahlen darin, dass sich Information beliebig kopieren lässt. Geld sollte sich jedoch nicht beliebig kopieren lassen. Herkömmliche Methoden, wie die Überweisung oder Bezahlung per Kreditkarte, haben jedoch einen gravierenden Nachteil: Um Betrug zu vermeiden, werden alle Transaktionen über Zentralrechner der jeweiligen Banken abgewickelt, welche nachprüfen, ob die Transaktion gültig ist oder nicht. Das ist nicht nur aufwendig und ineffizient, die Banken erfahren auch, wer wann und wo bezahlt. Da Banken nicht für ihre Uneigennützigkeit bekannt sind, wäre es möglich, dass sie diese Information ausnutzen. Die Methoden der Kryptographie erlauben es uns, Protokolle zu konstruieren, die die Anonymität von Zahlungsvorgängen gewährleisten.

Blinde Signatur. Elektronische Zahlungsverfahren benötigen oft die Methode der *digitalen Blindunterschrift*; sie dient als "Echtheitszertifikat" (bei Bargeld entsprechen dieser z.B. das Wasserzeichen und der Metallfaden, die von Fälschern nur schlecht reproduziert werden können), und verhindert, dass die Bank später den elektronischen Zahlungseinheiten die sie ausgebenden Personen zuordnen kann. Von der gewöhnlichen digitalen Signatur unterscheidet sich die blinde Signatur dadurch, dass der Unterzeichner eine verschlüsselte Nachricht signiert und an den Sender zurückschickt, welcher sie entschlüsseln kann, wobei die Unterschrift erhalten bleibt.

Protokoll 1: Elektronische Schecks.

- Alice erstellt k Schecks. Dabei besteht jeder Scheck aus:
  - dem Geldbetrag
  - einer (sehr großen) Eindeutigkeitsnummer, die den Scheck eindeutig identifiziert
  - einer "Identitätsfolge" von n Paaren von Hashwerten verschlüsselter Bitvektoren $(f(L_1), f(R_1)), \ldots, (f(L_n), f(R_n))$, wobei f eine kryptographische Hashfunktion ist und für jedes i gilt: $L_i \oplus R_i$ ergibt die Identität von Alice (Dies ist z.B. der Fall bei $L_i = X, R_i = X \oplus L_i$, wobei I Alices Identität und X ein zufällig erzeugter Bitvektor ist). Hierdurch legt sie sich auf die L_i, R_i fest (Commitment).

Geldbetrag	
Eindeutigkeitsnummer	
f(L_1) f(L_2) • • • f(L_n)	f(R_1) f(R_2) • • • f(R_n)

- sie schickt alle Schecks, verschlüsselt mit Blindfaktoren, an die Bank.
- Die Bank sucht sich einen der Schecks aus und bittet Alice um die Blindfaktoren und die L_i, R_i aller anderen $n-1$ Schecks.
- Alice legt diese offen und die Bank prüft, ob die $n-1$ Schecks gültig sind, d.h. ob sie denselben Geldbetrag aufweisen und ob die Identitätspaare, auf die sich Alice festgelegt hat, tatsächlich aus ihren Identitätsdaten berechnet wurde (Dieses Verfahren nennt sich "Divide and Select" – eine verschärfte Form der Stichprobe).
- Die Bank signiert den übriggebliebenen Scheck blind, bucht den entsprechenden Betrag von Alices Konto ab und schickt ihn an Alice zurück, welche ihren Blindfaktor entfernt.
- Alice will nun in Bobs Online-Versand mit dem Scheck bezahlen.
- Bob bittet Alice, jeweils ein (zufällig ausgewähltes) Element jedes Identitätspaares offenzulegen. Alice führt dies durch.
- Bob löst den Scheck, zusammen mit den entschlüsselten Hälften der Identitätspaare, bei der Bank ein.
- Die Bank prüft nach, ob die Eindeutigkeitsnummer der Nummer eines früher eingelösten Schecks gleicht.
- Ist dies nicht der Fall, so ist der Scheck "neu", es liegt kein Betrug vor – die Bank speichert die Eindeutigkeitsnummer in ihrer Datenbank und schreibt Bob den entsprechenden Betrag gut.
- Wird die Nummer gefunden, so gibt es zwei Möglichkeiten:
  (1) Auf beiden Schecks sind die gleichen Elemente der Identitätspaare entschlüsselt. Da bei großem k die Wahrscheinlichkeit, dass 2 Händler dieselben Elemente auswählen, sehr klein ist, hat der Händler versucht, den Scheck erneut einzulösen.
  (2) Auf den Schecks wurden bei mindestens einem Identitätspaar $(f(L_c), f(R_c))$ beide Elemente offengelegt. Alice hat also versucht, mit dem Scheck bei verschiedenen

Händlern zu bezahlen. Die Bank erfährt Alices Identität, indem sie $L_c \oplus R_c$ berechnet.

Dieses Verfahren hat den Vorteil, dass Betrüger leicht identifiziert werden können. Damit ist die Sicherheit für alle Beteiligten gewährleistet. Ein Problem des Verfahrens ist der Rechenaufwand, welcher umso größer wird, je höher man n und k wählt. Hier bietet sich z.B. an, die Werte von n und k abhängig vom gewünschten Geldbetrag zu machen – je höher dieser ist, umso sicherer soll das Verfahren sein. Ein weiterer Nachteil ist die Nichttransferierbarkeit der Schecks, durch die gerade die Zurückverfolgbarkeit von Betrügern erkauft wurde.

Protokoll 2: Elektronische Münzen. Ein anderes Verfahren für elektronisches Geld sind *elektronische Münzen*. Der Name rührt daher, dass das Verfahren darauf aufbaut, dass nur bestimmte Geldbeträge als Zahlungseinheiten zugelassen sind. Es ist folgendermaßen aufgebaut:

- Alice will eine 10-Euro-Münze erstellen.
- Sie erstellt eine große Zufallszahl, die ein von der Bank vorgegebenes Redundanzschema hat (z.B. ein Palindrom wie 175767571).
- Sie schickt diese Zahl, mit einem Blindfaktor verschlüsselt, an die Bank mit der Bitte, eine 10-Euro-Münze zu erstellen.
- Die Bank besitzt verschiedene Signierschlüssel für jeden möglichen Münzwert. Sie wählt den 10-Euro-Schlüssel aus, signiert Alices Zahl blind und schickt sie an Alice zurück.
- Alice entfernt ihren Blindfaktor und bezahlt in Bobs Geschäft mit der Münze.
- Bob entschlüsselt mit dem öffentlich bekannten 10-Euro-Schlüssel die Münze. Wenn das Ergebnis in das vorgegebene Schema passt, weiß er, dass die Münze echt ist.
- Bob schickt die Münze an die Bank.
- Die Bank prüft, ob die für die Münze verwendete Zufallszahl mit der einer bereits eingelösten Münze übereinstimmt. Ist dies nicht der Fall, schreibt sie Bob den entsprechenden Betrag gut.

Das Verfahren hat, anders als das vorhergehende, den Nachteil, dass ein mehrmaliges Einlösen der Münze erkannt, der Schuldige aber nicht identifiziert werden kann. Der Händler muss also die Münzen sofort einlösen, wenn er nicht betrogen werden will – was denselben technischen Aufwand wie das Bezahlen mit Kreditkarten erfordert. Immerhin ist hier jedoch ein Transferieren der Münzen an andere möglich.

Da der Betrag der Münze nicht von den von Alice gewählten Parametern abhängt, kann Alice den Betrag der Münze nicht fälschen; das “divide and select”-Verfahren aus dem vorigen Verfahren ist nicht nötig, der Aufwand ist weitaus geringer.

5. Fazit

Anonymität, wie sie z.B. die im zweiten Teil vorgestellten Protokolle gewährleisten, werden in den Zeiten des Internet und besonders des e-Commerce wichtiger werden; das Bewusstsein dafür, dass wir in vielerlei Hinsicht noch gläserne Benutzer sind, wird (hoffentlich) zunehmen und zur Verbreitung solcher Protokolle beitragen.

Weniger für den gewöhnlichen Nutzer interessant sind die im ersten Teil vorgestellten Multiparty-Computation-Verfahren, welche jedoch in Bereichen mit sehr hohen Sicherheitsanforderungen ihre Anwendung finden.

KAPITEL 4

Interaktive Beweise (Timo Neumann)

ZUSAMMENFASSUNG (ABSTRACT)
In dieser Ausarbeitung wird zunächst die Definition eines interaktiven Beweissystems motiviert, bevor formal gefasst wird, was ein interaktives Beweissystem ist.

Der Haupteil der Abhandlung beschäftigt sich dann mit der durch interaktive Beweissysteme beschreibbaren Klasse an Sprachen $\mathbb{IP}$. Dabei steht das Resultat von Shamir $\mathbb{IP} = \mathbb{PSPACE}$ im Mittelpunkt, das eine vollständige Charakterisation dieser Klasse liefert. Insbesondere wird dabei die algebraischen Methodik betont [1], die an einem ausführlichen Beispiel erläutert wird.

Im letzten Teil wird nach anfänglicher Einführung des Zero-Knowledge Begriffs[2] auf die Verbindung zwischen interaktiven Beweisen und interaktiven Zero-Knowledge Beweisen eingegangen.

1. Das interaktive Beweissystem

1.1. Einleitung: Zurück in die Antike? Was ist ein Beweis? Auf diese Frage gibt es in der Logik eine klare Antwort:

DEFINITION 1. Sei Λ ein Alphabet, $\mathcal{F} \subset \Lambda^*$ eine Menge (wohldefinierter) Formeln.
Gegeben sei ein Kalkül $\mathcal{K} = \mathcal{K}(Ax, R)$, wobei Ax $\subset \mathcal{F}$, R $\subset \mathcal{F}^*$ X $\mathcal{F}$.
Weiterhin sei $\Sigma \subset \mathcal{F}$, A $\in \mathcal{F}$ vorgelegt.
Eine Folge von Formeln $B_0, B_1, \ldots, B_n$ ist ein **Beweis** für $\Sigma \vdash A$, falls

(*) $A \equiv B_n$ und
(**) für alle $0 \leq i \leq n$ gilt:
$B_i \in Ax \cup \Sigma$ oder es gibt $i_1, \ldots, i_l < i$ mit $\frac{B_{i_1}, \ldots, B_{i_l}}{B_i} \in$ R.

Diese Definition ermöglicht es insbesondere, den Beweisbegriff mit einem Verifikationsalgorithmuses zu verknüpfen. Nach Eingabe einer *Aussage x* sowie eines *Beweises y*, überprüft dieser syntaktische Korrektheit, d.h. untersucht, ob die Forderungen (*) und (**)gelten. Ist dies der Fall akzeptiert der Algorithmus ansonsten verwirft er.

Damit ist eine beweisbaren Aussagen x in einem gegebenen Kalkül mit gegebenem Verifikationsalgorithmus dadurch charakterisiert, dass es ein y (i.e. Beweis) gibt, so dass eben dieser Algorithmus die Eingabe (x, y) akzeptiert.

[1] ausführliches hierzu fi ndet man in [**LFKN92**]
[2] siehe dazu auch Kapitel 2

Intuitiv existiert ein weiterer Begriff des Beweisens. Dieser ist weniger an die formale Korrektheit geknüpft, als vielmehr an die Idee des Beweises als überzeugendes Argument. In einem Gerichtsprozess mit Geschworenen versuchen Ankläger und Verteidiger beispielsweise, durch vorbringen verschiedener Argumente ihren Standpunkt als den richtigen darzustellen. Die Aufgabe des Anwalts ist es dabei, Zweifel der Geschworenen an der Schuld des Angeklagten auszuräumen, während der Verteidiger alles unternimmt, um neue Zweifel zu wecken.

Die Idee, einen Beweis als Dialog darzustellen, stammt aus der Antike. Man denke an einen Lehrer, der versucht, eine von ihm aufgestellte Behauptung gegen die Zweifel der Schüler verteidigen. Die Schüler werden die Behauptung als wahr anerkennen, wenn alle ihre Bedenken durch den Lehrer entkräftet werden können.

An dieser Stelle sei auf zwei wichtige Aspekte dieser Art des Beweisens hingewiesen. Zum einen lebt ein solcher Beweis von der Interaktion. Das Beispiel *Lehrer-Schüler* macht dies sehr gut deutlich: Woher soll der Lehrer wissen, welche Zweifel der Schüler hat, welche Gedankengänge ihm noch nicht klar bzw. einsichtig sind. Es ist nötig, dass beide Parteien miteinander kommunizieren. Einen solchen Beweis deshalb als interaktiv zu bezeichnen ist naheliegend. Zum zweiten stellt sich für den Schüler die Frage, ob er 'genug gebohrt' hat. Sind alle seine Fragen zu seiner Zufriedenheit beantwortet worden, muss dies noch nicht bedeuten, dass die Behauptung des Lehrers wirklich korrekt ist. Ein interaktiver Beweis muss nicht unbedingt eine Garantie für die Korrektheit der 'bewiesenen' Aussage liefern.

Es stellt sich die Frage, ob diese Art des Verständnisse des Beweisbegriffs neue Möglichkeiten eröffnet. Bevor jedoch dieser Frage nachgegangen werden soll, wird zunächst an einem einfachen Beispiel nochmals die Idee dieser Art des Beweisens dargestellt.

