Zero - Knowledge Beweise
Theorie und Anwendung

Diplomarbeit an der Universität Ulm
Fakultät für Informatik

vorgelegt von:

Andreas Wutzke

1. Gutachter: *Priv. Doz Dr. J. Köbler*

2. Gutachter: *Prof. Dr. U. Schöning*

1996

Einleitung und Zusammenfassung

Es war ein Meilenstein in der Geschichte der modernen Kryptographie, als 1985 die Wissenschaftler Goldwasser, Micali und Rackoff in ihrer Arbeit [GMR85] die Interaktiven Beweissysteme und darauf aufbauend die Zero-Knowledge Interaktiven Beweissysteme eingeführt haben.

Durch die Veröffentlichung wurde ein richtiger Forschungsboom auf diesem Gebiet ausgelöst. Kurz nach der Einführung der Zero-Knowledge Protokolle waren nur wenige Beispiele für solche Protokolle bekannt. Etwa die Protokolle für die Sprachen QR_n und QNR_n. Die Vermutung war zunächst, daß es nur wenige Probleme gibt, die sich für die Konstruktion eines Zero-Knowledge Protokolls eignen.

Es war daher ein überraschendes Resultat, als Goldreich, Micali und Wigderson 1986 in ihrer Arbeit [GMW91] zeigen konnten, daß jede Sprache in der riesigen Klasse NP einen Zero-Knowledge Beweis besitzt. Dieses bedeutende Resultat spielt auch in dieser Arbeit eine wichtige Rolle.

Damit war die Tür zu zahlreichen Anwendungen aufgestoßen. Inzwischen erkannte man, daß die ersten formalen Definitionen von Zero-Knowledge aus [GMR85] Schwächen hatten, aufgrund derer der Einsatz von Protokollen, die dieser Definition genügen, in der Praxis zweifelhaft erscheinen lassen. Besonders unbefriedigend ist bei der ursprünglichen Definition, daß sie nicht unter sequentieller Komposition abgeschlossen ist.

Es wurden stärkere Definitionen der Zero-Knowledge Eigenschaft gebraucht. Unter den ersten, die eine stärkere Definition vorgeschlagen haben, waren Tompa und Woll in [TW87]. Goldreich und Oren haben in [GO94] gezeigt, daß die Klasse von Protokollen, die diese Definition erfüllen, unter sequentieller Komposition abgeschlossen ist.

Jedoch ist die Klasse von Protokollen, welche dieser strengeren Definition genügen, nicht unter paralleler Komposition abgeschlossen.

Mittlerweile wurde auch eine Erweiterung der Interaktiven Beweissysteme untersucht, bei der der Verifier mit mehreren Provern kommunizieren darf. Dieses Modell heißt Multi-Prover Interaktives Beweissystem und wurde in [BGKW88] eingeführt.

Inzwischen gibt es eine Vielfalt von verschiedenen Definitionen des Zero-Knowledge Begriffs und einiger verwandter Konzepte.

Ziel dieser Diplomarbeit ist es, sich gründlich mit diesem Themenbereich vertraut zu machen und eine Auswahl von wichtigen Resultaten verständlich darzustellen. Inzwischen hat sich eine sehr große Anzahl von interessanten Resultaten in diesem Bereich angesammelt, welche nicht alle innerhalb der begrenzten Ressourcen einer Diplomarbeit darstellbar sind. Es mußte eine subjektive Auswahl getroffen werden.

Ich möchte an dieser Stelle PD Dr. J. Köbler und Prof. Dr. U. Schöning für die Begutachtung dieser Arbeit danken. Ebenso danke ich Dipl.-Inform. C. Karg für viele kleine technische Latex Tips, die bei der Erstellung einer solchen Arbeit sehr nützlich sind. Mein besonderer Dank gilt PD Dr. J. Köbler für die vielen wichtigen und anregenden Hinweise, die dazu führten, daß wichtige Punkte nicht unberücksichtigt blieben.

Inhaltsverzeichnis

Kapitel 1

Aufbau der Arbeit

- In **Kapitel 2** wird der Leser mit einigen grundlegenden Definitionen und Bezeichnungen vertraut gemacht. Um dieses Kapitel auf ein erträgliches Maß zu reduzieren, werden von einem Leser die Kenntnisse eines „vordiplomierten Informatikers“ vorausgesetzt. Manche sehr häufig benutzte Begriffe werden allerdings trotzdem kurz eingeführt.

- In **Kapitel 3** werden zunächst die Begriffe Beweis und Beweissystem informal diskutiert und dann für den Fall des Interaktiven Beweissystems formal definiert. Dazu führen wir geeignete Rechenmodelle ein. Durch ein anschauliches Beispiel werden die Definitionen erläutert.

- **Kapitel 4** ist das umfangreichste Kapitel dieser Arbeit. Wir versuchen zunächst eine intuitive Vorstellung von dem Zero-Knowledge Begriff zu bekommen. Diese Vorstellung wird in den folgenden formalen Definitionen mehr oder weniger eingefangen. Wir stellen in diesem Kapitel unterschiedliche „Grade“ von Zero-Knowledge dar und vergleichen verschiedene Definitionen von Zero-Knowledge. Dabei wurde darauf geachtet, theoretische Resultate zu behandeln, die wichtige Auswirkungen auf die Anwendung von Zero-Knowledge Beweisen haben.

- In **Kapitel 5** werden die zwei klassischen Protokolle für die Sprachen GI und GNI behandelt, um die Techniken aufzuzeigen, mit denen von Protokollen nachgewiesen wird, daß sie die Zero-Knowledge Eigenschaft haben.

- In **Kapitel 6** wird aufbauend auf den Beweis von Goldreich, Micali und Wigderson für ein anderes NP-vollständiges Problem gezeigt, daß dieses ein Zero-Knowledge Protokoll besitzt. Daraus folgt sofort, daß alle Sprachen in der Klasse NP ein Zero-Knowledge Protokoll besitzen. Als NP-vollständiges Problem wurde das Hamiltonkreis-Problem ausgewählt. Damit haben wir das bedeutende [GMW91] Resultat nachvollzogen.

- **Kapitel 7** beschäftigt sich mit dem erweiterten Modell mit mehreren Provern. Interessant (auch für Anwendungen) ist bei diesem Modell, daß hier meistens auf kryptographische Annahmen verzichtet werden kann, solange es sichergestellt ist, daß die Prover während des Protokolls nicht miteinander kommunizieren können.

- Der **Anhang** enthält noch ein Resultat, das in Kapitel 7 benötigt wird.

Kapitel 2

Grundlagen

2.1 Grundlegende Definitionen

Dieser Abschnitt soll einige sehr elementare Definitionen und Bezeichnungen, die in dieser Arbeit verwendet werden einführen.

Eine endliche nichtleere Menge nennen wir *Alphabet*. Die Elemente eines Alphabets Σ heißen *Symbole*. Ist ein Alphabet Σ gegeben, so bezeichnen wir mit Σ^* die Menge aller endlichen Folgen, die sich durch Konkatenation von Symbolen aus Σ bilden lassen. Eine solche Folge von Symbolen nennen wir ein *Wort*. Die Menge Σ^* enthält auch das leere Wort ϵ, welches aus null Symbolen besteht. Die Länge eines Wortes x aus Σ^* bezeichnen wir mit $|x|$. Bei einer Menge M bezeichnet $|M|$ die Kardinalität der Menge M. Eine Teilmenge S von Σ^* nennen wir eine Sprache über Σ.

Definition 2.1 (Polynom)
Ein Polynom ist eine Funktion $p : \mathbf{N} \rightarrow \mathbf{N}$ *der Form*

$$p(n) = \sum_{i=0}^{k} a_i n^i \qquad \textit{mit } a_i, k \in \mathbf{N}$$

Graphen

Ein *Graph*, bezeichnet mit $G = (N, K)$, besteht aus einer endlichen Menge N von Punkten, diese bezeichnet man als *Knoten* und einer endlichen Menge K von Paaren solcher Knoten. Die Menge N hat in unseren Betrachtungen immer die Form $N = \{1, \ldots, n\}$

Ist ein Paar $(u, v) \in K$, so bedeutet dies, daß die Knoten u und v durch eine *Kante* verbunden sind.

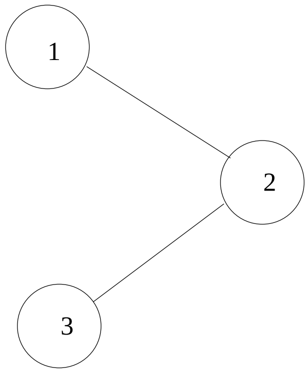

Abbildung 2.1: Beispiel-Graph

Beispiel eines Graphen: $G = (N, K) \quad \text{mit} \quad N = \{1, 2, 3\} \quad \text{und} \quad K = \{(1, 2), (2, 3)\}$.

Den Knoten werden keine Koordinaten zugeordnet. Deshalb werden unterschiedlich große Graphen, in bezug auf die Kantenlänge, nicht unterschieden. Stattdessen ist die Anzahl der Knoten ein Maß für die Größe eines Graphen.

In einem gerichteten Graphen gilt zusätzlich die Forderung, daß die Menge K des Graphen aus geordneten Paaren besteht. Beispiel eines gerichteten Graphen:
$G = (N, K) \quad \text{mit} \quad N = \{1, 2, 3\} \quad \text{und} \quad K = \{(1, 2), (2, 3), (3, 1)\}$.

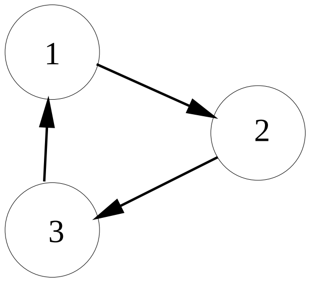

Abbildung 2.2: Gerichteter Graph

Ein *Pfad* in einem Graph G ist eine Folge $u_1, u_2, \ldots, u_k$, $k \geq 1$, so daß für jedes i, $1 \leq i \leq k$, (u_i, u_{i+1}) eine Kante in der Menge K des Graphen G ist. Die Länge des Pfades ist die Anzahl der Kanten entlang des Pfades, also gleich k-1. Einen Pfad in einem Graphen G von Knoten u_1 nach Knoten u_1 mit einer Länge größer als 1 nennt man einen *Kreis* (oder Zyklus). Einen Graphen G nennt man *azyklisch* falls er keinen Kreis enthält. Existiert von jedem Knoten des Graphen ein Pfad zu jedem anderen Knoten des Graphen, so heißt der Graph *zusammenhängend*. Die Anzahl der Knoten, die durch eine Kante mit dem Knoten u verbunden sind, nennt man den *Grad* des Knoten u. Bei einem gerichteten Graphen kann man darüberhinaus zwischen *Eingangsgrad* und *Ausgangsgrad* eines Knoten unterscheiden. Ein Knoten u in einem gerichteten Graph G, der einen Eingangsgrad = 0 besitzt, heißt *Eingangsknoten*, und ein Knoten mit Ausgangsgrad = 0 heißt dementsprechend *Ausgangsknoten*.