1.2. Der Grenzwertbegriff als Beispiel. Im folgenden soll der mathematische Grenzwertbegriff zur Konstruktion eines ersten Beispiel eines *interaktiven Beweises* herangezogen werden. Man vergleiche dazu [**BSW98**].
Zunächst zur Erinnerung die Definition des Grenzwertbegriffes:

DEFINITION 2. Eine Folge $(a_n)_{n\in\mathbb{N}}$ hat den **Grenzwert** a, $\lim_{n\to\infty} a_n = a$, wenn gilt:

$$\forall\varepsilon > 0 \exists\eta \in \mathbb{N}, \text{ so dass } \forall n > \eta \text{ gilt: } \mid a_n - a \mid \leq \varepsilon$$

Wie könnte diese Definition in ein interaktives Beweissystem transformiert werden?

Dazu soll konkret die Folge $(2^{-n})_{n\in\mathbb{N}}$ betrachtetet werden. Man könnte sich leicht folgenden Dialog zwischen einem Lehrer und einem ungläubigen Schüler vorstellen, in dem der Schüler die Rolle der beiden Allquantoren einnimmt, während der Lehrer die Rolle des Existenzquantors spielt:

▷ *Behauptung des Lehrers:* $\lim_{n\to\infty} 2^{-n} = 0$

▷ *Schüler* wählt ein $\varepsilon > 0$, zum Beispiel $\varepsilon = \frac{1}{10}$

▷ *Lehrer* bestimmt ein passendes η, also etwa $\eta = 4$

▷ *Schüler* wählt seinerseits ein $n > \eta$, sei dies in diesem Beispiel $n = 123$

Nun kann der Schüler nachrechnen, ob gilt

$$\mid a_n - a \mid \leq \varepsilon \tag{1}$$

Ist dies nicht der Fall, wird der Schüler die Behauptung des Lehrers sofort verwerfen. Interessanter ist die Frage, wie der Schüler reagiert, wenn, wie dies oben der Fall ist, (1) erfüllt ist. Ist der Schüler dann überzeugt? Höchstwahrscheinlich noch nicht. Der Schüler könnte durch Zufall 'ungünstige' Zahlen gewählt haben, durch die die Behauptung des Lehrers nicht zu widerlegen ist. Es ist deshalb anzunehmen, dass beide das obige Protokoll mehrfach durchlaufen. Trotzdem bleibt die Frage offen, wann der Schüler überzeugt ist. Eine absolute Sicherheit kann es für ihn nicht geben, da er nicht alle möglichen Zahlen ausprobieren kann. Er muss demnach entweder irgendwann verwerfen oder sich damit zufrieden geben, dass ein Rest Unsicherheit zurückbleibt. Nach diesem einführenden Beispiel soll nun eine formale Definition eines interaktiven Beweissystems folgen.

1.3. Die Definition des interaktiven Beweissystems.

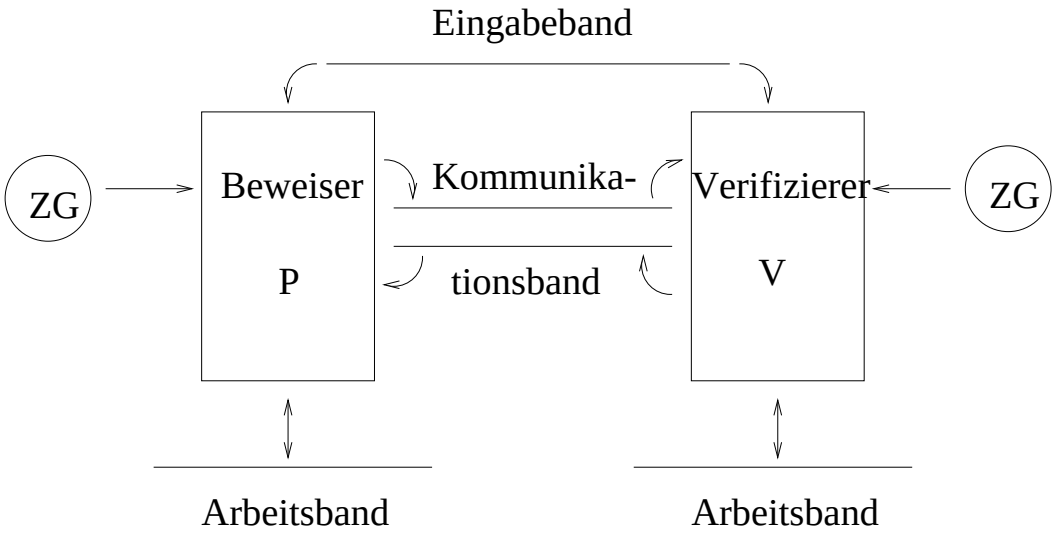
DEFINITION 3. Ein **interaktives Beweissystem (P↔V)** besteht aus einem Paar probabilistischer Turingmaschinen P und V mit einem gemeinsamen Alphabet Σ .
P und V besitzen ausgezeichnete Zustände *Start* und *Warten*. Zusätzlich besitzt V zwei Endzustände *Akzeptieren* und *Verwerfen*.
P und V arbeiten auf verschiedenen Bändern:

- ▷ P und V haben ein *gemeinsamen Eingabeband*.
- ▷ P und V greifen auf je ein *privates Arbeitsband* sowie einen *privaten, idealen Zufallszahlengenerator* zu.
- ▷ P und V haben ein *gemeinsames Kommunikationsband*.
- ▷ V ist *polynomial (in der Länge der Eingabe) zeitbeschränkt*.
- ▷ P ist zeitlich unbeschränkt.

$(P \leftrightarrow V)$ ist ein interaktives Beweissystem für eine Sprache $\mathcal{L}$, falls für ein $0 \leq \varepsilon < \frac{1}{2}$ gilt:

(1) *Vollständigkeit*: $x \in \mathcal{L} \Rightarrow Pr[(P \leftrightarrow V)(x)$ akzeptiert $] \geq 1 - \varepsilon$
(2) *Korrektheit*: $x \notin \mathcal{L} \Rightarrow Pr[(P \leftrightarrow V)(x)$ akzeptiert $] \leq \varepsilon \forall P$

Bemerkung:
Der Ablauf eines interaktiven Beweises erfolgt in Runden. In jeder Runde ist jeweils eine der Turing-Maschinen in einem idle-State, während die andere ihre Berechnungen ausführt. Jede Runde endet damit, dass eine Turing-Maschine der anderen eine Nachricht schickt und selbst in den idle-State geht, während die andere aufwacht.
Die Berechnung endet damit, dass V in einen der Endzustände wechselt.



ZG: Zufallszahlengenerator

ABBILDUNG 1. Skizze eines Interaktiven Beweissystems

2. Die Klasse IP

2.1. Die Definition von IP.

DEFINITION 4. Die **Klasse** $\mathbb{IP}$ besteht aus allen Sprachen, die ein interaktives Beweissystem besitzen.

Genauer:

Bei gegebener Funktion m*:* $\mathbb{N} \to \mathbb{N}$ *besteht die Komplexitätsklasse* $\mathbb{IP}(m(.))$ *aus den Sprachen, die ein interaktives Beweissystem besitzen, in dem die beiden Parteien bei Eingabe* x *höchstens* $m(\mid x \mid)$ *Nachrichten senden.*
Dabei ist

$$\mathbb{IP} = \mathbb{IP}\textit{(poly)}$$

Nachdem die grundlegenden Definitionen vorgestellt worden sind, ist es an der Zeit, ein konkretes Beispiel für ein korrektes interaktives Protokoll anzugeben:

Graphen-Nonisomorphismus: ▷ *Eingabe:* zwei Graphen $\mathcal{G}_0$(V,E_0) und $\mathcal{G}_1$(V,E_1)
- ▷ *Beh:* Die Graphen sind nichtisomorph.
- ▷ Der Verifizierer wählt zufällig m Bits $\alpha_i \in \{0,1\}, 1 \leq i \leq m$.
  Dann berechnet er m Graphen H_i, so dass H_i eine zufällige isomorphe Kopie von G_{α_i} ist.
  Der Verifizierer schickt die H_i an den Beweiser.
- ▷ Der Beweiser antwortet mit einem String von $\beta_i \in \{0,1\}, 1 \leq i \leq m$, so dass H_i isomorph zu G_{β_i} ist.

▷ Der Verifizierer überprüft $\alpha_i = \beta_i$ für alle $1 \leq i \leq m$.
Falls die Bedingung verletzt ist verwirft, ansonsten akzeptiert er.

Um zu verifizieren, dass es sich bei dem oben angegebenen Protokoll tatsächlich um ein interaktives Beweissystem handelt, müssen die beiden Eigenschaften

(1) Vollständigkeit
(2) Korrektheit

überprüft werden. In diesem konkreten Fall folgen beide unmittelbar:

Sind die beiden Graphen nicht isomorph, so kann der Beweiser immer eindeutig den entsprechenden Index i bestimmen und so den Verifizierer zum Akzeptieren bringen.

Sind die beiden Graphen nicht isomorph, so muss der Beweiser den Index i raten, wobei er eine Chance von $\frac{1}{2}$ hat, den richtigen Index zu tippen. Da er insgesamt n mal wählen muss, beträgt die Wahrscheinlichkeit, dass der Beweiser jeweils die richtige Wahl trifft 2^{-n}. Damit erfüllt das obige Protokoll die beiden geforderten Eigenschaften.
An dieser Stelle sollte erwähnt werden, dass nicht bekannt ist, ob das Graphenisomorphismusproblem $\mathbb{NP}$-vollständig ist. Dementsprechend ist es möglich, dass das Graphen-Nonisomorphismusproblem ein co$\mathbb{NP}$-vollständiges Problem darstellt. Wäre dies der Fall, so hätte dies weitreichende Konsequenzen: Die Polynomialzeithierarchie würde zusammenbrechen. Details hierzu findet man in [**Weg96**] sowie [**Pap94**].

2.2. IP = PSPACE. Als entscheidender Schritt soll an dieser Stelle die vollständige Charakterisation der Klasse IP erfolgen, die 1992 durch Adi Shamir [3] bewiesen wurde:

$$\mathbb{IP} = \mathbb{PSPACE}$$

Bevor auf den Beweis näher eingegangen wird, soll zunächst $\mathbb{PSPACE}$ definiert werden. Dazu [**Pap94**]:

DEFINITION 5. Sei $\mathcal{L}$ eine Sprache. Dann ist $\mathcal{L}$ in der Klasse $\mathbb{SPACE}$(f(n)), wenn eine Turingmaschine mit Ein- und Ausgabe existiert, die $\mathcal{L}$ auf durch f(n) begrenztem Speicherplatz entscheidet, wobei n die Länge der Eingabe ist. Insbesondere ist

$$\mathbb{PSPACE} = \mathbb{SPACE}(n^k)$$

SATZ 1. $\mathbb{QS}at$ ist $\mathbb{PSPACE}$-vollständig

Um diesen Satz zu verstehen, ist zunächst eine weitere Definition nötig, die klärt, was als $\mathbb{QS}at$ bezeichnet wird:

DEFINITION 6. Als $\mathbb{QS}at$ (quantified satisfiability) bezeichnet man folgende Fragestellung: Geben sein ein Boolscher Ausdruck Φ in konjunktiver Normalform mit Boolschen Variablen $x_1, \cdots, x_n$.
Ist dann $\exists x_1 \forall x_2 \exists x_3 \cdots Q_n x_n \Phi$ erfüllbar?

Für den Beweis des Satzes wird auf [**Pap94**] verwiesen.

Nach dieser Vorarbeit kann die Hauptaussage jetzt bewiesen werden.

[3] siehe [**Sha92**]

SATZ 2.

$$\mathbb{IP} = \mathbb{PSPACE}$$

Es soll im folgenden nur die Inklusion $\mathbb{IP} \supseteq \mathbb{PSPACE}$ gezeigt werden. Die umgekehrte Inklusion folgt aus der Tatsache, dass mit polynomialem Speicherplatz alle möglichen Interaktionen zwischen Verifizierer und Beweiser untersucht werden können, und sich daraus die Wahrscheinlichkeit, dass der Verifizierer akzeptiert, berechnen lässt. Näheres dazu findet man in [**Pap94**]
Wie kann die Behauptung $\mathbb{IP} \supseteq \mathbb{PSPACE}$ gezeigt werden?
Nach Satz (1) reicht es, einen interaktiven Beweis für $\mathbb{QS}at$ anzugeben. Dies soll in der Folge geschehen. Dabei sind drei Beobachtungen hilfreich:

(1) Quantifizierte boolsche Formeln (QBF) lassen sich in **einfache QBF** umformen.
(2) QBF können als **arithmetische Ausdrücke** aufgefasst werden.
(3) Die nötigen Berechnungen können in einem **endlichen Körper** erfolgen.

Punkt (1) bedarf noch einer Erklärung:

DEFINITION 7. Eine abgeschlossene QBF heißt einfach, wenn jedes Auftreten einer Variablen vom Ort ihrer Quantifizierung höchstens durch einen Allquantor getrennt ist (und beliebig viele andere Symbole).

Durch Bildung des existentiellen Abschlusses kann jede QBF in eine abgeschlossene QBF überführt werden. Diese wiederum bringt man anschließend durch Einführung von Füllvariablen auf einfache Form. Ein Beispiel soll dies demonstrieren:

$$\exists x_1 \forall x_2 \exists x_3 \forall x_4 ((x_1 \wedge x_2) \vee (x_3 \wedge x_4))$$

$$\leftrightarrow \exists x_1 \forall x_2 (x_1 = x_{1'} \wedge \exists x_3 \forall x_4 ((x_{1'} \wedge x_2) \vee (x_3 \wedge x_4))$$

$$\text{wobei:} x_1 = x_{1'} \Leftrightarrow (x_1 \wedge x_{1'}) \vee ((\neg x_1) \wedge (\neg x_{1'}))$$

Liegt eine QBF in einfacher, abgeschlossener Form vor, so kann sie wie folgt in einen arithmetischen Ausdruck verwandelt werden:

▷ **geg**: Sei A eine *abgeschlossene, einfache* QBF.
▷ **z.z.**: Es gibt einen interaktiven Beweis dafür, dass A erfüllbar ist.
▷ **Schritt**:
(1) Arithmetisiere die QBF
- Ersetze alle boolschen Variablen x_i durch Variablen $z_i \in \mathbb{Z}$
- Ersetze $\neg x_i$ durch $(1 - z_i)$
- Ersetze $\vee$ durch +, $\wedge$ durch $*$,
$\forall x_i$ durch $\prod_{z_i \in \{0,1\}}$ und $\exists x_i$ durch $\sum_{z_i \in \{0,1\}}$

(2) Führe einen interaktiven Beweis, dass $A \neq 0$ mod p.

Bemerkung:

Die Korrektheit des interaktiven Beweises beruht auf der Tatsache, dass $A \neq 0$, falls A erfüllbar ist. Dies lässt sich relativ einfach per vollständiger Induktion aufgrund der Form der Arithmetisierung zeigen.

Die Existenz einer Primzahl p, modulo derer $A \neq 0$ gilt, folgt ebenfalls per Induktion. Eine erste Abschätzung liefert sofort, dass $A \leq 2^{2^n}$ gilt, wobei n die Länge von A ist. In der Tat existiert sogar eine Primzahl p mit $2^n \leq p \leq 2^{3n}$.
Es ist nötig, die Rechnung jeweils modulo p auszuführen, damit die zu berechnenden Werte nicht zu groß werden, da ansonsten der Verifizierer die notwendigen Rechnungen nicht mehr vollziehen kann. Ebenfalls sollte der Beweiser die Primzahl wählen, da bei ungeschickter Wahl von p das Protokoll nicht erfolgreich durchgeführt werden kann.

Bevor der geforderte interaktive Beweis angegeben werden kann, ist eine weitere Definition nötig:

DEFINITION 8. Als **funktionelle Form** A' von A wird der arithmetische Ausdruck definiert, der durch *Elimination des am weitesten links stehenden* $\sum_{z_i \in \{0,1\}}$ *bzw.* $\prod_{z_i \in \{0,1\}}$*-Symbols* entsteht. Dieser kann als Polynom $q(z_i)$ einer freien Variable z_i ausgefaßt werden.

Interaktiver Beweis für $A \neq 0$ ***mod p***

- ▷ P sendet den von ihm behaupteten Wert a an V
- ▷ In jeder Runde wird A *in zwei Teile* A_1, der schon vollständig instanziert ist, *und* A_2, der mit dem am weitesten links stehenden $\sum$ bzw. $\prod$-Symbol beginnt, *zerlegt* und folgende Schritte vollzogen:
  (1) Falls A_2 *leer* ist, stoppt V und überprüft die Behauptung $a = a_1$
  (2) Falls A_1 *nicht leer* ist, so ersetzt V A durch A_2 und
  a durch $a - a_1$ mod p bzw. durch a/a_1 mod p.
  (3) *Ansonsten* sendet P das Polynom $q(z_i)$ von A' an V.
  - V überprüft $q(0) + q(1) = a$ mod p bzw. $q(0) * q(1) = a$ mod p,
  - schick eine Zufallszahl $r \in \mathbb{Z}_p$ an P und
  - ersetzt A durch $A'(r)$ mod p und a durch $q(r)$ mod p.

Das oben angegeben interaktive Protokoll soll an einem einfachen Beispiel getestet werden:
QBF: A = $\forall z_1(\neg z_1 \vee \exists z_2 \forall z_3(z_1 \wedge z_2 \vee z_3))$

- ▷ Arithmetisierung: *P und V* wandeln A um in
  $A = \prod_{z_1 \in \{0,1\}}((1 - z_1) + \sum_{z_2 \in \{0,1\}} \prod_{z_3 \in \{0,1\}}(z_1 * z_2 + z_3))$
- ▷ P berechnet den Wert des Ausdrucks A:
  $A = ((1-0) + ((0*0+0)*(0*0+1)) + ((0*1+0)*(0*1+1))) * ((1-1) + ((1*0+0)*(1*0+1)) + ((1*1+0)*(1*1+1))) = 2$
  P schickt anschließend a = 2 als Behauptung an V. (A $\neq$ 0 bedeutet, dass A erfüllbar ist.)
- ▷ Da A_2 nicht leer, A_1 hingegen leer ist, geht es mit (3) weiter.
  - P und V bestimmen die funktionelle Form
    $A' = ((1 - z_1) + \sum_{z_2 \in \{0,1\}} \prod_{z_3 \in \{0,1\}}(z_1 * z_2 + z_3))$
    Aus dieser berechnet P das Polynom $q(z_1)$:
    $A' = ((1 - z_1) + \sum_{z_2 \in \{0,1\}} \prod_{z_3 \in \{0,1\}}(z_1 * z_2 + z_3)) = z_1^2 + 1$
    das er an V schickt.

- V überprüft q(0) * q(1) = 1 + 2 = 2 = a und schickt ein zufälliges $r \in \mathbb{Z}_p$ an P, z.B. 3.
  Dann ersetzt V A durch $A'(r)$ und a = $q(r)$,
  hier also $q(3) = 3^2 + 1 = 10$.

▷ Nun ist A_2 immernoch nicht leer, da aber A_1 nicht leer ist berechnet V:
$A_1 = (1 - 3) = -2 \rightarrow a = 10 - (-2)$
und setzt $A = A_2 = \sum_{z_2 \in \{0,1\}} \prod_{z_3 \in \{0,1\}} (z_1 * z_2 + z_3)$

▷ Unter (3) werden jetzt folgende Schritte ausgeführt:
- P und V bestimmen die funktionelle Form
  $A' = \prod_{z_3 \in \{0,1\}} (3 * z_2 + z_3)$
  Aus dieser berechnet P das Polynom $q(z_2)$:
  $A' = \prod_{z_3 \in \{0,1\}} (3 * z_2 + z_3) = 9 * z_2^2 + 3 * z_2$
  das er an V schickt.
- V überprüft q(0) + q(1) = 0 + 12 = 12 = a und schickt ein zufälliges $r \in \mathbb{Z}_p$ an P, z.B. 2.
  Dann ersetzt V A durch $A'(r)$ und a = $q(r)$,
  hier also $q(2) = 9 * 2^2 + 3 * 2 = 42$.

▷ Wiederum ist A_2 nicht leer, A_1 hingegen leer. Deshalb wird Schritt (3) ausgeführt.
- P und V bestimmen die funktionelle Form
  $A' = (3 * 2 + z_3)$
  Aus dieser berechnet P das Polynom $q(z_3)$:
  $A' = (3 * 2 + z_3) = 6 + z_3$
  das er an V schickt.
- V überprüft q(0) * q(1) = 6 * 7 = 42 = a und schickt ein zufälliges $r \in \mathbb{Z}_p$ an P, z.B. 5.
  Dann ersetzt V A durch $A'(r)$ und a = $q(r)$,
  hier also $q(5) = 6 + 5 = 11$.

▷ Da jetzt A_2 leer ist, überprüft V, dass
A_1 = A = 11 = a gilt und akzeptiert.

Es bleibt die Vollständigkeit und Korrektheit des angegebenen interaktiven Protokolls zu zeigen. Dies ist relativ einsichtig:

(1) **Vollständigkeit**:
Ein ehrlicher Beweiser kann die von ihm behaupteten Werte und Polynome immer rechtfertigen, so dass der Verifizierer schließlich akzeptiert.

(2) **Korrektheit**:
Ein betrügerischer Beweiser, der ein inkorrektes Polynom übermittelt, erhält nur mit vernachlässigbarer Wahrscheinlichkeit einen korrekten Wert, wenn das Polynom an einer zufälligen Stelle ausgewertet wird.

Damit handelt es sich bei dem angegebenen Protokoll um ein interaktives Beweissystem. Da der Verifizierer die notwendigen Berechnungen in polynomialer Zeit und mit polynomialem Speicherplatz durchführen kann - dies dank der modulo p Rechnung - ist somit die Behauptung $\mathbb{IP} \supseteq \mathbb{PSPACE}$ gezeigt.

Interessierte finden in der Arbeit von Shamir [4] einen Erweiterung des Satzes auf sogenannte schwache Verifizierer, denen nur logarithmischer Speicherplatz zur Verfügung steht.

Bemerkung: Der hier vorgefürhte Beweis geht auf Shamir zurück, der als erster die Beziehung $\mathbb{IP} = \mathbb{PSPACE}$ bewies. Es gibt Vereinfachungen dieses Beweises [5], die allerdings ebenfalls die Idee der Arithmetisierung quantifizierter boolscher Formeln benutzen.

2.3. Weitere Folgerungen. Im vorhergehenden Abschnitt wurde die Mächtigkeit der Klasse IP gezeigt. Was aber macht diese Mächtigkeit aus?

Um der Antwort auf diese Frage etwas näher zu kommen, lohnt es sich, nochmals einen Blick zurück auf die Definition des interaktiven Beweissystems zu werfen. Diese soll im folgenden mehrfach leicht modifiziert werden. Die dabei beobachteten Änderung an der durch das neu entstehende System beschriebenen Klasse im Vergleich zu IP spiegelt die Bedeutung einzelner Teile der Definition für die Beweiskraft des interaktiven Systems wieder.

Man betrachte zunächst folgende Definition aus [**BOOGH$^+$90**]:

DEFINITION 9. **Arthur-Merlin Protokolle** sind interaktive Beweise für $\mathcal{L}, (M \leftrightarrow A)$, wobei Polynome r, l existieren, so dass jede Interaktion zwischen M und A bei einer Eingabe der Länge n folgende zusätzlichen Bedingungen erfüllt:

(1) Die Interaktion besteht aus aus $r(n)$ Runden.
(2) Jede gesendete Nachricht hat die Länge $l(n)$.
(3) *Die von A gesendeten $r(n)$ Nachrichten sind genau die ersten $r(n)$ durch den Zufallsgenerator von A erzeugten Werte.*

Worin liegt der Unterschied zur ursprünglichen Definition? Während die Festlegung auf eine Nachrichtenlängen sowie eine bestimmte Anzahl an Runden in Abhängigkeit von der Nachrichtenlänge offensichtlich keine wirkliche Einschränkung darstellt, scheint die Forderung, dass A nichts weiter durchführt, als Zufallsbits an M zu senden, nicht unerheblich zu sein. Umso überraschender ist das Ergebnis von Goldwasser und Sipser:

SATZ 3. Falls ein interaktives Beweissystem $(P \leftrightarrow V)$ für $\mathcal{L}$ existiert, so existiert ein Arthur-Merlin Protokoll $(M \leftrightarrow A)$

Es sei hervorgehoben, dass A die von ihm empfangene Nachricht nicht verstehen muss, um zu antworten. Die von A gesendete Zufallszahl ist in jeder Runde unabhängig von den vorhergegangenen Nachrichten, die M gesandt hat! Sendet A an M die von ihm erzeugten Zufallszahlen, so muss A keine weiteren Berechnungen ausführen, vielmehr kann M seinerseits alles, was A berechnet hätte, nun selbst berechnen.

Die Fähigkeit von A, Zufallszahlen zu generieren ist jedoch fundamental für einen interaktiven Beweis. Wäre A nämlich deterministisch, so könnte M alle von A gesendeten Nachrichten voraussagen und entsprechend das gesamte Protokoll alleine durchführen und, falls A im Protokoll akzeptiert, als Beweis an A schicken. Aus [**GMW91**] entnimmt man für diesen Fall, dass das interaktive Beweissystem mit deterministischem Verifizierer gerade $\mathbb{NP}$ beschreibt.

[4] [**Sha92**]
[5] siehe [**She92**]

Ebenso wichtig wie die Fähigkeit des Verifizierers, eine Münze zu werfen, ist die Fähigkeit des Beweisers, schwere Probleme zu lösen. In dem zu Anfang dieser Sektion angegebenen Beispiel des Graphen-Nonisomorphismus musste der Beweiser die Isomorphie zweier Graphen entscheiden. Auch in anderen interaktiven Protokollen wird vom Beweiser, der in der Definition des interaktiven Beweissytems nicht zeitlich beschränkt ist, erwartet, harte Probleme zu knacken.

Man kann der Auffassung sein, dass diese Forderung zu weit geht. In der Praxis ist es sinnvoll, P auch polynomial zeitbeschränkt anzunehmen. Mit dieser Annahme ist jedoch zunächst unklar, woher der Beweiser P Wissens erlangt hat, das der rechnerisch gleichstarke Verifizierer nicht erlangen kann, so dass es für den Verifizierer sinnvoll ist mit P überhaupt zu interagieren. Eine mögliche Erklärungen hierfür ist, dass P durch Zufall die Lösung eines harten Problems gefunden hat. Einsichtiger ist jedoch der Gedanke, dass P die Lösung bereits mit dem Problem erhalten hat, z.B. wenn P einen zu einem Graphen G isomorphen Graphen H erzeugt, so kennt P natürlich den Isomorphismus zwischen beiden.