Boolescher Schaltkreis

Ein Boolescher Schaltkreis ist ein azyklischer, zusammenhängender, gerichteter Graph. Die Eingangsknoten heißen *Eingaben* und sind mit einer Variablen oder einer Konstanten $\in \{0, 1\}$ markiert. Die Knoten mit Eingangsgrad $k > 0$ heißen *Gatter* und sind mit einer k-stelligen Boolschen Funktion $\{0, 1\}^k \rightarrow \{0, 1\}$ markiert. Der Eingangsgrad eines Knoten heißt *Fan-in* und der Ausgangsgrad heißt *Fan-out* des Knotens. Der Ausgangsknoten des Graphen heißt *Ausgang.* Am Ausgang ergibt sich die durch den Boolschen Schaltkreis berechnete Funktion. Die Größe eines Schaltkreises ist die Anzahl seiner Gatter, die Tiefe ist die Länge des längsten Weges von einer Eingabe zur Ausgabe.

2.2 Grundlagen der Zahlentheorie

Für das Verständnis einiger in dieser Arbeit vorgestellten Protokolle ist es wichtig die Grundlagen der Zahlentheorie zu kennen. In diesem Abschnitt werden die wichtigsten Resultate und Bezeichnungen kurz vorgestellt. Kurze Zusammenfassungen der Zahlentheorie findet man auch in [BSW95, Seite 113 ff] und in [Kob94, Seite 31 ff].

Grunddefinitionen

Mit $\mathbf{N} = \{1, 2, 3, ...\}$ bezeichnen wir die Menge der *natürlichen Zahlen* und mit
$\mathbf{Z} = \{..., -2, -1, 0, 1, 2, ...\}$ die Menge der *ganzen Zahlen.*
Eine ganze Zahl a teilt eine ganze Zahl b, bezeichnet mit $a \mid b$, wenn eine ganze Zahl k existiert, so daß gilt $a \cdot k = b$.
Die Zahl b ist in diesem Fall ein Vielfaches der Zahl a. Da $a \cdot 0 = 0 \; \forall a \in$ Z gilt, teilt jede ganze Zahl die Zahl 0. Die Zahl a nennt man *Teiler* von b. Es gilt $a \mid b \Leftrightarrow -a \mid b$. Gibt man die Teiler einer ganzen Zahl an, beschränkt man sich darauf, die positiven Teiler anzugeben.

Seien a, b, d ganze Zahlen und gilt $d \mid a$ sowie $d \mid b$, so nennt man d einen *gemeinsamen Teiler* von a und b. Mit d = $\mathbf{ggT}(a, b)$ bezeichnen wir den größten gemeinsamen Teiler von a und b. Der **ggT** von a und b läßt sich effektiv mit dem *Euklidschen Algorithmus* berechnen. Gilt für zwei ganze Zahlen a, b $\mathbf{ggT}(a, b) = 1$, dann nennt man a und b *teilerfremd.*

Jede ganze Zahl b besitzt die Teiler 1 und b. Diese nennt man deshalb die *trivialen Teiler* von b. Alle nicht trivialen Teiler von b nennt man *Faktoren* von b. Die Faktoren von 15 sind beispielsweise 3 und 5.

Man unterscheidet in der Zahlentheorie zwischen *Primzahlen* und *zusammengesetzten Zahlen.* Primzahlen sind natürliche Zahlen, die nur die trivialen Teiler besitzen.

Modulare Arithmetik

Dividiert man eine ganze Zahl a durch eine positive ganze Zahl n, wobei n kein Teiler von a ist, so bekommt man einen *Rest r*.
Für den Rest gilt : $0 \leq$ r $<$ n. Wenn $n \mid a$, dann ist r = 0.
Falls für zwei ganze Zahlen a, b gilt

$$(\text{ a mod n }) = (\text{ b mod n })$$

sagen wir a ist *kongruent* b modulo n und schreiben dafür $a \equiv_n b$. Durch die Äquivalenzrelation $a \equiv_n b$ wird $\mathbf{Z}$ in n Äquivalenzklassen unterteilt. Die Äquivalenzklasse, die die ganze Zahl a enthält, ist

$$[a]_n = \{a = k \cdot n \mid k \in \mathbf{Z}\}$$

Die Menge aller Äquivalenzklassen ist dann

$$Z_n = \{[a]_n \mid 0 \leq a \leq n-1\}$$

Die Addition, Subtraktion und Multiplikation auf Z_n ist wie folgt definiert :

$$[a]_n + [b]_n = [a+b]_n$$

$$[a]_n - [b]_n = [a-b]_n$$

$$[a]_n \cdot [b]_n = [a \cdot b]_n$$

Zur Vereinfachung schreiben wir auch a $\in Z_n$ und meinen damit $[a]_n \in Z_n$. Im folgenden bezeichnen wir mit $\oplus$ die modulare Addition, mit $\ominus$ die modulare Subtraktion und mit $\otimes$ die modulare Multiplikation. $(Z_n, \oplus)$ ist eine *kommutative Gruppe.*
$(Z_n, \otimes)$ bildet keine Gruppe, denn nicht alle Elemente von Z_n haben ein inverses Element bzgl. der modularen Multiplikation. Nur die Elemente von Z_n, die teilerfremd zu n sind, haben bzgl. der modularen Multiplikation ein inverses Element. Wir schränken Z_n auf diese Elemente ein und erhalten

$$Z_n^* = \{[a]_n \mid ggT(a,n) = 1\}$$

Sei $d = ggT(a,b)$, dann gibt es ganze Zahlen s und t mit $d = sa + tb$. Diese Darstellung bezeichnet man als *Vielfachsummendarstellung* des größten gemeinsamen Teilers.

Auf Z_n^* eingeschränkt gilt die Gleichung : $ggT(a,n) = 1 = sa + tn$. Die Zahlen s und t liefert der *Erweiterte Euklidsche Algorithmus* (siehe dazu [BSW95], Seite 115). Das multiplikative Inverse a^{-1} von a $\in Z_n^*$ ergibt sich dann zu a^{-1} = s mod n.

Die *Eulersche φ-Funktion* liefert bei n als Argument, die Anzahl der zu n teilerfremden Zahlen, welche kleiner oder gleich n sind.
Damit ist $\varphi(n) = |Z_n^*|$. Für eine Primzahl p ist $\varphi(n) = p - 1$. Ist n das Produkt von zwei Primzahlen p und q, so ist $\varphi(n) = (p-1)(q-1)$. Sind die Primzahlen p und q unbekannt ist es genauso schwer $\varphi(n)$ zu berechnen wie n zu faktorisieren. Die bisher besten Algorithmen zur Faktorisierung einer großen zusammengesetzten Zahl schaffen es nicht, in Polynomialzeit (in der Länge der Zahl) diese in Faktoren zu zerlegen (siehe [BSW95, Seite 119]).

Der Chinesische Restsatz

Hat man ein modulares Gleichungssystem gegeben und interessiert sich für eine Gesamtlösung des Systems, so läßt sich diese, unter bestimmten Umständen, mit dem *Chinesischen Restsatz* berechnen (siehe dazu [BSW95, Seite 115] oder [Kob94, Seite 35]).

Quadratische Reste

Eine Zahl a $\in Z_n^*$ heißt *quadratischer Rest modulo n*, falls es ein b $\in Z_n^*$ gibt mit $b^2 \equiv_n a$. Wenn a kein quadratischer Rest ist, dann heißt a *quadratischer Nichtrest modulo n*.
Wir fassen alle quadratischen Reste in Z_n^* zur Menge QR_n zusammen, ensprechend die quadratischen Nichtreste zur Menge QNR_n. Das Element b bezeichnet man als *Quadratwurzel* von a *modulo n*. Ist n eine Primzahl, dann besitzt ein Element aus Z_n^* entweder keine Quadratwurzel oder genau zwei Quadratwurzeln modulo n. Falls n dagegen eine zusammengesetzte Zahl ist, welche k verschiedene Primfaktoren besitzt, dann hat ein Element aus QR_n genau 2^k Quadratwurzeln modulo n.

Für die natürlichen Zahlen gibt es effektive Algorithmen um Quadratwurzeln zu berechnen. Wie schwer ist es nun, von einem a $\in QR_n$ die Quadratwurzel zu berechnen? Im Fall, daß n eine Primzahl ist, gibt es ebenfalls effektive Algorithmen. Ist allerdings n eine aus zwei großen unbekannten Primzahlen p und q zusammengesetzte Zahl, so ist es im allgemeinen schwer (d.h. nicht in Polynomialzeit, in der Länge von n), eine Quadratwurzel von a modulo n zu bestimmen. Wir wollen nun zeigen, daß es genau so schwierig ist, eine Quadratwurzel modulo n zu berechnen wie dieses n zu faktorisieren.

Hat man die Primfaktorzerlegung pq von n gegeben, dann läßt sich eine Quadratwurzel von a $\in QR_n$ modulo n berechnen, indem man eine Quadratwurzel von a modulo p und eine Quadratwurzel von a modulo q berechnet. Diese lassen sich mit dem Chinesischen Restsatz zu einer Quadratwurzel modulo n zusammensetzen.

Kennt man andererseits von demselben a $\in QR_n$ zwei Quadratwurzeln modulo n (die nicht die additiven Inversen voneinander sind), so reicht das, um n zu faktorisieren.
Denn falls

$$s^2 \equiv_n t^2 \text{ und } s \not\equiv_n \pm t$$

Dann ist

$$g = ggT(s \oplus t, n)$$

ein echter Faktor von n.

Beweis :
Nach Definition ist g ein Teiler von n.
So bleibt zu zeigen, daß g kein trivialer Teiler von n ist.

1. zu zeigen $g \neq n$:

Für zwei Zahlen s, t aus Z_n^* gilt immer: $s \oplus t \ < n$.
Da für zwei beliebige Zahlen a, b, beide ungleich 0, immer gilt, daß der ggT(a,b) eine Zahl

zwischen 1 und $\min(|a|, |b|)$ ist, kann der ggT$(s \oplus t, n)$ nur n sein, falls $s \oplus t$ kongruent 0 modulo n ist.
Nach Voraussetzung ist aber $s \not\equiv_n \pm t \;\Rightarrow\; s + t \not\equiv_n 0 \;\Rightarrow\; g \neq n$

2. zu zeigen $g \neq 1$:

n teilt 0 $\Rightarrow$ n teilt $s^2 - t^2 = (s+t)(s-t)$.
Falls $g = ggT(s \oplus t, n) = 1$, dann muß n $(s \ominus t)$ teilen. Das steht aber im Widerspruch zu der Voraussetzung $s \not\equiv_n \pm t \;\Rightarrow\; g \neq 1$.

$\square$

Wählt man die Primzahlen p und q so, daß $p \equiv_4 q \equiv_4 3$ gilt, dann nennt man
n = pq eine *Blum Integer*. Man verwendet gerne Blum Integers, weil man dann besonders einfach Quadratwurzeln modulo p und modulo q berechnen kann.
Die Wurzeln von a $\in QR_p$ sind dann:

$$w_1 = a^{\frac{p+1}{4}} \qquad\qquad w_2 = p - w_1$$

Entsprechend berechnet man die Quadratwurzel von a modulo q. Diese Ergebnisse setzt man mit Hilfe des Chinesischen Restsatzes zu vier Quadratwurzeln modulo n zusammen.

Erkennen von quadratischen Resten

Wenn die Primfaktorzerlegung einer zusammengesetzten Zahl n nicht bekannt ist, ist es ein schwieriges Problem, die Quadratwurzel modulo n eines Elements a aus QR_n zu berechnen. Eine andere wichtige, damit zusammenhängende Fragestellung ist: Wie schwierig ist es, von einer Zahl a $\in Z_n^*$ zu bestimmen, ob sie in QR_n oder in QNR_n enthalten ist? Also ob sie quadratischer Rest ist.

Falls n eine Primzahl ist, dann liefert das *Legendresymbol* die Antwort.

Definition 2.2 (Legendresymbol)
Das Legendresymbol $\mathcal{L}(x,p)$ ist für Primzahlen p und Zahlen $x \in Z_p^*$ wie folgt definiert:

$$\mathcal{L}(x,p) := \begin{cases} 1 & \text{, falls } x \in QR_p \\ -1 & \text{, falls } x \in QNR_p \end{cases}$$

Das Legendresymbol $\mathcal{L}(x,p)$ kann mit der Formel

$$x^{\frac{p-1}{2}} \equiv_p 1$$

effektiv berechnet werden (siehe [Kob94, Seite 39]).

Definition 2.3 (Jacobisymbol)
Das Jacobisymbol $\mathcal{J}(x,n)$ ist für alle Zahlen n und $x \in Z_n^*$ definiert, wir betrachten aber nur den Spezialfall $n = pq$:

$$\mathcal{J}(x,n) := \mathcal{L}(x,p) \cdot \mathcal{L}(x,q).$$

Auch das Jacobisymbol ist effektiv berechenbar (siehe [Kob94, Seite 42]).
Ist $\mathcal{J}(x,n)$ = -1, dann ist x $\in QNR_n$. Im Fall $\mathcal{J}(x,n) = 1$ läßt sich nicht sagen, ob x $\in QR_n$ ist. Da $\mathcal{J}(x,n) = 1$ auch dadurch möglich ist, wenn beide Legendresymbole -1 sind. Gerade für diese Elemente von Z_n^* mit Jacobisymbol = 1 ist es schwer zu erkennen, ob sie quadratische Reste sind. Wegen dieser wichtigen Eigenschaft fassen wir diese Elemente zu einer Menge zusammen:

$$Z_n^*(+1) = \{x \mid x \in Z_n^* \text{ und } \mathcal{J}(x,n) = 1\}$$

2.3 Wahrscheinlichkeitsrechnung

In dieser Arbeit wird viel mit Zufallsvariablen gearbeitet. Es sollen deshalb kurz die wichtigsten Resultate kompakt und ohne Beweis dargestellt werden.

Bei einem Zufallsexperiment bezeichnet man mit Ω die Menge der möglichen Ausfälle ω. Jede Teilmenge E von Ω heißt *Ereignis*. Die Abbildung (Funktion), die jedem Element $\omega \in \Omega$ eine reelle Zahl zuordnet, heißt Zufallsvariable. Zufallsvariablen bezeichnet man üblicherweise mit Großbuchstaben z.B. X, Y, Z. Also X: $\Omega \rightarrow \mathbf{R}$.

Die Mengen $\{\omega \in \Omega \mid X(\omega) = a\}$ und $\{\omega \in \Omega \mid X(\omega) \leq a\}$ kürzt man mit $X = a$ bzw. $X \leq a$ ab.

Man sagt auch, die Zufallsvariable nimmt einen Wert a an und meint damit, daß sie ein $\omega \in \Omega$ auf ein $a \in \mathbf{R}$ abbildet. Interessant ist nun, mit welcher Wahrscheinlichkeit bestimmte Werte von der Zufallsvariablen angenommen werden. Z.B. Wie groß ist die Wahrscheinlichkeit, daß die Zufallsvariable den Wert $a \in \mathbf{R}$ annimmt? Diese Wahrscheinlichkeit wird mit $\mathbf{Pr}[\mathrm{X} = \mathrm{a}]$ bezeichnet.

Definition 2.4 (Wahrscheinlichkeitsverteilung)

Eine Abbildung, die jedem Wert $a \in \mathbf{R}$ die Wahrscheinlichkeit des Ereignisses $X = a$ zuordnet, heißt Wahrscheinlichkeitsverteilung von X.

Zwei der wichtigsten Kenngrößen einer Zufallsvariablen ist deren *Erwartungswert* und *Varianz*. Der Erwartungswert einer Zufallsvariablen X ist der zu erwartende Mittelwert von X bei einer 'langen' Zufallsversuchsreihe. Die Varianz, einer Zufallsvariablen, ist die mittlere quadratische Abweichung vom Erwartungswert. Diese Kenngrößen berechnen sich wie folgt:

$$E(X) = \sum_{\omega \in \Omega} P(\omega)\, X(\omega) \tag{2.1}$$

$$V(X) = \sum_{\omega \in \Omega} P(\omega)\, (X(\omega) - E(X))^2. \tag{2.2}$$

In dieser Arbeit, wird auch eine Modifikation einer Zufallsvariable verwendet. Diese Zufallsvariablen bilden statt auf reelle Werte auf Strings aus Σ^* ab. Als Alphabet dient dabei das Binäralphabet, also $\Sigma = \{0, 1\}$.

Binomialverteilung

Zufallsversuche mit nur zwei möglichen Ausfällen heißen *Bernoulli-Versuche*. Die zwei Ausfälle bezeichnet man auch mit *Treffer (1)* und *Fehlschlag (0)*. Die n-fache Ausführung eines Bernoulli-Versuchs heißt *Bernoulli-Kette der Länge n.* Generell läßt sich jeder Zufallsversuch zu einem Bernoulli-Versuch 'vergröbern', indem man nur noch das Auftreten eines Ereignisses E, für das man sich interessiert, notiert. Alle anderen Ausfälle werden zu dem Gegenereigniss $\overline{E}$ zusammengefaßt. Z.B. Beim Würfeln interessiert man sich für das Ereigniss $E = \{6\}$. Alle anderen möglichen Ausfälle werden zu dem Gegenereigniss $\overline{E} = \{1, 2, 3, 4, 5\}$ zusammengefaßt.

Die Ausfälle ω einer Bernoulli-Kette der Länge n sind n-Tupel der folgenden Form: $\omega = (\alpha_1, \alpha_2, \ldots, \alpha_n)$

Ist beim i-ten Versuch ein Treffer gelungen, so ist $\alpha_i = 1$, bei einem Fehlschlag ist $\alpha_i = 0$.

Sei P die Trefferwahrscheinlichkeit in einem Bernoulli-Versuch, dann ist die Wahrscheinlichkeit für einen Fehlschlag $1 - p$. In der dazugehörigen Bernoulli-Kette der Länge n hat jeder Ausfall ω mit übereinstimmender Trefferzahl k die gleiche Wahrscheinlichkeit $P(\omega_k)$, für $0 \leq k \leq n$.
Diese Wahrscheinlichkeit läßt sich folgendermaßen berechnen:

$$P(\omega_k) = p^k \ \cdot \ (1-p)^{n-k}$$

Wegen der k Treffer tritt k mal die Trefferwahrscheinlichkeit p als Faktor auf. Die restlichen n-k Versuche waren Fehlschläge mit jeweils Wahrscheinlichkeit $1 - p$.

Sei X die Zufallsvariable, die jedem Ausfall ω die Anzahl der Treffer in ω zuordnet. Das Ereigniss $X = k$ besteht aus allen Ausfällen mit k Treffern.

Definition 2.5 (Binomialverteilte Zufallsvariable)

Bei einer Bernoulli-Kette der Länge n sei p die Trefferwahrscheinlichkeit. Gibt die Zufallsvariable X die Anzahl der Treffer an, so gilt für $0 \leq k \leq n$

$$P[X = k] \ = \ \binom{n}{k} \ \cdot p^k \ \cdot \ (1-p)^{n-k}.$$

Jede Zufallsvariable mit einer solchen Wahrscheinlichkeitsverteilung heißt **binomialverteilt** *mit den* Parametern n *und* p.

Ist X binomialverteilt mit den Parametern n und p, so gilt für den Erwartungswert E(X) und die Varianz V(X) der Zufallsvariablen

1. $E(X) = n \cdot p$
2. $V(X) = n \cdot p \cdot (1 - p)$

Abweichung einer Zufallsvariablen vom Erwartungswert

Eine wichtige Fragestellung ist: Wie groß ist die Wahrscheinlichkeit, daß eine Zufallsvariable um einen bestimmten Wert von ihrem Erwartungswert abweicht? Diese Wahrscheinlichkeit läßt sich mit der *Ungleichung von Tschebyscheff* oder (bei Bernoulli-Versuchen) mit den *Chernoff-Schranken* abschätzen. Durch die Chernoff-Schranken bekommt man i.a. eine exponentiell kleine Wahrscheinlichkeit. Wir verwenden die Chernoff-Schranken in Form von folgenden Gleichungen:

$$\mathbf{Pr}[X \leq (1-\delta)E(X)] \; < \; e^{\left(\frac{-E(X)\delta^2}{2}\right)} \tag{2.3}$$
$$\mathbf{Pr}[X > (1+\delta)E(X)] \; < \; e^{-0{,}386E(X)\delta^2} \tag{2.4}$$

Ausführlicheres zu den Chernoff-Schranken ist in [MR95, Kapitel 4] zu finden.

2.4 Grundlagen der Komplexitätstheorie

In diesem Abschnitt sollen einige der Grundlagen der Komplexitätstheorie kurz erwähnt werden. Weiterführende Literatur dazu ist [HU79] und [BDG88].

Die Grundfrage in der Komplexitätstheorie lautet: Welchen Aufwand erfordert die Lösung eines algorithmischen Problems? Beim Aufwand unterteilt man in

- Verbrauch von Speicherplatz und
- Verbrauch von Rechenzeit

Als Berechnungsmodell dienen häufig Turingmaschinen. Es ist klar, daß je größer das Problem wird, desto mehr Ressourcen werden zu seiner Lösung verbraucht. Die Größe eines Problems (z.B. Sortieren einer Zahlenliste) hängt von einem Eingangsparameter ab (z.B. der Länge der Zahlenliste). Interessant ist es, die dem Problem „inneliegende" Komplexität und weniger die Größe des Eingangsparameters zu bestimmen. Deshalb wird die Komplexität eines Problems in Abhängigkeit von dem Eingangsparameter n angegeben. Die Komplexität eines Problems ist dann eine Funktion f(n).