Woher P auch immer diese Zusatzinformation gewonnen hat, sie wird sicherlich polynomial in der Länge der Eingabegröße sein. Wie groß ist nun die entsprechende Klasse, die hier mit 'Practical IP' bezeichnet wird, die durch einen solchen polynomialen Beweiser mit polynomialer Zusatzinformation entschieden werden kann? Brassard gelang es, Folgendes zu zeigen [6]:

SATZ 4.

$$\textit{"Practical } \mathbb{IP}\textit{"} \subseteq \mathbb{IP}(1)$$

wobei $\mathbb{IP}(1)$ die Klasse der Sprachen bezeichnet, die von einem interaktiven Beweissystem akzeptiert werden, beim dem der Beweiser genau eine Nachricht an den Verifizierer schickt und dieser anschließend alleine die Verifikation ausführt.

Somit stellt die Einschränkung des Beweisers auf polynomiale Zeit einen erheblichen Einschnitt in die Mächtigkeit der Klasse IP dar. Umgekehrt lässt sich schließen, dass der unbeschränkte Beweiser notwendig für die Mächtigkeit von IP ist.

An zwei weiteren Stellen der Definition könnte man Modifikationen vornehmen, und zwar bei der Definition der Vollständigkeit bzw. der Korrektheit. In beiden Fällen ist in der ursprünglichen Definition eine Fehlerwahrscheinlichkeit zugelassen. Was geschieht, wenn auf diese Fehlerwahrscheinlichkeit verzichtet wird?
Interessanterweise ist die Antwort nicht in beiden Fällen die gleiche:

SATZ 5. Falls $(P \leftrightarrow V)$ ein interaktives Beweissystem für $\mathcal{L}$ ist, so existiert ein interaktives Beweissystem $(P' \leftrightarrow V')$ für $\mathcal{L}$, wobei gilt:
absolute Vollständigkeit: $x \in \mathcal{L} \Rightarrow Pr[(P \leftrightarrow V)(x) \text{ akzeptiert }] = 1$

SATZ 6. Falls die Korrektheitsbedingung eines interaktives Beweissystem ersetzt wird durch:
absolute Korrektheit: $x \notin \mathcal{L} \Rightarrow Pr[(P \leftrightarrow V)(x) \text{ akzeptiert }] = 0 \qquad \forall P$
so gilt für die dadurch beschriebene Klasse $\mathbb{IP}$*:

[6] siehe [**GB90**]

$$\mathbb{IP}^* = \mathbb{NP}$$

Während also bedenkenlos gefordert werden kann, das ein interaktives Beweissystem alle korrekten Eingaben auch akzeptiert, ist die Möglichkeit, das $(P \leftrightarrow V)$ auch inkorrekte Eingaben akzeptiert notwendig, damit die beschriebene Klasse nicht auf $\mathbb{NP}$ zusammenfällt.

3. Interaktive Beweise und Zero-Knowledge

3.1. Was bedeutet Zero-Knowledge? Um die Zero-Knowledge Idee zu motivieren wird in vielen Büchern ein in etwa wie folgt geartetes Beispiel vorangestellt:

> Alice und Victor treffen sich auf dem Schulhof.
> Alice: Ich kenne . . . (ein Geheimnis)!
> Victor: Glaub ich dir nicht!
> Alice: Ich kann's dir aber beweisen. Hier, ich zeig's dir.
> Victor: Toll, jetzt kenn ich's auch!
> Alice: *!'§*!

Das Dilemma in dem Alice steckt, ist offensichtlich. Zwar will sie auf der einen Seite Victor davon überzeugen, dass sie eine bestimmte Information besitzt, auf der anderen Seite will sie eben diese Information vor ihm auch nicht offenlegen. Alice Wunsch wäre demnach ein Protokoll, mit dem sie Victor ihr Wissen 'beweisen' kann, dass Bob allerdings kein zusätzliches Wissen verschafft.

Um ein solches Protokoll angeben zu können, ist es jedoch zunächst nötig, die Aussage *'Bob gewinnt kein Wissen'* zu präzisieren und formal zu fassen.
Auf die Frage, was Zero-Knowledge bedeutet, gibt es mehrere Ansätze. In Bruce 'Angewandte Kryptographie' findet man folgende vier Arten von Zero-Knowlege Beweisen aufgelistet:

Perfect:: Es gibt einen Simulator, der Protokolle liefert, die dieselbe Verteilung wie die echten Protokolle aufweisen [. . .]

Statistical:: Es gibt einen Simulator, der Protokolle liefert, die bis auf eine gleichbleibende Anzahl von Ausnahmen dieselbe Verteilung wie die echten Protokolle aufweisen.

Computational:: Es gibt einen Simulator, der Protokolle liefert, die von echten Protokollen nicht zu unterscheiden sind.

No-use:: Ein Simulator ist vielleicht nicht vorhanden, wir können aber beweisen, das Victor keine polynomial anwachsende Menge an Informationen aus den Beweisen erfährt.

In der Praxis ist es nicht nötig, die *perfekte Zero-Knowledge Eigenschaft* von einem Protokoll zu fordern, vielmehr ist der Begriff des *computational Zero-Knowledge* hier der wichtige. Dieser garantiert, dass Victor alles, was er nach der Interaktion mit Alice effizient berechnen kann, auch schon vorher hätte effizient berechnen können.

Der Begriff *computational Zero-Knowledge* soll deshalb nochmals näher betrachtet werden, insbesondere muss der intuitiv verständliche Begriff eines Simulators formal gefasst werden [7]:

[7] siehe [**Gol01**]

DEFINITION 10. Sei $(P \leftrightarrow V)$ ein interaktives Beweissystem für eine Sprache L.
Dann ist $(P \leftrightarrow V)$ **computational Zero-Knowledge**, wenn zu jeder polynomial zeitbeschränkten interaktiven Turing-Maschine $\mathbb{ITM}$ V* ein probabilistischer, polynomial zeitbeschränkter Algorithmus M* existiert, so dass die folgenden beiden Mengen rechnerisch nicht zu unterscheiden sind:

(1) $(P \leftrightarrow V^*)(x)_{x \in L}$
(Ausgabe der $\mathbb{ITM}$ V* nach der Interaktion mit P bei gemeinsamer Eingabe x)
(2) $M^*(x)$
(Ausgabe von M* bei Eingabe von x)

Bemerkungen:

- ▷ Unter einer interaktive Turing-Maschine versteht man ein deterministische multi-Band Turing-Maschine. Diese besitzt ein read-only Eingabeband, ein read-only Zufallsband, ein Arbeitsband, ein write-only Ausgabeband sowie ein Paar Kommunikationsbänder, wovon eines read-only, das andere write-only ist. Zusätzlich verfügt eine Turing-Maschine über ein Identitätsbit sowie ein switch-tape, das aus einer einzelnen Zelle besteht. Falls der Inhalt dieser Zelle gleich der Maschinenidentität ist, ist die $\mathbb{ITM}$ aktiv, ansonsten befindet sie sich in einem idle-Zustand. Man vergleiche diese Definition aus [**Gol01**] mit der zu Anfang gegebenen Definition!
- ▷ M* wird dann als Simulator für die Interaktion von V* mit P bezeichnet. Man beachte, dass M* nicht mit V* interagiert, jedoch nicht gefordert wird, dass ein Simulator M* für alle V* existiert.

  Es gibt (äquivalente Definitionen) des Begriffs 'Computational Zero-Knowledge', die die Existenz eines Simulators M fordern, der für alle V* eine entsprechende Ausgabe liefert. In diesen Definitionen wird jedoch dann davon ausgegangen, dass M mit V* interagiert.
- ▷ Für die Definition von rechnerischer Ununterscheidbarkeit soll an dieser Stelle auf [**Gol01**] verwiesen werden.

3.2. Interaktion und Zero-Knowledge. Nachdem der Begriff des *computational Zero-Knowledge* als Eigenschaft eines interaktiven Beweissystems definiert wurde, stellt sich nun die Frage, welche Sprachen damit erfasst werden. Darüber gibt der nächste Satz Auskunft.

SATZ 7. Unter der Annahme, dass Einwegfunktionen existieren, lässt sich jeder interaktive Beweis in einen interaktiven computational Zero-Knowledge Beweis transformieren.

$$\mathbb{IP} = \mathbb{CZK}$$

Den Beweis dieses Theorems entnimmt man [**BOOGH$^+$90**]. Hier soll lediglich die Beweisidee in groben Zügen wiedergegeben werden.

Die Idee des Beweises ist ein gegebenes interaktives Beweissystem $(P \leftrightarrow V)$ in ein System $(P' \leftrightarrow V')$ zu transformieren, so dass letzteres die *computational Zero-Knowledge* Eigenschaft besitzt.

Bereits im vorigem Abschnitt wurde erwähnt, dass ohne Einschränkung $(P \leftrightarrow V)$ als einseitiges Arthur-Merlin Protokoll $(M \leftrightarrow A)$ mit M deterministisch angenommen werden kann. Dieses wird nun wie folgt zu $(P' \leftrightarrow V')$ modifiziert:
Das Protokoll soll in zwei Phasen ablaufen.

(1) In der ersten Phase verhalten sich P' und V' genauso wie M und A, mit dem einzigen Unterschied, dass P' seine *Nachrichten an V' verschlüsselt*.
(2) In der zweiten Phase, wenn die festgelegte Anzahl an Runden der ersten Phase vorbei ist, entscheidet P' ob A die unverschlüsselte Interaktion akzeptiert hätte. Ist dies der Fall, überzeugt er V' von dieser Tatsache. Dies ist ein $\mathbb{NP}$-Problem und kann mit Hilfe eines bekannten Zero-Knowledge Protokolls gelöst werden.

Betrachte man das angegebene Protokoll, so ist klar, dass sowohl die Vollständigkeit als auch die Korrektheit bei diesem interaktiven Beweissystem gegeben sind, da diese sich auf die Vollständigkeit und Korrektheit des ursprünglichen Protokolls $(M \leftrightarrow A)$ sowie des verwendeten Protokolls, dass V' von der Tatsache, dass A akzeptiert hätte, überzeugt, zurückführen lassen. Der Beweis der Zero-Knowledge Eigenschaft reduziert sich darauf, die Zero-Knowledge Eigenschaft in der ersten Phase nachzuweisen. Auch hier soll wieder auf [**BOOGH$^+$90**] verwiesen werden.
Eine interessante sich nun stellende Frage ist, ob es Zero-Knowledge Protokolle ohne Interaktion geben kann. In der Tat ist dies der Fall. Hierfür soll ein Protokoll angeben werden, dass [**Sch96**] entnommen ist.

(1) Alice verwendet ihre Information und n Zufallszahlen, um das harte Problem in n verschiedene isomorphe Probleme zu überführen
(2) Alice legt sich auf die Lösungen der n harten Probleme fest.
(3) Diese Festlegungen werden als Eingabe einer Einweg-Hashfunktion verwendet, deren n erste Ausgabebits Alice speichert.
(4) Je nachdem ob das i-te Bit 0 oder 1 war, verfährt Alice wie folgt: Bei 0 beweist sie die Isomorphie des alten und des i-ten neuen Problems. Bei 1 löst sie das i-te neue Problem
(5) Alice veröffentlich ihre Festlegungen sowie die jeweiligen Lösungen aus (4)

Jeder kann nun nachprüfen, ob Alice die Schritte (2)-(5) korrekt ausgeführt hat.

Die Korrektheit des Protokolls beruht dabei auf der Tatsache, dass Alice die Ausgabe der Einweg-Hashfunktion nicht vorhersehen kann, da diese eine zufällige Bitfolge erzeugt. Somit weiß Alice nicht im voraus, welcher Aufforderung sie in (4) nachkommen muss. Die *Einweg-Hashfunktion hat somit den Verifizierer Viktor ersetzt*.

Jedoch besteht bei diesem Protokoll die Möglichkeit, dass Alice die Ausgabebits der Hashfunktion erät bzw. durch einen bruteforce-Angriff solange das Protokoll durchführt, bis sie eine 'passende' Ausgabefolge erhält.

Damit dies ausgeschlossen werden kann, muss die Anzahl der Iterationen des Protokolls entsprechend höher sein als bei einem interaktiven Zero-Knowledge Beweis. In Zahlen bedeutet dies, dass im Gegensatz zu etwa 10 Iterationen bei interaktiven Protokollen hier 64 bis 128 Iterationen notwendig sind, um sicherzustellen, dass Alice die Ausgabe der Hashfunktion nicht kennt und das Protokoll somit korrekt ist.

4. Offene Probleme

Zum Abschluss sollen noch einige bisher offene Probleme und Fragestellung angesprochen werden:

(1) **Die Struktur der $\mathbb{IP}$-Hierarchie**
Bisher bekannt ist, dass ein sogenannter linearer Speed-up Theorem gilt:

$$\mathbb{IP}(O(f(.)))\mathbb{IP}(f(.))$$

Welchen Einfluss ansonsten die Anzahl der ausgetauschten Nachrichten auf die Ausdruckskraft des interaktiven Beweissystems hat, ist unbekannt.

(2) **Stärke des Beweisers**
Bei der Diskussion über die Klasse 'Practical IP' wurde die Einschränkung des Beweisers vorgenommen. In diesem Zusammenhang stellt sich die Frage, wie stark bei gegebenem L ein Beweiser sein muss, bzw. wie weit er eingeschränkt werden kann, so dass er trotzdem noch einen Verifizierer überzeugt.

(3) $\mathbb{IP} = \mathbb{PSPACE}$
Gibt es für diese Gleichung einen Beweis, der nicht auf abstrakte, algebraische Methoden zurückgreift?

5. Fazit

Das interaktive Beweissystem zeichnet sich durch die Elemente *Unsicherheit* und *Zufälligkeit* aus. Ohne diese Elemente besitzt ein solches Beweissystem nicht die Ausdruckskraft, die für $\mathbb{IP}$ durch Shamir bewiesen wurde. Der Beweis der vollständigen Charakterisation der Klasse $\mathbb{IP}$ beruht auf algebraischen Methoden und konnte bis heute noch nicht ohne diese geführt werden.

Ein genaueres Studium der Hierarchie der Klasse $\mathbb{IP}$ verspricht weitreichende Einsichten. So konnte angedeutet werden, dass, falls $\mathbb{NP} \subseteq \mathbb{IP}(2)$ gilt, die Polynomialzeithierarchie kollabiert. Ob dies tatsächlich der Fall ist, ist noch Gegenstand der Forschung.

Neben diesem theoretischen Aspekt bieten sich interaktive Beweise für die Kryptographie an. Sie erlauben beispielsweise, Zero-Knowledge Beweise wesentlich effektiver durchzuführen.

KAPITEL 5

Primzahlen (Dominik Schultes)

ZUSAMMENFASSUNG (ABSTRACT)
Diese Ausarbeitung beschäftigt sich mit dem Entscheidungsproblem, ob eine gegebene Zahl eine Primzahl ist. Im Vordergrund steht dabei ein stochastischer Algorithmus, der dieses Problem sehr effizient, aber mit einer gewissen Fehlerwahrscheinlichkeit löst.

*„Dass die Aufgabe, die Primzahlen von den zusammengesetzten zu unterscheiden und letztere in ihre Primfaktoren zu zerlegen, zu den wichtigsten und nützlichsten der gesamten Arithmetik gehört und die Bemühungen und den Scharfsinn sowohl der alten wie auch der neueren Geometer in Anspruch genommen hat, ist so bekannt, dass es überflüssig wäre, hierüber viele Worte zu verlieren ... außerdem aber dürfte es die Würde der Wissenschaft erheischen, alle Hülfsmittel zur Lösung jenes so eleganten und berühmten Problems fleißig zu vervollkommnen.“ (*GAUSS*, Disquisitiones Arithmeticae, Abs. 329)*

1. Einführung

1.1. Motivation. Dass große Primzahlen in der Kryptographie eine entscheidende Rolle spielen, kann man schnell feststellen, wenn man sich die vorhergehenden Ausarbeitungen anschaut:

- „[Der RSA-Algorithmus] Sei $n = p * q$ das Produkt zweier verschiedener Primzahlen p und q ...“ (Christian Ziemann, 26.11.02)
- „[Das Fiat-Shamir Identifikations-Verfahren] Peggy wählt große, verschiedene Primzahlen p und q und berechnet $n := pq$ ...“ (Alexander Petry, 03.12.02)
- „[Das No-Key-Verfahren] Alice, Bob und Carol bestimmen große Primzahl p ...“ (Georg Westenberger, 10.12.02)

Große Primzahlen sind also eine Grundlage von vielen kryptographischen Verfahren, so dass man einen Algorithmus finden muss, der die benötigten großen Primzahlen liefert.

1.2. Grundsätzliche Vorgehensweise. Während man große zusammengesetzte Zahlen einfach konstruieren kann, indem man zwei beliebige Zahlen (mit ungefähr der Hälfte der Anzahl an benötigten Stellen) multipliziert, ist ein ähnlich einfacher Ansatz für Primzahlen nicht möglich.
Deshalb behilft man sich im Allgemeinen folgendermaßen: Man erzeugt große Zufallszahlen und entscheidet dann, ob es sich um eine Primzahl handelt. [1] Als erstes stellt sich bei dieser

[1] Das Primzahl-Entscheidungsproblem wird PRIMES genannt.

Vorgehensweise natürlich die Frage, ob man überhaupt eine Chance hat, eine Primzahl zu finden.
Man fängt bei einer beliebigen ungeraden Zahl n_0 der gewünschten Größenordnung an und entscheidet nacheinander für $n_0, n_0+2, n_0+4, n_0+6, \ldots$, ob es sich um eine Primzahl handelt, so lange bis man eine Primzahl gefunden hat. [2] Da es unendlich viele Primzahlen gibt, findet man auf jeden Fall in endlicher Zeit eine Primzahl.
Um abschätzen zu können, wie lange es ungefähr dauert, bis man eine Primzahl findet, helfen folgender Satz und folgende Folgerung.[3]

SATZ 1 (Primzahlsatz). Die Anzahl der Primzahlen unterhalb n_0 ist asymptotisch gleich $\frac{n_0}{\ln(n_0)}$, d.h. $\pi(n_0) \sim \frac{n_0}{\ln(n_0)}$.

FOLGERUNG 2 (Primzahlen in einem Intervall). $\pi(2n_0) - \pi(n_0) \sim \pi(n_0)$, die Anzahl der Primzahlen im Intervall $[n_0, 2n_0]$ ist also ebenfalls asymptotisch gleich $\frac{n_0}{\ln(n_0)}$.

Vereinfacht ausgedrückt, ist, wenn man das Intervall $[n_0, 2n_0]$ betrachtet, durchschnittlich jede $\ln(n_0)$-te Zahl eine Primzahl. (Bsp.: $n_0 = 10^{200} + 1 \rightsquigarrow \ln(n_0) \approx 461$)

2. Zufallszahlen

2.1. Vorbemerkungen. Wie im vorhergehenden Abschnitt gezeigt, benötigen wir eine beliebige ungerade Zahl n_0 der gewünschten Größenordnung als Startwert bei der Suche nach einer großen Primzahl. Eine solche Zahl kann – wie im nächsten Abschnitt gezeigt wird – „zufällig“ erzeugt werden. Hierbei spielt es keine Rolle, dass man mit einer deterministischen Maschine nur Pseudozufallszahlen erzeugen kann, da ja einfach nur eine beliebige Zahl gesucht wird.
Darüber hinaus wird ein Zufallszahlengenerator benötigt, wenn im Abschnitt 3.4 ein stochastischer Primtest-Algorithmus vorgestellt wird. Dieser benötigt eigentlich „echte“ Zufallszahlen, die uniform verteilt sind. Und bei den mathematischen Ausführungen beim Korrektheitsbeweis wird auch davon ausgegangen, dass „echte“ Zufallszahlen verwendet werden.
Trotzdem benutzt man natürlich in der Praxis (aus Mangel an Alternativen) auch für diesen Fall nur uniform verteilte Pseudozufallszahlen.

2.2. Erzeugung von (Pseudo-)Zufallszahlen. Die mit Abstand am häufigsten verwendete Methode zur Erzeugung von Pseudozufallszahlen ist die der linearen Kongruenz. Man kann sie mit Hilfe folgender Formel beschreiben: [4]

$$R_m = (aR_{m-1} + c) \mod n$$

Nach der Wahl einer Startzufallszahl R_0 und Konstanten a, c und n wird durch diese Formel eine Folge (R_m) von „Zufallszahlen“ aus der Menge $\{0, 1, \ldots, n-1\}$ definiert. Die nächste

[2] s.a. [**Hei02**], Kap. 4, S. 120
[3] Beweise in [**HW58**], Kap. 22
[4] s.a. [**Ski98**], S. 218

„Zufallszahl“ berechnet sich also bei diesem Verfahren immer aus der zuletzt erzeugten „Zufallszahl“.
Bei der Wahl der Konstanten kann man viel falsch machen, d.h. wenn die Konstanten schlecht gewählt werden, kann die Folge sehr schnell in einen Zyklus geraten, so dass die Folge nicht die Eigenschaft einer uniformen Verteilung besitzt, dass alle Zahlen aus der Menge $\{0, 1, \ldots, n-1\}$ gleichwahrscheinlich ausgewählt werden, da in diesem Fall viele Zahlen nie ausgewählt werden. Die Konstanten sollte man also nicht „zufällig“ wählen.
Knuth untersucht in [**Knu98**], Kap. 3.2.1 sehr genau, welche Eigenschaften die Konstanten haben müssen, damit die erzeugten Zufallszahlen möglichst „gut“ sind. Wenn man die auf S. 184f zusammengefassten Regeln befolgt, hat die Methode der linearen Kongruenz die Vorteile, dass sie einfach und effizient realisiert werden kann und dass sie sehr gute Ergebnisse liefert, d.h. dass die erzeugten Zahlen die wesentlichen Eigenschaften von uniform verteilten Zufallszahlen besitzen.

3. Primzahl-Tests

3.1. Problematik. Es ist nicht einfach, bei einer großen Zahl festzustellen, ob es sich um eine Primzahl handelt, da man in der Kryptographie unter einer „großen Zahl“ eine Zahl mit 512 binären Stellen und mehr versteht.
Das Brute-Force-Verfahren, das, um zu testen, ob eine Zahl n prim ist, Divisionen durch alle möglichen Teiler von 2 bis $\sqrt{n}$ durchführt, müsste also bei einer Zahl der Größenordnung 2^{512} ca. $2^{256} \approx 10^{77}$ Divisionen vornehmen.
Das Sieb des ERATOSTHENES[5] würde nicht nur an der Laufzeit, sondern auch am Speicherplatz scheitern.
Die naiven Verfahren haben bzgl. der Größe der Eingabe (also der Anzahl an Stellen) exponentielle Laufzeit und sind daher für die in der Kryptographie benötigte Größenordnung ungeeignet.
Im worst case werden nämlich $\sqrt{n} - 1 = 2^{\log_2 (n^{1/2})} - 1 = (\sqrt{2})^{\log_2 n} - 1$ potentielle Teiler betrachtet, wobei n die zu testende Zahl ist und demzufolge $\log_2 n$ die Länge der Eingabe.

3.2. Lösungsmöglichkeiten. Trotz dieser Problematik wurden in den letzten Jahren verschiedene Lösungsmöglichkeiten entdeckt, die auf zahlentheoretischen Erkenntnissen beruhen, wie z.B. dem „kleinen FERMAT“ (Satz 6).
1976 stellte MILLER einen deterministischen Algorithmus vor, der PRIMES in polynomialer Zeit löst unter der Voraussetzung, dass die Erweiterte Riemannsche Hypothese (ERH) gilt.[6]
1977 veröffentlichten SOLOVAY und STRASSEN einen stochastischen Algorithmus, der PRIMES garantiert in polynomialer Zeit - allerdings mit einer gewissen Fehlerwahrscheinlichkeit - löst. Eine vereinfachte Variante dieses Algorithmus bildet den Kern dieser Ausarbeitung.

[5] Eine kurze Beschreibung dieses Verfahrens fi ndet man bspw. in [**Kra86**], S. 40 oder in [**HW58**], S. 4.
[6] [**Kra86**], S. 59-62

1983 wurde durch einen deterministischen Algorithmus von ADLEMAN, POMERANCE und RUMELY die Komplexität $(\log n)^{O(\log\log\log n)}$ erreicht. [7]
Aus Sicht der theoretischen Informatik gab es im August 2002 den größten Durchbruch, als AGRAWAL, KAYAL und SAXENA [**AKS02**] bewiesen, dass PRIMES in P liegt, indem sie einen deterministischen Algorithmus angaben, der das Problem in polynomialer Zeit löst, ohne unbewiesene mathematische Hypothesen zu benutzen. Aus praktischer Sicht hat dieser Algorithmus allerdings den Nachteil, dass die Laufzeit sich nur durch ein Polynom relativ hohen Grades beschränken lässt, nämlich durch $O((\log n)^{12})$ bzw. unter gewissen Voraussetzungen durch $O((\log n)^6)$. Deshalb sind stochastische Algorithmen aus Effizienzgründen nach wie vor interessant.
Einen etwas ausführlicheren „historischen Abriss“ findet man in der Einleitung zu [**AKS02**], weitere Verfahren in [**Kra86**], Kap. 2.

3.3. mathematische Grundlagen. Um den im folgenden Abschnitt angegebenen stochastischen Primtest-Algorithmus formulieren zu können, wird die Definition des Restklassenrings $\mathbb{Z}$ modulo n benötigt.[8]

DEFINITION 1 (kommutativer Ring mit Einselement). Ein kommutativer Ring $(R, +, \cdot)$ mit Einselement ist eine Menge $R \neq \emptyset$ mit zwei Verknüpfungen $+$ und $\cdot$ auf R, so dass folgende Axiome gelten:

(I) Addition: $(R, +)$ ist eine abelsche Gruppe.
(II) Multiplikation: $(R, \cdot)$ ist eine abelsche Halbgruppe mit neutralem Element.
(III) Distributivgesetze: $\forall a, b, c \in R$:

$$\begin{aligned} a \cdot (b + c) &= a \cdot b + a \cdot c \text{ und} \\ (a + b) \cdot c &= a \cdot c + b \cdot c \end{aligned}$$

DEFINITION 2 (Restklassenring $\mathbb{Z}$ modulo n).

- $\mathbb{Z}_n := \{0, 1, \ldots, n-1\}, n \in \mathbb{Z}, n > 1$,
- $a \oplus b \quad (\text{in } \mathbb{Z}_n) := (a + b) \mod n \quad (\text{in } \mathbb{Z})$,
- $a \odot b \quad (\text{in } \mathbb{Z}_n) := (a \cdot b) \mod n \quad (\text{in } \mathbb{Z})$.