Gibt es einen Algorithmus, der ein spezielles Problem löst, z.B. *Bubblesort* sortiert eine Liste von natürlichen Zahlen, und kennt man den Rechenaufwand g(n) des Algorithmus (in Abhängigkeit von der Größe der Eingabe n), so hat man auch eine prinzipielle *obere Schranke* für die Komplexität des Problems. Denn dieses Problem läßt sich ja mit dem Algorithmus mit dem Aufwand g(n) lösen. Es kann aber durchaus sein, daß das Problem mit geringerem Aufwand gelöst werden kann, deshalb ist g(n) nur eine obere Schranke der Komplexität des Problems. Eine untere Schranke der Komplexität eines Problems, ist der Aufwand der mindestens nötig ist, um ein Problem zu lösen. Die Eingangsgröße n ist trivialerweise eine untere Schranke für die Zeitkomplexität eines Problems. Denn jeder

Algorithmus, der ein gegebenes Problem korrekt löst muß zumindet die Eingabe ganz lesen und dies erfordert n Schritte.

Der Nachweis (von nicht trivialen) unteren Schranken ist im allgemeinen schwer zu führen. Dies liegt daran, daß die Argumentation über alle möglichen Algorithmen zu führen ist, auch über bisher noch gar nicht erfundene. Für das Sortierproblem beispielsweise kann man beweisen, daß die untere Schranke der Komplexität n $\cdot$ log n ist. Da in diesem Fall auch Algorithmen bekannt sind, die denselben Aufwand benötigen, fallen hier obere und untere Schranke zusammen und die Komplexität des Sortierproblems ist genau bestimmt.

Um eine Ordnung in die große Vielfalt von möglichen Problemen und deren Komplexität zu bringen, faßt man Probleme mit ähnlicher Komplexität zu *Komplexitätsklassen* zusammen.

Definition 2.6
Sei $f : \mathbf{N} \longrightarrow \mathbf{N}$ eine Funktion.
*Eine Turing-Maschine M heißt $f(n)$-*zeitbeschränkt*, falls für alle $x \in \Sigma^*$*
$time_M(x) \leq f(|x|)$ gilt.

Hierbei bedeutet $time_M : \Sigma^* \longrightarrow \mathbf{N}$ die Anzahl der Rechenschritte von M bei Eingabe von x. Dann ist DTIME($f(n)$) *die Klasse aller Sprachen S, für die es eine deterministische Mehrbandturingmaschine gibt, die S erkennt und die $f(n)$-zeitbeschränkt ist.*

Entsprechend läßt sich NTIME($f(n)$) für nichtdeterministische Turingmaschinen definieren.

Definition 2.7
Die Komplexitätsklasse P beschreibt die Menge aller Sprachen, die von einer deterministischen Turingmaschine in polynomieller Zeit entschieden werden kann, also:

$$\mathbf{P} := \bigcup_{p \text{ ist Polynom}} DTIME(p(n))$$

Entsprechend ist NP:

$$\mathbf{NP} := \bigcup_{p \text{ ist Polynom}} NTIME(p(n))$$

Das P-NP Problem

Es ist klar, P ist in NP enthalten. Ob allerdings P echt in NP enthalten ist, also P$\neq$ NP gilt, oder nicht ist das bekannte P-NP Problem. Dieses Problem das ca. 1970 erstmals gestellt wurde ist weiterhin ein offenes Problem. Es kann als das wichtigste Problem der gesamten (theoretischen) Informatik aufgefaßt werden. Der Grund dafür ist, daß es viele für die Praxis wichtige Aufgabenstellungen gibt, von denen man leicht nachweisen kann, daß sie in der Klasse NP liegen. Z.B. das Stundenplan-Problem, das Rucksackproblem oder das Traveling Salesman Problem. Es sind für diese Probleme keine Algorithmen bekannt, deren worst-case Laufzeit sich durch ein Polynom beschränken läßt. Einige der Probleme in NP, von denen kein polynomieller Algorithmus bekannt ist hängen in der Weise zusammen, daß entweder alle diese Probleme polynomiale Algorithmen besitzen oder gar keins. Diese Probleme scheinen die schwierigsten Probleme der Klasse NP zu sein. Man nennt sie deshalb NP-vollständig. Da schon viele Forscher erfolglos seit langer Zeit versucht haben polynomiale Algorithmen für NP-vollständige Sprachen zu finden, glaubt man, daß es keine solchen Algorithmen gibt (für diese Probleme). Das ist natürlich kein Beweis. Aber die NP-vollständigen Probleme scheinen Kandidaten für die Menge NP-P zu sein. Die Vermutung ist deshalb P $\neq$ NP.

Definition 2.8 (polynomial-reduzierbar)
Seien A,$B \subseteq \Sigma^*$ Sprachen.
Dann heißt A auf B **polynomial reduzierbar** *- symbolisch mit $A \leq_p B$ bezeichnet - falls es eine totale und mit polynomialer Komplexität berechenbare Funktion f: $\Sigma^* \longrightarrow \Sigma^*$ gibt, so daß für alle $x \in \Sigma^*$ gilt:*

$$x \in A \Longleftrightarrow f(x) \in B$$

Definition 2.9 NP-vollständig
Eine Sprache A heißt NP-vollständig, falls folgendes gilt:

- *in NP liegt*
- *für alle Sprachen $S \in NP$ gilt $S \leq_p A$*

Ein Beispiel für ein NP-vollständiges Problem ist das Hamilton-Kreis Problem.

Gegeben ist ein ungerichteter Graph G(K_n, K_a).
Gefragt ist: Gibt es einen geschlossenen Weg, der jeden Knoten genau einmal besucht?

Effiziente Berechnungen

Wir wollen von der konservativen Vorstellung abrücken, daß die Klasse P die effizienten Berechnungen charakterisiert. Es setzt sich immer mehr durch, die effizienten Berechnungen durch die Berechnungen welche von probabilistischen polynomial zeitbeschränkten Turingmaschinen durchgeführt werden zu charakterisieren.

Solch eine probabilistische Turingmaschine M kann ihr Verhalten von den Zufallsbits abhängig machen. Häufig betrachtet man die Sequenz von Zufallsbits als zusätzliche Eingabe. Dann bezeichnet M(x, r) die Zufallsvariable, die mit der Ausgabe der Maschine M bei Eingabe von x korrespondiert. Für jedes x ist M(x, r) eine eigene Zufallsvariable. Es wird die Verteilung von M(x, r) für jedes feste x und gleichverteiltem r (einer festen Länge) betrachtet.

Die Komplexitätsklasse, die mit den effizient durch obige prob. Maschinen akzeptierten Sprachen korrespondiert, ist die Klasse BPP.

Definition 2.10 Die Klasse BPP
BPP ist die Klasse von Sprachen S, die von probabilistischen, polynomial zeitbeschränkten Maschinen akzeptiert werden. Eine Sprache S wird von einer prob. poly. zeitbeschränkten Maschine M akzeptiert, falls

- $\forall x \in S$ *gilt :* $Prob(M(x) = 1) \geq \frac{2}{3}$
- $\forall x \notin S$ *gilt :* $Prob(M(x) = 1) < \frac{1}{3}$

Wahrscheinlichkeitsverstärkung

Warum wurde gerade $\frac{2}{3}$ bzw. $\frac{1}{3}$ als Konstanten bei der BPP Definition gewählt? Tatsächlich ist diese Wahl rein willkürlich. Es reicht, wenn zwischen beiden Wahrscheinlichkeiten eine kleine Lücke besteht, dann läßt sich eine *Wahrscheinlichkeitsverstärkung* durchführen.

Dieses Prinzip der Wahrscheinlichkeitsverstärkung läßt sich häufig anwenden und deshalb wollen wir es hier exemplarisch vorführen. Substituiert man in obiger Definition $\frac{2}{3}$ durch $\frac{1}{2} + \epsilon$ und entsprechend $\frac{1}{3}$ durch $\frac{1}{2} - \epsilon$, so wird die gleiche Klasse definiert. Dies kann man mit der Tschebyscheff-Ungleichung zeigen, wir wollen allerdings noch etwas besseres zeigen, nämlich mit der Wahrscheinlichkeitsverstärkung bekommt man sogar die Wahrscheinlichkeiten $1 - 2^{-n}$ bzw 2^{-n} hin. Diese Wahrscheinlichkeitsverstärkung erreicht man folgendermaßen:

Gegeben ein probabilistischer Algorithmus M mit Akzeptanzwahrscheinlichkeit $\frac{1}{2} + \epsilon$ und Verwurfswahrscheinlichkeit von $\frac{1}{2} - \epsilon$. Wir lassen den Algorithmus M auf gleicher Eingabe x k-mal ablaufen und führen dann einen Mehrheitsentscheid durch.

```
INPUT x;
z := 0;
FOR i:= 1 TO k do
    Simuliere M auf x; das Ergebnis sei y;
    IF y = „akzeptiert“ THEN z := z + 1; END;
END
IF z > ½k THEN akzeptiere ELSE verwerfe END
```

Das k-fache Ausführen des Zufallsversuchs M(x) stellt eine Bernoulli-Kette der Länge k dar, denn es wird k-mal unabhängig ein Zufallsexperiment durchgeführt, das eine Erfolgswahrscheinlichkeit von p hat.

Ist x $\in S$, dann ist $p \geq \frac{1}{2} + \epsilon$, sonst ist $p \leq \frac{1}{2} - \epsilon$.

Die Wahrscheinlichkeit für i Treffer in einer Bernoulli-Kette der Länge k ist

$$\binom{k}{i} \cdot p^i (1-p)^{k-i}$$

Die Wahrscheinlichkeit, daß dieser Algorithmus, bei Eingabe von $x \in S$ ein falsches Ergebnis liefert (also verwirft) ist höchstens:

$$\sum_{i=0}^{\frac{1}{2}k} \binom{k}{i} \cdot p^i (1-p)^{k-i}$$

Also werden die Wahrscheinlichkeiten aller Ereignisse mit weniger als $k/2$ Treffer aufsummiert. Die Zufallsvariable X beschreibt die Anzahl der Treffer in der Bernoulli-Kette der Länge k.

Der Erwartungswert E(X) von obiger Bernoulli-Kette der Länge k ist dann $k \cdot (\frac{1}{2} + \epsilon)$. Wir schätzen nun die Wahrscheinlichkeit nach oben ab, daß die Zufallsvariable X um mehr als $k \cdot \epsilon$ vom Erwartungswert abweicht. Mit höchstens dieser Wahrscheinlichkeit liefert der k-mal wiederholte Algorithmus ein falsches Ergebnis.