SATZ 3 (Restklassenring $\mathbb{Z}$ modulo n). $(\mathbb{Z}_n, \oplus, \odot)$ ist ein kommutativer Ring mit Einselement.
Beweis:

(I) Addition
  (i) (Abgeschlossenheit) Seien $a, b \in \mathbb{Z}_n$beliebig. $a \oplus b = ((a + b) \mod n) \in \mathbb{Z}_n$
  (ii) (Assoziativgesetz) gilt, da das Assoziativgesetz in $\mathbb{Z}$ gilt
  (iii) (Nullelement) 0 ist das Nullelement: Sei $a \in \mathbb{Z}_n$ beliebig.
  $a \oplus 0 = a + 0 \mod n = a \mod n = a$, da $a < n$, da $a \in \mathbb{Z}_n$
  (iv) (inverses Element) Sei $a \in \mathbb{Z}_n$ beliebig. $-a := n - a$ ist das inverse Element zu a, denn $a \oplus (-a) = a \oplus (n - a) = (a + n - a) \mod n = n \mod n = 0$

[7] [**Kra86**], S. 73-77

[8] Die im Folgenden angeführten Definitionen findet man vermutlich in jedem Buch über „Lineare Algebra“, diese Ausarbeitung basiert auf [**Ung00**].

(v) (Kommutativgesetz) gilt, da das Kommutativgesetz in $\mathbb{Z}$ gilt

(II) Multiplikation

(i) (Abgeschlossenheit) Seien $a, b \in \mathbb{Z}_n$beliebig. $a \odot b = ((a \cdot b) \mod n) \in \mathbb{Z}_n$

(ii) (Assoziativgesetz) gilt, da das Assoziativgesetz in $\mathbb{Z}$ gilt

(iii) (Einselement) 1 ist das Einselement: Sei $a \in \mathbb{Z}_n$ beliebig.
$a \odot 1 = a \cdot 1 \mod n = a \mod n = a$, da $a < n$, da $a \in \mathbb{Z}_n$

(iv) (Kommutativgesetz) gilt, da das Kommutativgesetz in $\mathbb{Z}$ gilt

(III) Distributivgesetze
gelten, da die Distributivgesetze in $\mathbb{Z}$ gelten

q.e.d. Satz 3

$\mathbb{Z}_n$ ist also für jedes $n > 1$ ein kommutativer Ring mit Einselement.
Ist $\mathbb{Z}_n$ aber auch ein Körper ? Die Antwort gibt Satz 4.

DEFINITION 3 (Körper). Ein Körper $(K, +, \cdot)$ ist ein kommutativer Ring mit Einselement, in dem $(K \setminus \{0\}, \cdot)$ eine Gruppe ist.

SATZ 4. $\mathbb{Z}_n$ Körper gdw. n prim

Beweis:

$\Rightarrow$)

Kontraposition: n nicht prim $\Rightarrow \mathbb{Z}_n$ kein Körper.
Sei $n \in \mathbb{Z}, n > 1$, keine Primzahl.
$\rightsquigarrow \exists a, b \in \{2, \ldots, n-1\} : a \cdot b = n$
Annahme: $\mathbb{Z}_n$ ist ein Körper.
$\rightsquigarrow \exists b^{-1} \in \mathbb{Z}_n$
$\rightsquigarrow a \odot b = (a \cdot b) \mod n = n \mod n = 0$
$\rightsquigarrow a \odot b \odot b^{-1} = 0 \odot b^{-1}$
$\rightsquigarrow a \odot 1 = 0$
$\rightsquigarrow a = 0$
Dies ist ein Widerspruch zu $a \in \{2, \ldots, n-1\}$.
$\rightsquigarrow \mathbb{Z}_n$ ist kein Körper. q.e.d. $\Rightarrow$)

$\Leftarrow$)

Sei $n \in \mathbb{Z}$ eine Primzahl. Sei $a \in \mathbb{Z}_n \setminus \{0\}$ beliebig.
z.z.: $\exists a^{-1} : a \odot a^{-1} = 1$
Betrachte die Zahlen $a \odot 0, a \odot 1, \ldots, a \odot (n-1)$.
Hilfsbehauptung: Diese Zahlen sind alle verschieden.
Beweis der Hilfsbehauptung:
Indirekt. Annahme: Zwei dieser Zahlen sind gleich.
$\rightsquigarrow \exists b, c, 0 \leq b < c < n : a \odot b = a \odot c$
$\rightsquigarrow a \odot (c - b) = 0$
$\rightsquigarrow a \cdot (c - b) \mod n = 0$
$\rightsquigarrow \exists x \in \mathbb{Z}, x > 0 : a \cdot (c - b) = x \cdot n$, da $a \cdot (b - c) \neq 0$, da $a \neq 0$ und $(c - b) \neq 0$
Die Zahl $a \cdot (c - b)$ enthält n nicht als Primfaktor, da $a < n$ und $(c - b) < n$;
$x \cdot n$ hingegen enthält offensichtlich n als Primfaktor.

Dies ist ein Widerspruch zur Gleichung $a \cdot (c - b) = x \cdot n$.
q.e.d. Hilfsbehauptung
Da alle Zahlen $a \odot 0, a \odot 1, \ldots, a \odot (n-1)$ verschieden sind und wegen der Abgeschlossenheit der Multiplikation in $\mathbb{Z}_n$ liegen, gilt:
$\{a \odot 0, a \odot 1, \ldots, a \odot (n-1)\} = \{0, 1, \ldots, n-1\} = \mathbb{Z}_n$ (*)
$\rightsquigarrow \exists a^{-1} \in \mathbb{Z}_n : a \odot a^{-1} = 1$ q.e.d. $\Leftarrow$)
q.e.d. Satz 4

3.4. Stochastischer Primtest-Algorithmus. Der stochastische Primtest-Algorithmus, der nun vorgestellt wird, basiert auf dem Algorithmus, den STRASSEN in [**Weg96**], S. 261 angibt und der wiederum eine Vereinfachung des Algorithmus von SOLOVAY und STRASSEN [**SS77**] darstellt, wobei die Vereinfachung durch folgende Einschränkung möglich wird: n und $\frac{n-1}{2}$ müssen ungerade sein.
Sei $\varepsilon, 0 < \varepsilon < 1$, die Fehlerwahrscheinlichkeit beliebig, aber fest gewählt und sei $n \in \mathbb{N}, n$ und $\frac{n-1}{2}$ ungerade, die zu testende Zahl.

function isPrime(n)
  $k := \lceil -\log_2 \varepsilon \rceil$
  for $i := 1$ **to** k **do begin**
    Wähle Zufallszahl a_i aus einer uniformen Verteilung auf $\{1, \ldots, n-1\} = \mathbb{Z}_n \setminus \{0\}$
    Berechne $b_i := a_i^{\frac{n-1}{2}}$ in $\mathbb{Z}_n$
    if $b_i \neq \pm 1$ **then return** *false*
  end
  return *true*

Wenn n prim ist, trifft der Algorithmus *immer* die richtige Entscheidung (also immer *true*).
Wenn n nicht prim ist, trifft der Algorithmus mit einer Wahrscheinlichkeit $\leq \varepsilon$ die falsche Entscheidung (also *true*, obwohl n nicht prim ist).
Oder anders ausgedrückt: Die Antwort *false* ist immer korrekt, d.h. n ist garantiert nicht prim, wenn der Algrotihmus *false* liefert. Wenn der Algorithmus hingegen *true* liefert, ist es möglich, dass n trotzdem nicht prim ist.

3.5. Bemerkungen zu den Einschränkungen. Die Einschränkung, dass nur ungerade n getestet werden können, ist keine wirkliche Einschränkung, da jede gerade Zahl $n > 2$ keine Primzahl ist.
Die zweite Bedingung, $\frac{n-1}{2}$ ungerade, hingegen schließt potentielle Primzahlen aus, nämlich jede zweite ungerade Zahl. Bspw. kann durch den Algorithmus weder die 5 noch die 9 getestet werden, wobei 5 eine Primzahl ist, 9 aber nicht.
Der Vorteil dieser Einschränkung ist, dass der Korrektheitsbeweis vereinfacht wird. Außerdem war ja das ursprüngliche Ziel, große Primzahlen zu finden, und dies ist mit Hilfe des Verfahrens aus Abschnitt 1.2 (das natürlich entsprechend angepasst werden muss) und dem angegebenen Algorithmus möglich. Es gilt nämlich Folgendes:[9]

[9]Beweis in [**HW58**], S. 14

SATZ 5. Es gibt unendlich viele Primzahlen der Form $4m + 3$.

Es gibt also unendlich viele Primzahlen $n = 4m + 3$, wobei $\frac{n-1}{2} = \frac{4m+2}{2} = 2m + 1$ ungerade.
Demzufolge findet man trotz der Einschränkung in endlicher Zeit eine Primzahl.
Welche Änderungen am oben angegebenen Algorithmus vorgenommen werden müssen, wenn man auch Zahlen $n, \frac{n-1}{2}$ gerade, testen möchte, werde ich hier nur kurz andeuten, mich aber beim Korrektheitsbeweis wieder auf die vereinfachte Version beziehen. Den Original-Algorithmus von SOLOVAY und STRASSEN (ohne die Einschränkung, dass $\frac{n-1}{2}$ ungerade sein muss) samt Fehlerabschätzung findet man in [**SS77**] und ausführlicher in [**Hei02**], Kap 4.4.
Als erstes muss zusätzlich geprüft werden, ob n und a_i teilerfremd sind. Falls nein, wird sofort mit dem Ergebnis *false* abgebrochen.
Außerdem muss, damit n weiter als Primzahl in Frage kommt, b_i nicht nur gleich $+1$ oder -1 sein, sondern zusätzlich auch das richtige Vorzeichen haben, wobei es von n und a_i abhängt, ob b_i gleich $+1$ oder gleich -1 sein muss; genauer: Damit n weiter als Primzahl in Frage kommt, muss $b_i = \left(\frac{a_i}{n}\right)$ gelten, wobei $\left(\frac{a_i}{n}\right) \in \{-1, +1\}$ das Jacobi-Symbol [10] ist.
Das Ergebnis der Fehlerabschätzung ändert sich nicht.

3.6. Korrektheitsbeweis. Nun soll die schon im Abschnitt 3.4 angegebene Fehlerabschätzung bewiesen werden.[11]
Die Grundlage ist der kleine Satz von FERMAT:

SATZ 6 (kleiner FERMAT). n prim $\Rightarrow \forall a \in \mathbb{Z}_n \setminus \{0\} : a^{n-1} = 1$ (in $\mathbb{Z}_n$)

Beweis:

Sei $n \in \mathbb{Z}$ eine Primzahl.
Aus der Gleichung (*) aus dem Beweis zu Satz 4 folgt auch [12]
$\{a \cdot 1, \ldots, a \cdot (n-1)\} = \{1, \ldots, n-1\}$ (da $a \cdot 0 = 0$)
$\rightsquigarrow 1 \cdot 2 \cdots (n-1) = (a \cdot 1) \cdot (a \cdot 2) \cdots (a \cdot (n-1)) = a^{n-1} \cdot (1 \cdot 2 \cdots (n-1))$
Da $\mathbb{Z}_n$ ein Körper ist, folgt die Behauptung, indem man beide Seiten der Gleichung mit den multiplikativen Inversen von $(n-1), (n-2), \ldots, 2$ multipliziert.

q.e.d. Satz 6

Die Umkehrung gilt ebenfalls:

SATZ 7 (Umkehrung des kleinen FERMAT). $\forall a \in \mathbb{Z}_n \setminus \{0\} : a^{n-1} = 1$ (in $\mathbb{Z}_n$) $\Rightarrow n$ prim

Beweis:

Sei $a \in \mathbb{Z}_n \setminus \{0\}$ beliebig.
$\rightsquigarrow a^{n-1} = 1$
$\rightsquigarrow a \cdot a^{n-2} = 1$

[10] siehe bspw. [**Hei02**], S. 112

[11] Die im Folgenden angegebenen Beweise basieren zu großen Teilen auf [**Weg96**], S. 257-261.

[12] Im Folgenden wird nicht mehr explizit zwischen $+$ und $\oplus$ bzw. $\cdot$ und $\odot$ unterschieden, sondern der Einfachheit halber durchgehend $+$ und $\cdot$ verwendet. Ob die Operatoren sich auf $\mathbb{Z}$ oder $\mathbb{Z}_n$ beziehen, geht aus dem jeweiligen Kontext hervor.