Es gilt (Chernoff-Schranke):

$$\text{Prob}[E(x) - X \geq r] \;\leq\; e^{\frac{-r^2}{4kp(1-p))}} \;\underbrace{\leq}_{p(1-p) \leq \frac{1}{4}}\; e^{\frac{-r^2}{k}}$$

Unsere Werte eingesetzt ergibt:

$$\text{Prob}[k \cdot (\frac{1}{2} + \epsilon) - X \geq k\epsilon] \leq e^{-\epsilon/k} \leq 2^{-\Omega(k)}$$

Es genügt also, das Protokoll O(n) mal zu wiederholen, um eine vernachlässigbare Fehlerwahrscheinlichkeit zu bekommen. Das Prinzip der Wahrscheinlichkeitsverstärkung läßt sich auch bei Interaktiven Beweissystemen anwenden.

Definition 2.11 (vernachlässigbar)

Wir nennen eine Funktion $f : \mathbf{N} \longrightarrow \mathbf{N}$ vernachlässigbar, falls für jedes Polynom $p(\cdot)$ ein n_0 existiert, so daß für alle $n \geq n_0$ gilt:

$$f(n) < \frac{1}{p(n)}$$

Eine vernachlässigbare Funktion bleibt vernachlässigbar, falls sie mit einem beliebigen Polynom multipliziert wird. Daraus folgt, daß ein Ereigniss, welches bei einem Zufallsversuch nur mit vernachlässigbarer Wahrscheinlichkeit auftritt, auch bei polynomial-oft wiederholtem Zufallsversuch nur äußerst selten auftritt.

Kapitel 3

Interaktive Beweise

Im Jahr 1985 führten Goldwasser, Micali und Rackoff in ihrem Paper [GMR85] den Begriff des interaktiven Beweissystems ein und lösten damit eine stürmische Entwicklung im Bereich der Komplexitätstheorie und besonders im Bereich der Kryptographie aus. Denn diese Interaktiven Beweise bilden die Grundlage für die Zero-Knowledge Beweise, die wir im nächsten Kapitel einführen wollen. In diesem Kapitel werden wir zunächst die Begriffe Beweis und Beweissystem informal diskutieren, bevor sie dann für den Fall des Interaktiven Beweissystems formal definiert werden. Die Interaktiven Beweissysteme sind für sich betrachtet schon interessant und haben vielfältige und wichtige Auswirkungen auf die Komplexitätstheorie, aber wir wollen nur die speziellen Interaktiven Beweissysteme mit der Zero-Knowledge-Eigenschaft und deren Anwendungen in der Kryptographie betrachten.

3.1 Einleitung

Was ist eigentlich ein Beweis?
Ein Beweis ist ein Zeugnis, oder ein Grund um jemanden von einer Behauptung zu überzeugen[1]. Ein traditioneller mathematischer Beweis ist ein festes Objekt. Er besteht aus einer festen Sequenz von Beweisschritten und läßt sich niederschreiben.

Um einen solchen Beweis formulieren zu können, sind bestimmte Vorrausetzungen nötig. Zunächst müssen eine Reihe von Grundtatsachen festgelegt werden. Diese werden nicht bewiesen, aber man sieht sie trotzdem als richtig an. Diese Grundtatsachen nennt man *Axiome*. Ein Beispiel für eines der Peano'schen Axiome über die natürlichen Zahlen lautet „Jede natürliche Zahl hat einen Nachfolger“. Als nächstes muß eine Menge von *Schlußregeln* festgelegt werden. In einem Beweisschritt beruft man sich entweder auf ein Axiom oder man schließt nach den Schlußregeln von Axiomen und bereits bewiesenen Aussagen auf neue Aussagen.

Eine Aussage, die mit einem solchen Beweis bewiesen wurde, gilt als absolut richtig, sofern man von folgendem überzeugt ist:

[1] Eine Formulierung dieser Art findet man in verschiedenen Wörterbüchern unter dem Begriff „Beweis“.

- Die Axiome sind wahr.
- Die Schlußregeln sind korrekt, d.h. es kommen wieder wahre Aussagen heraus, wenn man die Schlußregeln auf wahre Aussagen anwendet.

Wie kann man überzeugt sein, daß die Axiome wahr sind? Man schaut sich z.B. die Axiome solange an, bis einem die Intuition sagt, „jawohl, das muß richtig sein“.[2]

Aus dem Blickwinkel der theoretischen Informatik ist zusätzlich zur Korrektheit eines Beweises wichtig, daß sich dieser effizient überprüfen läßt. Damit er sich effizient verifizieren läßt, muß der Beweis erst einmal ganz durchgelesen werden und darf deshalb auch nicht zu groß sein. Genauer heißt das, der Beweis darf nicht größer sein als ein Polynom in der Länge der Aussage, die bewiesen wird. Einen solchen Beweis nennen wir einen *effizienten* Beweis.

Die Effizienz bezieht sich hier nur auf den Verifikationsprozeß. Der Beweis heißt effizient, falls er schnell[3] von einer Verifikationsprozedur verifiziert werden kann. Dabei kann es allerdings recht aufwendig sein, diesen Beweis zu erzeugen. Diese Asymmetrie der Komplexität zwischen der Erzeugung und der Verifikation eines Beweises findet sich auch bei den Elementen der Komplexitätsklasse NP. Jeder Sprache $S \in NP$ läßt sich eine Relation R_S zuordnen, welche in Polynomialzeit auswertbar ist. Damit hat jede Sprache S in NP eine effiziente Verifikationsprozedur für Aussagen der Form $x \in S$.

Alle Sprachen S aus NP lassen sich somit auch wie folgt definieren:

$$S \;=\; \{x\,|\exists y,\text{ so daß } (x,y) \in R_S\}$$

Die Länge von y ist dabei polynomiell in der Länge von x. Gilt $(x, y) \in R_S$, so heißt y Beweis für $x \in S$[4]. Damit ist die Klasse NP die Klasse von Sprachen S, die effiziente Beweise für $x \in S$ haben. Nur korrekte Aussagen (also nur x die tatsächlich Element von S sind) haben einen Beweis y in diesem Beweissystem. Allgemein sind für effiziente Beweissysteme folgende Eigenschaften grundlegend:

- *Durchführbarkeit:*
  Die Verifikationsprozedur muß von der Richtigkeit einer Aussage überzeugt werden können, falls die Aussage tatsächlich wahr ist. Es muß also möglich sein, eine richtige Behauptung zu beweisen.
- *Korrektheit:*
  Die Verifikationsprozedur soll nicht von der Richtigkeit falscher Aussagen überzeugt werden können. Es soll nicht möglich sein, eine falsche Behauptung zu beweisen.

---

[2]Die Intuition kann man durch Experimente stützen, z.B. indem man viele Striche auf eine große Tafel malt und sich damit überzeugt, daß zumindest sehr viele natürliche Zahlen einen Nachfolger haben.

[3]d.h. in Polynomialzeit (in der Länge der Aussage)

[4]Dieses y kann man sich als Konfigurationsfolge einer akzeptierenden, nichtdeterministischen Turingmaschine, bei Eingabe x, vorstellen, um den Zusammenhang zur üblichen NP-Definition zu sehen

- Egal wie aufwendig es ist, einen Beweis zu erzeugen, dieser Beweis muß effizient[5] überprüfbar sein.

In einem Beweissystem gibt es eine Instanz, welche den Beweis erzeugt. Diese nennen wir im folgenden **Prover**. Die Verifikationsprozedur, welche den Beweis verifiziert, nennen wir **Verifier**.

Betrachtet man den Beweisbegriff außerhalb der mathematischen „Welt", z.B. bei Zeugenaussagen vor Gericht, so fällt auf, daß bei diesem Beweisbegriff die Interaktion von mehreren Parteien eine große Rolle spielt. Es ist demzufolge eine naheliegende Idee, diese Interaktion auch bei mathematischen Beweisen anzuwenden. Dies haben Goldwasser, Micali und Rackoff 1985 [GMR85] gemacht, als sie die interaktiven Beweise eingeführt haben. Mit diesen interaktiven Beweisen läßt sich vermutlich effizient mehr beweisen, als mit Beweisen ohne Interaktion. Ein Indiz dafür ist das Interaktive Beweissystem für Graph-Nichtisomorphie (siehe Kapitel 3.3.1).

3.2 Grundlegende Definitionen

Als erstes wollen wir ein geeignetes Berechnungsmodell einführen.

Definition 3.1 (Interaktive Turingmaschine)
Eine **interaktive Turingmaschine (ITM)** *ist eine probabilistische Mehrbandturingmaschine. Sie besitzt folgende Bänder:*

- *Ein Eingabeband mit Lesezugriff.*
- *Ein Zufallsband mit Lesezugriff.*
- *Ein Arbeitsband mit Lesezugriff und Schreibzugriff.*
- *Zwei Kommunikationsbänder, eins mit Lesezugriff, das andere mit Schreibzugriff.*
- *Ein Ausgabeband mit Schreibzugriff.*
- *Ein Schaltband mit einer einzelnen Zelle. Die Zelle enthält am Anfang immer eine 0.*

Jede ITM hat eine Identität. *Diese ist entweder 0 oder 1. Eine ITM ist* aktiv, *solange auf ihrem Schaltband ihre Identität steht. Im anderen Fall ist sie außer Betrieb. Dann kann sie ihren Zustand, sowie die Inhalte auf ihren Bändern nicht ändern.*

Den Inhalt, den die ITM auf das Kommunikationsband schreibt bzw. liest, während sie aktiv ist, heißt gesendete bzw. empfangene Nachricht. *Auf das Eingabeband bekommt sie ihre* Eingabe, *auf das Ausgabeband schreibt sie ihre* Ausgabe, *bevor sie anhält.*

[5] D.h. in Polynomialzeit in der Länge der Behauptung

Bemerkungen zu obiger Definition:
Wir nehmen an, daß der Inhalt des Zufallsbands einer ITM aus einer unendlichen Folge von unabhängig unter Gleichverteilung gezogenen Bits besteht.

Warum unterscheidet man bei einer ITM eigentlich das Eingabeband und das Kommunikationsband mit Lesezugriff bzw. das Ausgabeband und das Kommunikationsband mit Schreibzugriff? Wofür ist das Schaltband da? Denn ist die ITM erstmal außer Betrieb, kann sie sich selbst nicht mehr einschalten, da sie nach obiger Definition die Inhalte ihrer Bänder (insbesondere das Schaltband) nicht mehr ändern kann.

Die ITM ist eine interaktive Turingmaschine, wie dem Namen schon zu entnehmen ist und wird niemals allein für sich betrachtet, sondern nur in Verbindung mit einem Partner, mit dem sie kommunizieren kann. Der Partner ist in unseren Betrachtungen ebenfalls eine ITM. Die Maschinen werden so zusammengeschaltet, daß sie sich einige ihrer Bänder teilen.

Definition 3.2 (Zusammenschaltung von ITMs)
Man sagt, zwei ITMs A und B sind verbunden*, falls folgendes gilt:*

- *Die beiden Maschinen haben ein gemeinsames Eingabeband.*
- *A und B haben gegensätzliche Identitäten, d.h. nur eine Maschine ist zu einem Zeitpunkt aktiv. Die Schaltbändern der Maschinen sind verbunden. Die aktive Maschine ändert irgendwann einmal den Inhalt des Schaltbandes und setzt sich damit selbst außer Betrieb, während die andere Maschine dadurch aktiviert wird*[6]*.*
- *Das Kommunikationsband mit Lesezugriff von A ist mit dem Kommunikationsband mit Schreibzugriff von B verbunden und umgekehrt.*
- *Alle anderen Bänder haben keine Verbindung zueinander.*

Die gemeinsame Berechnung von zwei verbundenen ITMs, bei einer gemeinsamen Eingabe x*, ist eine Folge von Paaren* (a, b)*. Jedes Paar* (a, b) *enthält die beiden Konfigurationen der Maschinen. Die gemeinsame Berechnung der beiden ITMs ist beendet, falls eine ITM anhält, obwohl noch ihre Identität auf dem Schaltband steht.*

Ein **Interaktives Protokoll** ist ein geordnetes Paar (A, B) von verbundenen ITMs. Der Maschine A ist keine Beschränkung bzgl. der Rechenzeit auferlegt, während die Rechenzeit von Maschine B durch ein Polynom in der Länge der Eingabe beschränkt ist.

Die Berechnung in einem Interaktiven Protokoll läuft ab wie folgt. Eine der beiden Maschinen ist zu Beginn aktiv. O.B.d.A. sei B am Anfang aktiv. In einer aktiven Phase führt

[6] Es ist das gleiche Prinzip wie bei einer Schachuhr. Der Spieler der gerade am Zug ist, führt seine Berechnungen durch, während seine Uhr läuft. Hat er seinen Zug ausgeführt (entspricht in unserem Fall, daß die Maschine die Nachricht auf das Kommunikationsband geschrieben hat), drückt er auf seine Uhr. Die Uhr des Gegners wird somit aktiv und dieser führt nun seine Berechnungen durch.

die Maschine B einige interne Berechnungen aus und kann dabei ihr Eingabeband, ihr Arbeitsband und ihr Zufallsband benützen. Dann schreibt sie eine Nachricht (als Bitstring kodiert) auf ihr Kommunikationsband mit Schreibzugriff und schaltet als letzte Aktion den Inhalt der Zelle des Schaltbands auf ihre Gegenidentität.

Die andere Maschine ist nun aktiv und führt ebenso einige interne Berechnungen aus. Am Ende schaltet sie wieder auf Maschine B zurück. Damit ist eine **Runde** des Interaktiven Protokolls beendet. Eine Runde besteht somit aus zwei Wechseln.

Zur Notation:
Für zwei verbundene ITMs A und B repräsentiert die Zufallsvariable $\langle A, B\rangle(x)$ die Ausgabe der ITM B, falls diese bei gemeinsamer Eingabe x, mit ITM A kommuniziert hat. Dabei wird angenommen, daß die Zufallsbänder der beiden Maschinen wirklich fair gezogene Bits enthalten und daß die gemeinsame Berechnung von A und B bei jeder Eingabe x und allen Zufallsbitfolgen endlich ist.

Jetzt haben wir ein geeignetes Berechnungsmodell für ein interaktives Beweissystem definiert. Eine ITM übernimmt die Rolle des Provers, die andere die Rolle des Verifiers. Was wollen wir kurz zusammengefaßt von einem interaktiven Beweissystem fordern?

- Ein interaktives Beweissystem soll ein effizientes Beweissystem sein, d.h. der Verifier muß polynomial zeitbeschränkt[7] sein.
- Es muß einen Prover geben, der durch die Interaktion dem Verifier beweist, daß die gemeinsame Eingabe x eine bestimmte Eigenschaft besitzt. Dies bedeutet, daß der Prover beweist, daß die gemeinsame Eingabe x ein Element einer Sprache S ist. Dabei sind die Berechnungsressourcen des Provers nicht eingeschränkt.
- Der Verifier ist eine probabilistische Maschine. Die Korrektheits- und Durchführungsbedingungen eines Beweissystems werden aufgelockert. D. h. der Verifier muß nur noch mit hoher Wahrscheinlichkeit von richtigen Aussagen überzeugt werden können und darf mit geringer Wahrscheinlichkeit von falschen Aussagen überzeugt werden.
- Der Verifier muß am Ende der Interaktion ein Urteil ausgeben. Er soll eine 1 ausgeben, falls er akzeptiert, sonst gibt er eine 0 aus.

Definition 3.3 (interaktives Beweissystem)
Ein Paar von verbundenen ITMs (P, V) heißt **interaktives Beweissystem** *für eine Sprache S, falls die Maschine V eine polynomial zeitbeschränkte ITM ist und folgende zwei Bedingungen gelten:*

- *Durchführbarkeit:* $\forall x \in S$

$$Prob(\langle P, V\rangle(x) = 1) \;\geq\; \frac{2}{3}$$

[7] In der Länge der zu beweisenden Aussage, entspricht hier der Länge der gemeinsamen Eingabe

- *Korrektheit:* $\forall x \notin S$ *und* $\forall$ *ITMs* P^*

$$Prob(\langle P^*, V\rangle(x) = 1) \;\leq\; \frac{1}{3}$$

Die Wahl der Konstanten $\frac{2}{3}$ und $\frac{1}{3}$ in obiger Definition ist rein willkürlich. Die Fehlerwahrscheinlichkeit des Protokolls bekommt man auf exponentiell kleine Werte, indem man das obige Protokoll polynomiell oft ausführt und dann einen Mehrheitsentscheid macht. Dieses Prinzip heißt *Wahrscheinlichkeitsverstärkung* (siehe Grundlagenkapitel).

Wie leicht zu sehen ist, hat jede Sprache S aus der Klasse NP einen interaktiven Beweis. Das interaktive Beweissystem bekommt eine Eingabe $x \in S$ und der Prover berechnet einen Beweis y. Er schickt diesen Beweis an den Verifier und dieser wertet die Relation R_S in Polynomialzeit aus. Der Verifier akzeptiert hier immer jede Eingabe $x \in S$. Andererseits akzeptiert der Verifier nie eine Eingabe, die nicht in S ist. Damit können beide Maschinen deterministisch sein, da sie ihre Zufallsbits bei NP-Beweisen nicht benötigen.

Definition 3.4 (Die Klasse IP)
Alle Sprachen, die ein interaktives Beweissystem haben, werden zur Klasse **IP** *zusammengefaßt.*

Offensichtliche Inklusionsbeziehungen

Es folgt sofort NP $\subseteq$ IP aus den obigen Bemerkungen. Da die Klasse BPP aus Sprachen besteht, welche von einer probabilistischen polynomial zeitbeschränkten Turingmaschine akzeptiert werden, ist auch BPP in IP enthalten. Jede BPP Sprache hat also ein Interaktives Beweissystem ohne jede Interaktion.

Arthur-Merlin Beweissysteme

Ein Interaktives Beweissystem, bei dem der Verifier alle seine Nachrichten an den Prover zufällig wählt heißt *Arthur-Merlin Beweissystem* (kurz AM-Beweissystem). Da der Prover alle Zufallsbits sieht, die der Verifier benutzt, sagt man auch, daß der Verifier *öffentliche Zufallsbits* benutzt. Somit ist die einzige nicht-triviale Berechnung des Verifiers die Berechnung eines in Polynomialzeit auswertbaren Prädikats am Ende der Interaktion. Diese Auswertung entscheidet über Akzeptanz oder Ablehnung der Eingabe. In diesem Beweissystem, das von Babai 1985 eingeführt wurde, spielt der mit normalen „Geisteskräften“ ausgestatette König Arthus die Rolle des Verifiers, während der allmächtige Zauberer Merlin in die Rolle des Provers schlüpft. Da ein AM-Beweissystem ein Spezialfall eines Interaktiven Beweissystems ist, folgt sofort, daß jede Sprache S, die ein AM-Beweissystem hat, auch ein Interaktives Beweissystem besitzt.

Goldwasser und Sipser haben 1986 gezeigt, daß eine Sprache genau dann einen Interaktiven Beweis hat, wenn sie einen AM-Beweis hat. Es ist nützlich, beide Definitionen zur Verfügung zu haben, denn mit Interaktiven Beweissystemen ist es einfacher, bestimmte Protokolle zu konstruieren, während es mit AM-Beweissystemen oft leichter fällt, Komplexitätsresultate zu beweisen.

3.3 Isomorphie von Graphen

Zwei Graphen gleicher Größe, also mit gleich viel Knoten, werden *isomorph* genannt, wenn der eine Graph aus dem anderen Graphen durch eine Umordnung (Permutation) der Knoten hervorgeht.

Es ist eine leichte Aufgabe, von einem gegebenen Graph G einen dazu isomorphen Graphen H zu erzeugen. Dazu braucht man nur die Knotenmenge von G zu permutieren.

Umgekehrt ist es, nach heutigem Wissen, schwer (das heißt nicht in Polynomialzeit, gemessen in der Größe des Graphen), von zwei gegebenen Graphen G und H zu entscheiden, ob sie isomorph sind bzw. diesen Isomorphismus (also die Permutation) anzugeben. Allerdings kann ein angegebener Isomorphismus schnell verifiziert werden.

Wir wollen den Graph-Isomorphie-Begriff nun formal definieren...

Definition 3.5 (Graph-Isomorphie)
Zwei Graphen, $G_1(N, K_1)$ und $G_2(N, K_2)$ heißen **isomorph**, *bezeichnet durch $G_1 \cong G_2$, falls es eine Permutation π auf der Menge $N = \{1, \ldots, n\}$ gibt, so daß eine Kante (u, v) genau dann in der Menge K_1 enthalten ist, wenn $(\pi(u), \pi(v))$ in der Menge K_2 enthalten ist. Die Funktion π bezeichnet man als* **Isomorphismus** *zwischen den beiden Graphen.*

Die Sprache Graphisomorphie *ist definiert als*

$$GI := \{(G_1, G_2) \mid G_1 \cong G_2\}.$$

Das Komplement der Sprache GI ist definiert als

$$GNI := \{(G_1, G_2) \mid G_1 \ncong G_2\}.$$

Bei den zu diesen Sprachen gehörenden Entscheidungsproblemen sind jeweils zwei Graphen G_1 und G_2 mit n Knoten gegeben und die Frage lautet: Gibt es einen Isomorphismus zwischen diesen beiden Graphen.

3.3.1 Interaktives Beweissystem für GNI

Das in diesem Abschnitt betrachtete Beispiel eines Interaktiven Beweissystems für die Sprache GNI stammt von Goldreich, Micali und Wigderson [GMW91]. Ein wichtiger

Aspekt der interaktiven Beweismethode wird in diesem Beispiel deutlich. Es lassen sich in einem interaktiven Beweissystem Aussagen effizient beweisen, von denen kein effizienter traditioneller Beweis bekannt ist.