$\rightsquigarrow$ inverses Element von a existiert
$\rightsquigarrow \mathbb{Z}_n$ ist ein Körper
$\overset{\text{Satz 4}}{\rightsquigarrow} n$ prim
q.e.d. Satz 7

Satz 6 und Satz 7 liefern zwar ein Kriterium, um zu entscheiden, ob eine Zahl n prim ist, die direkte Anwendung ist aber nicht sinnvoll, da man, wenn n prim ist, $n-1$ Zahlen potenzieren muss, um dies sicher festzustellen; der Aufwand wäre also größer als beim
Brute-Force-Verfahren aus Abschnitt 3.1.
Deshalb muss man noch etwas weiter gehen:

SATZ 8. n ungerade und prim $\Rightarrow \forall a \neq 0 : a^{\frac{n-1}{2}} = \pm 1$
Beweis:
Sei n ungerade und prim.
$\left(a^{\frac{n-1}{2}} - 1\right)\left(a^{\frac{n-1}{2}} + 1\right) = a^{n-1} - 1 \overset{\text{Satz 6}}{=} 0$
$\rightsquigarrow a^{\frac{n-1}{2}} - 1 = 0 \quad$ oder $\quad a^{\frac{n-1}{2}} + 1 = 0 \quad$ (da $\mathbb{Z}_n$ nach Satz 4 ein Körper ist, da n prim)
$\rightsquigarrow a^{\frac{n-1}{2}} = \pm 1$
q.e.d. Satz 8

Hieraus folgt sofort, dass der Algorithmus immer die richtige Entscheidung trifft, wenn n prim ist, da in diesem Fall immer $b_i = a_i^{\frac{n-1}{2}} = \pm 1$ gilt, somit die if-Bedingung nie erfüllt wird und am Ende nach der for-Schleife *true* ausgegeben wird.
Nun muss noch der Fall, dass n keine Primzahl ist, untersucht werden. Um zu beweisen, dass die Fehlerwahrscheinlichkeit in diesem Fall $\leq \varepsilon$ ist, genügt es zu zeigen, dass bei einem zufällig gewählten $a_i \in \mathbb{Z}_n \setminus \{0\}$ die Wahrscheinlichkeit, dass $a_i^{\frac{n-1}{2}} = \pm 1$ gilt, obwohl n keine Primzahl ist, $\leq \frac{1}{2}$ ist. Eine falsche Entscheidung wird nämlich nur getroffen, wenn bei allen k Tests die if-Bedingung nicht erfüllt wird und somit am Ende *true* ausgegeben wird. Die Wahrscheinlichkeit, dass bei k unabhängig gewählten Zufallszahlen a_i immer $a_i^{\frac{n-1}{2}} = \pm 1$ gilt, obwohl n keine Primzahl ist, beträgt dann maximal $(\frac{1}{2})^k \leq (\frac{1}{2})^{-\log_2 \varepsilon} = \varepsilon$.
Der Beweis wird nun nicht direkt in $\mathbb{Z}_n$ geführt, sondern es wird eine äquivalente Aussage in einem zu $\mathbb{Z}_n$ isomorphen Ring gezeigt. Dazu wird ein Spezialfall des chinesischen Restsatzes verwendet, der den benötigten Ringisomorphismus angibt:

SATZ 9 (chinesischer Restsatz (Spezialfall)). $\mathbb{Z}_n \simeq \mathbb{Z}_p \times \mathbb{Z}_q$, wobei $n = p \cdot q, p$ und q teilerfremd
(Die Addition und Multiplikation in $\mathbb{Z}_p \times \mathbb{Z}_q$ ist komponentenweise definiert.)
Beweis:
Sei $h : \mathbb{Z}_n \to \mathbb{Z}_p \times \mathbb{Z}_q, h(a) = (a \mod p, a \mod q)$.
z.z.: h ist ein Ringisomorphismus, d.h. h ist ein Ringhomomorphismus und h ist bijektiv.
Behauptung: h ist injektiv.
Beweis:

z.z.: $h(a) = h(b) \Rightarrow a = b$
Seien $a, b \in \mathbb{Z}_n$ beliebig (o.B.d.A. $a \geq b$), so dass
$(a \mod p, a \mod q) = (b \mod p, b \mod q)$.
$\rightsquigarrow a \mod p = b \mod p$ und $a \mod q = b \mod q$
$\rightsquigarrow \exists x, y \in \mathbb{N} : a = x \cdot p + b$ und $a = y \cdot q + b$,
wobei $x < q$ und $y < p$ (*), damit $a < n = p \cdot q$
$\rightsquigarrow x \cdot p + b = y \cdot q + b$
$\rightsquigarrow x \cdot p = y \cdot q$
Annahme: $x > 0, y > 0$
p und q sind nach Voraussetzung teilerfremd, das kgV ist also $p \cdot q$.
$\rightsquigarrow \min\{(x, y) \mid x, y > 0 \wedge x \cdot p = y \cdot q\} = (q, p)$
Dies ist ein Widerspruch zu (*).
$\rightsquigarrow x = 0, y = 0$
$\rightsquigarrow a = b$
q.e.d. injektiv
Behauptung: h ist bijektiv.
Beweis:
$|\mathbb{Z}_n| = n = p \cdot q = |\mathbb{Z}_p \times \mathbb{Z}_q| \in \mathbb{N}$ und h ist injektiv.
$\rightsquigarrow h$ ist bijektiv.
q.e.d. bijektiv
Behauptung: h ist ein Ringhomomorphismus.
Beweis:
Seien $a, b \in \mathbb{Z}_n$ beliebig.
$h(a + b) = ((a + b) \mod p, (a + b) \mod q) =$
$((a \mod p) + (b \mod p), (a \mod q) + (b \mod q)) =$
$(a \mod p, a \mod q) + (b \mod p, b \mod q) = h(a) + h(b)$
$h(a \cdot b) =^{\text{analog}}_{\ldots} h(a) \cdot h(b)$
$\rightsquigarrow h$ ist ein Ringhomomorphismus.
q.e.d. Ringhomomorphismus
q.e.d. Satz 9

Als nächstes muss eine Aussage für den Ring $\mathbb{Z}_p \times \mathbb{Z}_q$ gefunden werden, die äquivalent ist zu der ursprüngliche Aussage für $\mathbb{Z}_n$, die bewiesen werden soll. Dies geschieht durch folgendes Lemma, wobei zunächst noch eine sprachliche Vereinfachung eingeführt wird:

DEFINITION 4 (Eulersch).

- $a \in \mathbb{Z}_n$ heißt Eulersch gdw. $a^{\frac{n-1}{2}} = \pm 1$
- $(a_1, a_2) \in \mathbb{Z}_p \times \mathbb{Z}_q$ heißt Eulersch gdw. $(a_1, a_2)^{\frac{n-1}{2}} = \pm(1, 1)$

LEMMA 10. $a \in \mathbb{Z}_n$ Eulersch gdw. $h(a) = (a \mod p, a \mod q) \in \mathbb{Z}_p \times \mathbb{Z}_q$ Eulersch

Beweis:

$$a \in \mathbb{Z}_n \text{ Eulersch} \overset{\text{Def 4}}{\iff} a^{\frac{n-1}{2}} = \pm 1 \overset{\text{Satz 9}}{\iff} h\left(a^{\frac{n-1}{2}}\right) = h(\pm 1) \overset{\text{Satz 9}}{\iff} h(a)^{\frac{n-1}{2}} = \pm h(1)$$

$\Leftrightarrow h(a)^{\frac{n-1}{2}} = \pm(1,1) \overset{\text{Def 4}}{\Longleftrightarrow} h(a) \in \mathbb{Z}_p \times \mathbb{Z}_q$ Eulersch
q.e.d. Lemma 10

Die zu beweisende Aussage für $\mathbb{Z}_n$ kann man nun also auch folgendermaßen formulieren, wobei für n die im Abschnitt 3.4 genannten Einschränkungen (n und $\frac{n-1}{2}$ ungerade) gelten:
Wenn n keine Primzahl ist, beträgt die Wahrscheinlichkeit, dass ein zufällig gewähltes $a \in \mathbb{Z}_n \setminus \{0\}$ Eulersch ist, maximal $\frac{1}{2}$.
Die dazu äquivalente Aussage für $\mathbb{Z}_p \times \mathbb{Z}_q$ ist wegen Lemma 10 quasi identisch:
Wenn n keine Primzahl ist, beträgt die Wahrscheinlichkeit, dass ein zufällig gewähltes $a \in \mathbb{Z}_p \times \mathbb{Z}_q \setminus \{0\}$ Eulersch ist, maximal $\frac{1}{2}$.
Diese Zusammenhänge werden nun beim noch ausstehenden Beweis der ursprünglichen Aussage verwendet:

SATZ 11. $n, \frac{n-1}{2}$ ungerade, n nicht prim $\Rightarrow a^{\frac{n-1}{2}} = \pm 1$ für höchstens die Hälfte der $a \neq 0$

Beweis:

Sei $n, \frac{n-1}{2}$ ungerade, n nicht prim.

n sei zunächst keine Potenz einer Primzahl.
$\rightsquigarrow \exists$ teilerfremde $p, q : n = p \cdot q$
$\overset{\text{Satz 9}}{\rightsquigarrow} \mathbb{Z}_n \simeq \mathbb{Z}_p \times \mathbb{Z}_q$
$\overset{\text{Lemma 10}}{\rightsquigarrow}$ g.z.z.: Höchstens die Hälfte der $(a_1, a_2) \in \mathbb{Z}_p \times \mathbb{Z}_q \setminus \{0\}$ sind Eulersch.
$\forall (a_1, 0) \in \mathbb{Z}_p \times \mathbb{Z}_q : (a_1, 0)$ ist nicht Eulersch, da $(a_1, 0)^{\frac{n-1}{2}} = \left(a_1^{\frac{n-1}{2}}, 0\right) \neq \pm(1,1)$
Betrachte nun (a_1, a_2) und $(a_1, -a_2)$ für $a_2 \neq 0$.
Bemerkung: $(a_1, a_2) \neq (a_1, -a_2)$, da $a_2 \neq -a_2$
(Annahme: $a_2 = -a_2 \overset{a_2 \neq 0}{\rightsquigarrow} a_2 = q - a_2 \rightsquigarrow q = 2a_2 \rightsquigarrow q$ gerade $\rightsquigarrow n = p \cdot q$ gerade.
Widerspruch zur Voraussetzung.)
Fall 1: (a_1, a_2) ist nicht Eulersch.
Fall 2: (a_1, a_2) ist Eulersch.
$\rightsquigarrow (a_1, -a_2)^{\frac{n-1}{2}} = \left(a_1^{\frac{n-1}{2}}, (-a_2)^{\frac{n-1}{2}}\right) \overset{\frac{n-1}{2} \text{ ungerade}}{=} \left(a_1^{\frac{n-1}{2}}, -\left(a_2^{\frac{n-1}{2}}\right)\right) \overset{(a_1, a_2) \text{ Eulersch}}{=} \pm(1,-1)$
$\rightsquigarrow (a_1, -a_2)$ ist nicht Eulersch.
$\overset{\text{Fall 1,2}}{\rightsquigarrow} (a_1, a_2)$ oder $(a_1, -a_2)$ ist nicht Eulersch.
$\rightsquigarrow \mathbb{Z}_p \times \mathbb{Z}_q$ zerfällt in die Menge $\{(a_1, 0) \mid a_1 \in \mathbb{Z}_p\}$, deren Elemente nicht Eulersch sind, und in die zwei-elementigen Mengen $\{(a_1, a_2), (a_1, -a_2)\}$ mit $a_1 \in \mathbb{Z}_p, a_2 \in \mathbb{Z}_q \setminus \{0\}$, wobei jeweils mindestens ein Element nicht Eulersch ist.
$\rightsquigarrow$ Mindestens die Hälfte der Elemente in $\mathbb{Z}_p \times \mathbb{Z}_q$ sind nicht Eulersch.
$\rightsquigarrow$ Mindestens die Hälfte der Elemente in $\mathbb{Z}_p \times \mathbb{Z}_q \setminus \{0\}$ sind nicht Eulersch, da n ungerade.

Betrachte nun den Sonderfall $n = p^e$, wobei p prim, $e > 1$:
Sei $b = 1 - p$.
Nach Lemma 12, das als nächstes gezeigt wird, gilt:

b ist invertierbar und nicht Eulersch.
Sei $a \in \mathbb{Z}_n$ Eulersch.
$\rightsquigarrow (a \cdot b)^{\frac{n-1}{2}} = a^{\frac{n-1}{2}} \cdot b^{\frac{n-1}{2}} \neq \pm 1$, da $a^{\frac{n-1}{2}} = \pm 1$ und $b^{\frac{n-1}{2}} \neq \pm 1$.
$\rightsquigarrow a \cdot b$ ist nicht Eulersch.
Sei $E = \{a \mid a \in \mathbb{Z}_n \wedge a \text{ ist Eulersch}\}, \bar{E} = \{a \cdot b \mid a \in E\}$
$\rightsquigarrow E \subseteq \mathbb{Z}_n, \bar{E} \subseteq \mathbb{Z}_n, \forall a \in E : a$ ist Eulersch, $\forall \bar{a} \in \bar{E} : \bar{a}$ ist nicht Eulersch.
$\rightsquigarrow E \cap \bar{E} = \emptyset, |E| = |\bar{E}|$, da b invertierbar, die Multiplikation mit b also eine Bijektion ist.
$\rightsquigarrow$ Mindestens die Hälfte der Elemente in $\mathbb{Z}_n$ sind nicht Eulersch.
$\rightsquigarrow$ Mindestens die Hälfte der Elemente in $\mathbb{Z}_n \setminus \{0\}$ sind nicht Eulersch, da n ungerade.
q.e.d. Sonderfall
q.e.d. Satz 11

LEMMA 12. Sei p prim, $e > 1, n = p^e, b = 1 - p$. b ist invertierbar und nicht Eulersch.
Beweis:

$$\begin{aligned}(1-p)^{p^e} &= \sum_{k=0}^{p^e} \binom{p^e}{k}(-p)^k && \text{(Binomischer Satz)}\\ &= \sum_{k=0}^{e-1} \binom{p^e}{k}(-p)^k && \text{(da alle Summanden mit höherem Exponenten } k \geq e \text{ offensichtlich}\\ & && \text{durch } n = p^e \text{ teilbar sind und durch } \mod n \text{ wegfallen)}\\ &= 1 + \sum_{k=1}^{e-1} \tfrac{p^e \cdot (p^e-1) \cdots (p^e-k+1)}{k!}(-p)^k\end{aligned}$$

Hilfsbehauptung: Jeder dieser Summanden ist durch p^e teilbar.
Beweis der Hilfsbehauptung:
z.z.: $\forall k < e : \frac{p^e \cdot (p^e-1) \cdots (p^e-k+1)}{k!}(-p)^k \mod p^e = 0$
Sei $k < e$ beliebig.
$\frac{p^e \cdot (p^e-1) \cdots (p^e-k+1)}{k!}(-p)^k = (-1)^k \cdot p^e \cdot \left(\prod_{i=1}^{k-1} \frac{p^e - i}{i} \right) \cdot \frac{p^k}{k}$ (*)
Für jeden Faktor $\frac{p^e-i}{i}, i < k < e$, gilt:
$\exists x \in \mathbb{N}, y \in \mathbb{N} \setminus \{0\}$, wobei y nicht den Faktor p enthält: $i = p^x \cdot y$
$\rightsquigarrow x \leq p^x \leq p^x \cdot y = i < k < e$, also $x < e$
$\rightsquigarrow \frac{p^e-i}{i} = \frac{p^e - p^x \cdot y}{p^x \cdot y} = \frac{p^{e-x} - y}{y}$
$\rightsquigarrow$ Der Faktor p steht (nach dem Kürzen) nicht im Nenner.
Betrachte nun den letzten Faktor von (*), nämlich $\frac{p^k}{k}$:
Nach dem Kürzen steht p nicht im Nenner, da der Faktor p offensichtlich nicht häufiger in k als in p^k enthalten sein kann.