Die Sprache GNI ist in der Klasse co-NP enthalten. Es ist nicht bekannt, ob GNI in NP $\cup$ BPP enthalten ist. D.h. es ist auch kein effizientes traditionelles Beweisverfahren bekannt, das zeigt, daß zwei gegebene Graphen nicht isomorph sind.

Ein offensichtliches traditionelles Beweisverfahren, das zeigt, daß zwei Graphen G_0 und G_1 nicht isomorph sind, müßte alle Permutationen von G_0 aufschreiben und mit G_1 vergleichen.

Für einen Graphen mit n Knoten sind dies $n!$ Permutationen. Da $n!$ sich durch kein Polynom in n beschränken läßt, ist dieser Beweis nicht effizient.

Die Problemstellung ist:
Gegeben sind zwei Graphen G_0 und G_1 mit jeweils n Knoten. Beweise interaktiv, daß die beiden Graphen nicht isomorph sind. Das folgende Protokoll leistet das Erforderliche.

Interaktives Beweissystem für GNI

- Wiederhole die folgenden Schritte n mal:

  1. **V1:**
     Der Verifier rät zufällig ein $i \in \{0,1\}$ und eine Permutation π auf der Menge $\{1,\ldots,n\}$ unter Gleichverteilung.
     Er berechnet einen isomorphen Graphen $H = \pi(G_i)$ und schickt diesen an den Prover.
     $V \longrightarrow P \ : \ H$
  2. **P1:**
     Der Prover berechnet ein $j \in \{0,1\}$, so daß H isomorph zu G_j ist. Er schickt j an den Verifier.
     $P \longrightarrow V \ : \ j$
  3. **V2:**
     Überprüfe ob $i = j$. Falls nicht halte an und verwerfe.

- akzeptiere den Beweis falls $i = j$ in jedem der n Iterationen.

Satz 3.1
Das obige Protokoll ist ein Interaktives Beweissystem für GNI

Beweis :
Zu zeigen ist, daß das Protokoll durchführbar ist, d.h. Graphen aus GNI werden mit hoher Wahrscheinlichkeit akzeptiert, außerdem muß das Protokoll korrekt sein, was bedeutet, daß Graphenpaare aus GI nur mit sehr kleiner Wahrscheinlichkeit akzeptiert werden dürfen.

Durchführbarkeit:
Sind die beiden Graphen G_0 und G_1 nicht isomorph, dann kann der Prover (in jeder Iteration des Protokolls) immer herausfinden zu welchem Graphen der Graph H isomorph ist. Er kann also in jedem Fall das richtige j schicken. D.h. der Verifier akzeptiert mit Wahrscheinlichkeit 1. Damit ist das Protokoll durchführbar.

Korrektheit:
Nehmen wir an, die Graphen G_0 und G_1 sind isomorph, dann ist jeder Graph H, den der Verifier schickt, isomorph zu beiden Graphen G_0 und G_1. Der Prover bekommt also einen beliebigen zu beiden Graphen isomorphen Graphen und hat damit keine Möglichkeit, zu bestimmen, ob der Verifier diesen aus G_0 oder G_1 konstruiert hat. Auch seine unbeschränkte Rechenkraft hilft ihm hier nicht weiter und er kann nichts besseres tun, als eine Münze zu werfen, also zufällig raten. Der einzige Weg für den Prover, den Verifier zum Akzeptieren zu bringen, ist in jeder Iteration des Protokolls richtig zu raten. Die Wahrscheinlichkeit, daß er in einer Iteration richtig rät, ist $\frac{1}{2}$ und für alle n Iterationen ist die Wahrscheinlichkeit nur $(\frac{1}{2})^n$. Damit ist das Protokoll auch korrekt. □

Für das obige interaktive Beweissystem gilt, daß der Verifier die Eingaben, die in der Sprache liegen, immer mit Wahrscheinlichkeit 1 akzeptiert. Goldwasser, Mansour und Sipser [GMS87] haben gezeigt, daß man jedes Interaktive Beweissystem in ein anderes Interaktives Beweissystem (für die gleiche Sprache) umwandeln kann, welches alle Eingaben, die in der Sprache liegen mit Wahrscheinlichkeit 1 akzeptiert.

Mit obigem Satz folgt, daß die Sprache GNI $\in$ IP ist. Es ist aber nicht bekannt, ob GNI in NP liegt.

3.4 Erweiterung des Modells

In unserem bisher betrachteten Modell haben die beiden verbundenen Maschinen nur eine gemeinsame Eingabe. Private Eingaben für diese Maschinen konnten wir in unserem Modell nicht berücksichtigen. Es soll aber auch der Umstand berücksichtigt werden können, daß den Maschinen zusätzliche Informationen zur Verfügung stehen. Dazu eine kleine Diskussion, für welche Situationen eine zusätzliche private Eingabe von Bedeutung ist.

Ein Prover, dessen Berechnungsressourcen nicht eingeschränkt sind[8], läßt sich in der Praxis nicht verwirklichen. Auch der Prover muß effizient zu implementieren sein. Die zusätzliche private Eingabe könnte es gestatten, die Aufgaben des Provers effizient zu implementieren[9].

Wird der Prover genauso wie der Verifier beschränkt, so hat er im ursprünglichen Beweissystem keinen Nutzen mehr für den Verifier, denn der Prover kann ihm nichts mehr

[8] Es reicht den Prover auf PSPACE zu beschränken. Noch mehr Rechenleistung bringt ihm keinen Vorteil mehr.

[9] Diese private Eingabe könnte z.B. ein Isomorphismus zwischen zwei Graphen sein, beim Interaktiven Beweissystem für Graph-Isomorphie

mitteilen, was er sich nicht selbst berechnen kann. Hat der Prover allerdings Zugriff auf eine private Eingabe, so ersetzt diese unter Umständen die Einschränkung der Rechenzeit.

Interaktive Beweissysteme können auch als Unterprotokolle in einem größeren (kryptographischen) Protokoll eingesetzt werden. Dann könnten beide Parteien, Prover und Verifier, bereits zusätzliche Informationen von dem bisherigen Protokollverlauf bekommen haben, bevor sie an dem Interaktiven Beweissystem teilnehmen. Diesen Zugriff auf zusätzliche Information kann in unserem Modell durch eine private Eingabe modelliert werden.

Definition 3.6 (ITM mit privater Eingabe)
Die ITM wird durch ein weiteres Band mit Lesezugriff erweitert. Das Band heißt **Zusatz-Eingabeband**. *Die Eingabe auf diesem Band heißt* **Zusatzeingabe**. *Ist eine ITM polynomial zeitbeschränkt, so bedeutet dies, daß die Maschine weiterhin polynomial in der Länge der gemeinsamen Eingabe beschränkt ist.*

Zur Notation: Mit $\langle A(y), B(z)\rangle(x)$ bezeichnen wir die Zufallsvariable, welche die Ausgabe der ITM B repräsentiert, wenn B mit ITM A bei gemeinsamer Eingabe x kommuniziert. Dabei hat die ITM A die Zusatzeingabe y und die ITM B die Zusatzeingabe z.

Definition 3.7 (Erweitertes Interaktives Beweissystem)
Ein Paar (P, V) von ITMs heißt **Erweitertes Interaktives Beweissystem** *für eine Sprache S, falls die ITM V polynomial zeitbeschränkt ist und folgende zwei Bedingungen erfüllt sind.*

- *Durchführbarkeit:*
  Für alle $x \in S$ existiert ein String y, so daß für jedes $z \in \{0,1\}^*$

$$\mathit{Prob}(\langle P(y), V(z)\rangle(x) = 1) \;\geq\; \frac{2}{3}$$

- *Korrektheit:*
  Für alle $x \notin S$, jede ITM P^* und jedes $y, z \in \{0,1\}^*$ gilt

$$\mathit{Prob}(\langle P^*(y), V(z)\rangle(x) = 1) \;\leq\; \frac{1}{3}$$

Kapitel 4

Die Zero-Knowledge Eigenschaft

Goldwasser, Micali und Rackoff haben in ihrer bedeutenden Arbeit [GMR85] den Begriff *Zero-Knowledge* erstmals formal definiert. In diesem Paper stellten sie auch die ersten zwei Beispiele für Interaktive Beweissysteme mit der Zero-Knowledge Eigenschaft vor. Das waren Beweissysteme für die Sprachen QR_n[1] und QNR_n. Beide Sprachen liegen in NP $\cup$ co-NP. Die angegebenen Protokolle in [GMR85] und die Beweise, daß sie Perfekt Zero-Knowledge sind, konnten noch vereinfacht werden und sind in dieser Form in dem überarbeiteten Paper [GMR89] zu finden.

In der Einleitung zu diesem Kapitel werden wir informal den Zero-Knowledge Begriff erklären und an anschaulichen Beispielen erläutert. Um im dritten Abschnitt dieses Kapitels den Zero-Knowledge Begriff formal definieren zu können, benötigen wir das Konzept von nichtunterscheidbaren Wahrscheinlichkeitsverteilungen. Dieses werden wir im zweiten Abschnitt behandeln. In den weiteren Abschnitten dieses Kapitels stellen wir verschiedene Definitionen der Zero-Knowledge Eigenschaft vor und untersuchen die Beziehungen der verschiedenen Definitionen untereinander. Außerdem betrachten wir, ob diese formalen Definitionen die intuitiven Anforderungen, die wir an die Zero-Knowledge Eigenschaft stellen, erfüllen.

4.1 Einleitung

Intuitiv geht es bei einem Zero-Knowledge Beweis darum, jemanden davon zu überzeugen, daß man eine Information[2] besitzt, ohne das dabei irgendetwas über diese Information preisgegeben wird. Wurde eine Person durch einen Zero-Knowledge Beweis überzeugt, so hat diese Person nichts über die ursprüngliche Information erhalten und kann somit keinem Dritten beweisen, diese Information zu besitzen.

Es gibt tatsächlich ein schönes historisches Beispiel für einen interaktiven Beweis mit der Zero-Knowledge Eigenschaft. Dieses Beispiel ist aus [BSW95].

[1] Für die Definition von QR_n siehe Seite 13.

[2] Solch eine Information könnte z.B. ein formaler mathematischer Beweis einer Behauptung sein.

Schon seit dem Altertum kannte man eine Lösungsformel für quadratische Gleichungen. Allerdings war keine solche Formel für Gleichungen dritten Grades bekannt. Eine Gleichung dritten Grades hat folgende Normalform:

$$x^3 + ax^2 + bx + c = 0$$

So etwa um das Jahr 1535 fand Nicoló Tartaglia aus Oberitalien eine Lösungsformel für die Gleichungen dritten Grades. Tartaglia wollte seine Lösungsformel geheimhalten, behauptete aber, eine gefunden zu haben. Dadurch geriet er in einen Wettstreit mit dem italienischen Rechenmeister Antonio Maria Fior. Tartaglia stand vor dem Problem, Fior zu beweisen, daß er eine Lösungsformel für Gleichungen dritten Grades gefunden hatte, ohne Fior seine Lösungsformel zu zeigen. Nun legte Fior ihm 30 Gleichungen dritten Grades vor. Tartaglia löste alle 30 Gleichungen und konnte damit seine Behauptung beweisen.