$\rightsquigarrow$ In (*) befindet sich der Faktor p (nach dem Kürzen) nicht im Nenner, aber mindestens e-mal im Zähler, da p^e als (zweiter) Faktor enthalten ist.
$\rightsquigarrow$ (*) ist durch p^e teilbar.
q.e.d. Hilfsbehauptung
$\rightsquigarrow b^n = 1 \quad (\rightsquigarrow b \cdot b^{n-1} = 1 \rightsquigarrow b$ ist invertierbar mit $b^{-1} = b^{n-1})$
$\rightsquigarrow b^{n-1} \neq 1$ (denn sonst würde $b^n = b = 1 - p \neq 1$ gelten)

$\rightsquigarrow \left(b^{\frac{n-1}{2}} - 1\right) \cdot \left(b^{\frac{n-1}{2}} + 1\right) = b^{n-1} - 1 \neq 0$
$\rightsquigarrow \left(b^{\frac{n-1}{2}} - 1\right) \neq 0 \quad \text{und} \quad \left(b^{\frac{n-1}{2}} + 1\right) \neq 0$
$\rightsquigarrow b^{\frac{n-1}{2}} \neq \pm 1$
$\rightsquigarrow b$ ist nicht Eulersch.
q.e.d. Lemma 12

3.7. Komplexitätsanalyse. Zum Schluss soll jetzt noch die Komplexität des stochastischen Algorithmus analysiert werden, um zu zeigen, dass er effizienter als jeder bis jetzt bekannte deterministische Algorithmus ist und deshalb große praktische Bedeutung hat.
Die for-Schleife wird k-mal durchlaufen, wobei jedesmal eine Zufallszahl erzeugt, eine Potenzierung modulo n durchgeführt und ein Vergleich vorgenommen wird.
Die Erzeugung einer Zufallszahl mit der im Abschnitt 2.2 vorgestellten Methode der linearen Kongruenz erfordert im Wesentlichen eine Multiplikation zweier $\log_2 n$-stelliger Zahlen mit Restbildung modulo n. Multiplikation und Division (Restbildung) können mit „Schulmethoden“ in quadratischer Zeit durchgeführt werden, so dass ein Aufwand von $O((\log_2 n)^2)$ entsteht.
Der Hauptaufwand fällt bei der Potenzierung an. Um nicht $\frac{n-1}{2} - 1$ Multiplikationen durchführen zu müssen, geht man folgendermaßen vor:
Sei $(d_m d_{m-1} \dots d_0)_2$ die binäre Darstellung des Exponenten $\frac{n-1}{2}$. Dann gilt $m = \lfloor \log_2 (\frac{n-1}{2}) \rfloor \leq \log_2 n$ und $\frac{n-1}{2} = \sum_{j=0}^{m} d_j 2^j$. Die Potenzierung lässt sich nun durch wiederholtes Quadrieren von a_i realisieren:

$$a_i^{\frac{n-1}{2}} = a_i^{\sum_{j=0}^{m} d_j 2^j} = \prod_{j=0}^{m} a^{d_j 2^j} = \prod_{\substack{j=0 \\ d_j=1}}^{m} a^{2^j}$$

Man muss somit nur m-mal a_i quadrieren und maximal weitere m Multiplikationen durchführen, so dass insgesamt maximal $2m$ Multiplikationen benötigt werden. Nach (oder während) jeder Multiplikation wird dabei der Rest modulo n bestimmt, so dass die Zahlen relativ klein bleiben, es sich also jeweils nur um Multiplikationen zweier $\log_2 n$-stelliger Zahlen mit Restbildung handelt. Somit werden maximal $c \cdot 2m \cdot (\log_2 n)^2$ Schritte benötigt (wobei c eine Konstante ist), so dass man den Aufwand mit $O((\log_2 n)^3)$ abschätzen kann.
Als Letztes muss noch der Vergleich mit ± 1 durchgeführt werden. Dies geht in $O(\log_2 n)$.
Der Gesamtaufwand lässt sich demzufolge mit
$O(k \cdot ((\log_2 n)^2 + (\log_2 n)^3 + (\log_2 n))) = O((\log_2 n)^3)$ abschätzen, wobei man beachten muss, dass k nicht von n, sondern nur von der fest gewählten Fehlerwahrscheinlichkeit ε abhängt.

3.8. Arithmetik. Anstelle der „Schulmethoden“, auf die ich mich bei der Komplexitätsanalyse bezogen habe, kann man effizientere arithmetische Verfahren verwenden. Zum einen kann man die Tatsache ausnutzen, dass man bei der Potenzierung mehrmals eine Restbildung modulo n vornehmen muss. Peter L. MONTGOMERY veröffentlichte 1985 ein

Verfahren[13], bei dem die Restbildung auf einfache Schiebeoperationen reduziert wird, die praktisch in konstanter Zeit durchgeführt werden können. Das Verfahren erfordert zwar eine Vor- und Nachverarbeitung, die sich bei einer einzelnen Multiplikation mit Restbildung nicht lohnen würde, da dieser zusätzliche Aufwand bei einer Potenzierung aber nur einmal auftritt, hat dieser Algorithmus den Vorteil, dass man für die Restbildung nicht $2m$-mal quadratischen Aufwand benötigt.
Somit bleibt nur noch die Multiplikation als „bottle neck“.
Hier kann man nun die KARATSUBA-Multiplikation[14] anwenden, die auf einem geschickten Divide&Conquer-Ansatz basiert und nur $O((\log_2 n)^{\log_2 3})$ Schritte benötigt (wobei $\log_2 3 \approx 1,58$), so dass die Laufzeit des gesamten Algorithmus auf $O((\log_2 n)^{2,58})$ reduziert wird.
Es gibt zwar Multiplikationsalgorithmen, die in noch günstigeren Komplexitätsklassen liegen – wie bspw. die FFT-Multiplikation[15], die auf der schnellen FOURIER-Transformation beruht und nur $O(\log_2 (n) \cdot \log (\log_2 n))$ Schritte benötigt –, diese sind aber aufgrund der hohen Konstanten erst für Zahlen mit deutlich mehr Stellen als in der Kryptographie üblich sinnvoll.

4. Fazit

Zum einen hat sich gezeigt, dass zahlentheoretische Erkenntnisse, die auf den ersten Blick aus praktischer Sicht wertlos erscheinen, eine wichtige Grundlage für die Lösung des Entscheidungsproblems, ob eine gegebene Zahl eine Primzahl ist, darstellen und dass die Lösung für die Kryptographie von großem praktischen Interesse ist.
Zum anderen kann man festhalten, dass Varianten des vorgestellten stochastischen Primtest-Algorithmus in der Praxis eine große Bedeutung haben, obwohl im August 2002 ein deterministisches Verfahren entdeckt wurde, das das Problem in polynomialer Zeit löst, da der stochastische Algorithmus sehr effizient ist und in kurzer Rechenzeit große Primzahlen ermitteln kann mit einer Fehlerwahrscheinlichkeit, die deutlich kleiner ist als die Wahrscheinlichkeit, dass man 10-mal hintereinander im Lotto gewinnt.

[13] siehe bspw. [**Knu98**], S. 284
[14] siehe bspw. [**AH00**], Kap. 11.2 oder [**Knu98**], S.294f
[15] siehe bspw. [**AH00**], Kap. 11.3 oder [**Sed92**], Kap. 41

Literaturverzeichnis

[AH00] Jörg Arndt and Christoph Haenel, *Pi : Algorithmen, Computer, Arithmetik*, 2nd ed., Springer, 2000.

[AKS02] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in P*, Tech. report, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, August 2002.

[BG92] M. Bellare and O. Goldreich, *Papers on zero-knowledge*, 1992.

[BOOGH+90] Michael Ben-Or, Shafi Goldwasser Oded Goldreich, Johan Hastad, Joe Kilian, Silvio Micali, and Phillip Rogaway, *Everything provable is provable in zero-knowledge*, Advances in Cryptography — Crypto'88 (S. Goldwasser, ed.), Lecture Notes in Computer Science, Springer, 1990, pp. 37–56.

[BSW98] Albrecht Beutelspacher, Jörg Schenk, and Klaus-Dieter Wolfenstetter, *Moderne Verfahren der Kryptographie*, 3rd ed., Mathematik, Vieweg, 1998.

[Cha88] David Chaum, *The dining cryptographers problem: Unconditional sender and recipient untraceability*, Journal of Cryptology **1** (1988), no. 1.

[Die93] Roland Dieterich, *SmartCards - Grundlagen, Technik, Sicherheitsaspekte*, Ausarbeitung, Fachbereich Informatik, Universität Koblenz, http://www.fi t.fraunhofer.de/ cactus/smartcards.html, 1993.

[DK02] Hans Delfs and Helmut Knebl, *Introduction to Cryptography*, 1st ed., Springer, 2002.

[GB90] Ivan Bjerre Damgaard Gilles Brassard, *ppractical ip≪= ma*, Advances in Cryptography — Crypto'88 (S. Goldwasser, ed.), Lecture Notes in Computer Science, Springer, 1990, pp. 580–582.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson, *Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems*, Journal of the Association for Computing Machinery **38** (1991), no. 1, 691–729.

[Gol01] Oded Goldreich, *Foundations of cryptography:basic tools*, 1st ed., Cambridge University Press, 2001.

[Hei02] Stefan Heinrich, *Stochastische Algorithmen*, Skript, Fachbereich Informatik, Universität Kaiserslautern, 2002.

[HW58] G. H. Hardy and E. M. Wright, *Einführung in die Zahlentheorie*, R. Oldenbourg, München, 1958.

[Knu98] Donald E. Knuth, *The Art of Computer Programming*, 3rd ed., vol. 2, Addison-Wesley, 1998.

[Kow02] Prof. Dr. W. Kowalk, *Rechnernetze*, Skript, Fachbereich Informatik, Universität Oldenburg, http://einstein.informatik.uni-oldenburg.de/rechnernetze/zero-knowledge-problem.htm, 2002.

[Kra86] Evangelos Kranakis, *Primality and Cryptography*, Wiley-Teubner, 1986.

[Kre02] Bernd Kreussler, *Mathematik für Informatiker 1*, Tech. report, Universität Kaiserslautern, 2002.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan, *Algebraic methods for interactive proof systems*, Journal of the Association for Computing Machinery **39** (1992), no. 4, 859–868.

[Pap94] Christos H. Papadimitriou, *Computational complexity*, 1st ed., Addison Wesley, 1994.

[Sch96] Bruce Schneier, *Angewandte Kryptographie*, Addison-Wesley, 1996.

[Sed92] Robert Sedgewick, *Algorithmen*, Addison-Wesley, 1992.

[Sha92] Adi Shamir, *Ip = pspace*, Journal of the Association for Computing Machinery **39** (1992), no. 4, 869 – 877.

[She92] A. Shen, *Ip = pspace: Simplifi ed proof*, Journal of the Association for Computing Machinery **39** (1992), no. 4, 878 – 880.

[Ski98] Steven S. Skiena, *The Algorithm Design Manual*, Springer, 1998.

[SS77] R. Solovay and V. Strassen, *A fast Monte-Carlo test for primality*, SIAM Journal on Computing **6** (1977), 84–85.

[Sti95] Douglas R. Stinson, *Cryptography*, 2nd ed., Addison-Wesley, 1995.

[Tie01] Veith Tiemann, *Asymmetrische/moderne Kryptographie – Ein interaktiver Überblick*, Skript, Fakultät für Wirtschaftswissenschaften, Universität Bielefeld, http://www.wiwi.uni-bielefeld.de/StatCompSci/lehre/material_spezifi sch/statalg00/rsa/node20.html, 2001.

[Ung00] Luise Unger, *Lineare Algebra I*, Skript, Fachbereich Mathematik, FernUniversität Hagen, 2000.

[Weg96] Ingo Wegener (ed.), *Highlights aus der Informatik*, Springer, 1996.