Betrachten wir obigen interaktiven Beweis genauer.

- Zu jeder ihm gegebenen Gleichung kann er mit seiner Lösungsformel die Lösungen berechnen.
- Diese Lösungen werden einfach in die Gleichung eingesetzt und können somit schnell überprüft werden.
- Niemand kann nur aufgrund der Lösung die Lösungsformel herausbekommen.

Dieses Beweissystem ist also *durchführbar*. Es ist auch *korrekt*, denn jemand der die Lösungsformel nicht kennt, kann die Lösungen einer Gleichung nur erraten. Die Wahrscheinlichkeit eine richtige Lösung zu erraten ist aber für die meisten Gleichungen sehr klein, so daß nur genügend viele Gleichungen zum Test benutzt werden müssen. Tartaglia ist in diesem Beispiel der Prover, Fior der Verifier. Da Tartaglia seine Behauptung beweisen konnte ohne seine Information preiszugeben ist dies, intuitiv, auch ein Beispiel eines Zero-Knowledge Beweises.

In einem solchen Beweis gilt offenbar, daß der Prover dem Verifier etwas voraus haben muß. D.h. entweder hat er eine zusätzliche Information[3], oder er besitzt gegenüber dem Verifier größere Fähigkeiten. Beides können wir mit unseren, im dritten Kapitel definierten Interaktiven Beweissystemen modellieren. Im zuerst definierten Interaktiven Beweissystem 29 hat der Prover unbeschränkte Rechenleistung, während der Verifier polynomial zeitbeschränkt in der Länge der gemeinsamen Eingabe ist. Haben Verifier und Prover beide die gleiche Zeitbeschränkung, so braucht der Prover eine Zusatzinformation die ihn gegenüber dem Verifier auszeichnet. Dies läßt sich mit unserem Erweiterten Interaktiven Beweissystem modellieren (siehe Definition auf Seite 34).

Ein anschauliches Beispiel[4] für einen interaktiven Beweis mit der Zero-Knowledge Eigenschaft, bei dem der Prover, zusätzliche, Fähigkeiten besitzt um den Verifier zu überzeugen ist folgendes:

[3] Wie im obigen Beispiel die Lösungsformel

[4] In einer Vorlesung von Prof. Schöning gehört.

In einer Welt voller Farben gibt es bis auf einen nur farbenblinde Personen. Diese eine Person, die alle Farben unterscheiden kann, ist unser Prover. Der Prover kann nun jeder anderen farbenblinden Person interaktiv beweisen, daß es Farben gibt.

Dazu drückt er einem beliebigen Farbenblinden, der die Rolle des Verifiers übernimmt, zwei Socken mit unterschiedlichen Farben in die Hand, die sich außer in der Farbe in nichts weiterem unterscheiden. Für den Farbenblinden sehen beide Socken absolut identisch aus. Der Prover behauptet nun, die Socken haben unterschiedliche Farben beispielsweise rot und blau.

Der Verifier traut dem Prover nicht. Er nimmt die rote Socke in die linke Hand und die blaue Socke in die rechte Hand. Dann vertauscht er, beliebig oft, die Socken hinter seinem Rücken so, daß es der Prover nicht sehen kann. Er merkt sich dabei, wohin die Socke, die er ursprünglich in der linken Hand hatte, gewandert ist. Nach einer gewissen Zeit zeigt er dem Prover beide Hände mit den entsprechenden Socken und fragt ihn, in welcher Hand er die blaue Socke habe. Der Prover kann die Socken unterscheiden und sagt ihm die richtige Antwort. Der Verifier ist trotz der richtigen Antwort noch nicht so ganz überzeugt. Er denkt sich, der Prover könnte ja nur geraten haben und die Wahrscheinlichkeit, daß er zufällig richtig geraten hat, ist ja immerhin $\frac{1}{2}$. Also spielt er nochmals eine Runde. Er vertauscht wieder die Socken beliebig und fragt ein weiteres mal den Prover. Der Prover antwortet wieder richtig. Der Verifier ist nun schon etwas mehr überzeugt, denn die Wahrscheinlichkeit[5], daß der Prover zwei mal hintereinander richtig rät ist nur noch $\frac{1}{4}$. Der Verifier spielt noch einige Runden. Dabei wächst pro Runde seine Überzeugung, daß es tatsächlich Farben gibt. Irgendwann ist die Wahrscheinlichkeit, daß der Prover immer richtig geraten hat, so gering, daß er überzeugt ist.

Der Verifier ist davon in Zero-Knowledge Manier überzeugt worden, das heißt, er kann weiterhin keine Farben unterscheiden und kann deshalb auch nicht gegenüber einem Dritten die Existenz von Farben weiterbeweisen.

4.2 Nicht unterscheidbare Verteilungen

Um die Zero-Knowledge Eigenschaft im nächsten Abschnitt formal definieren zu können, brauchen wir den Begriff von nicht unterscheidbaren Wahrscheinlichkeitsverteilungen.

Klar ist, daß sich identische Objekte nicht unterscheiden lassen. Sind die Objekte zwar unterscheidbar, aber nicht durch eine effiziente[6] Prozedur, so bezeichnen wir die Objekte als *nicht effizient unterscheidbar*.

Besonders wichtig ist dabei, daß Objekte, die nicht effizient unterscheidbar sind, für alle praktischen Zwecke als identisch angesehen werden können.

Betrachten wir diesen Begriff im Kontext von Wahrscheinlichkeitsverteilungen. Grob gesagt heißen zwei Wahrscheinlichkeitsverteilungen nicht effizient unterscheidbar, falls es

[5] Die Zufallsversuche sind voneinander unabhängig

[6] Wir bezeichnen eine Berechnung als effizient, falls sie polynomial zeitbeschränkt ist. Ein effizienter Algorithmus darf allerdings Zufallsbits verwenden.

keinen effizienten Algorithmus D gibt, der diese unterscheiden kann. Da je zwei nur auf endlich vielen Werten definierten Wahrscheinlichkeitsverteilungen, welche nicht identisch sind, immer effizient unterscheidbar sind, definiert man diesen Begriff in Bezug auf zwei unendliche Folgen von Wahrscheinlichkeitsverteilungen. Eine solche Folge nennt man Wahrscheinlichkeits-Ensemble.

Definition 4.1 (Wahrscheinlichkeits-Ensemble)
Sei I eine abzählbar unendliche Indexmenge. Eine Folge von Zufallsvariablen $\Pi = \{\Pi(x)\}_{x\in I}$*, die mit Elementen von I indiziert ist heißt* **Wahrscheinlichkeits-Ensemble**, *oder kurz* **Ensemble**.

Ein Ensemble von Zufallsvariablen $\Pi = \{\Pi(x)\}_{x\in I}$ nennt man auch Familie von Zufallsvariablen. Die Indexmenge I ist in unseren Betrachtungen immer eine unendliche Teilmenge S von $\{0,1\}^*$

Definition 4.2 (nicht effizient unterscheidbar)
Sei S eine unendliche Teilmenge von $\{0,1\}^*$ *und seien* $\Pi_1 = \{\Pi_1(x)\}_{x\in S}$ *und* $\Pi_2 = \{\Pi_2(x)\}_{x\in S}$ *zwei Ensembles.*
Jede Zufallsvariable aus dem Ensemble nimmt Werte aus $\{0,1\}^*$ *an. Für jeden Algorithmus D bezeichnet* p_i^D *die Wahrscheinlichkeit, daß D eine "1" ausgibt bei Eingabe von x und einem Element* $\alpha \in \{0,1\}^*$ *entsprechend der Wahrscheinlichkeitsverteilung der ausgewählten Zufallsvariablen* $\Pi_i(x)$.

$$p_i^D(x) \;=\; \sum_{\alpha\in\{0,1\}^*} Prob(D(x,\alpha)=1)\cdot Prob(\Pi_i(x)=\alpha)$$

Zwei Ensembles $\Pi_1 = \{\Pi_1(x)\}_{x\in S}$ *und* $\Pi_2 = \{\Pi_2(x)\}_{x\in S}$ *heißen* **nicht effizient unterscheidbar**, *falls für alle probabilistischen polynomial zeitbeschränkten Algorithmen D, allen Polynomen* $q(\cdot)$ *und hinreichend großen x gilt:*

$$|p_1^D(x) - p_2^D(x)| \;\leq\; \frac{1}{q(|x|)}$$

Alle praktischen Anwendungen laufen in endlicher Zeit. Daraus folgt, daß keine solche Anwendung, außer für endlich viele kleine x, einen Unterschied zwischen zwei nicht effizient unterscheidbaren Ensembles feststellen kann. Deshalb können zwei solche Ensembles für alle praktischen Anwendungen als identisch angesehen werden.

Zwischen Gleichheit und nicht effizienter Unterscheidbarkeit von Ensembles gibt es noch eine Zwischenstufe.

Definition 4.3 (statistisch nicht unterscheidbar)
Zwei Ensembles heißen **statistisch nicht unterscheidbar**, *falls für alle Algorithmen D, alle Polynome* $q(\cdot)$ *und ausreichend große x gilt:*

$$|p_1^D(x) - p_2^D(x)| \leq \frac{1}{q(|x|)}$$

$p_1^D(x)$ und $p_2^D(x)$ sind dabei wie oben definiert.

Im Unterschied zu nicht effizient unterscheidbaren Ensembles ist die Laufzeit des Algorithmus D in dieser Definition nicht beschränkt. Die statistisch nicht unterscheidbaren Ensembles sind in der Praxis nicht so bedeutend wie die nicht effizient unterscheidbaren Ensembles.

4.3 Definitionen

In diesem Abschnitt werden wir die Zero-Knowledge Eigenschaft formal definieren. Die formale Definition soll folgende intuitive Vorstellung einfangen. Ein Interaktives Beweissystem (P, V) nennen wir Zero-Knowledge für eine Sprache S, falls sich alles, was bei Eingabe x nach der Interaktion mit dem Prover P effizient berechnen läßt, auch ohne jede Interaktion nur von der Eingabe x berechnet werden kann.

Das heißt, alles was der Verifier zu Gesicht bekommt, ist für jede effiziente Berechnung ohne Nutzen. Dabei gilt dies für jede effiziente Interaktion mit dem Prover P, auch für solche Verifier die sich nicht entsprechend dem Programm von V verhalten. Die Zero-Knowledge Eigenschaft ist somit eine Eigenschaft des Provers P. Sie beschreibt gewissermaßen die Widerstandsfähigkeit eines Provers gegenüber dem Versuch, durch effiziente Interaktion dem Prover Wissen zu entlocken.

4.3.1 Perfekt Zero-Knowledge

Definition 4.4 (Perfekt Zero-Knowledge)
Sei (P, V) ein Interaktives Beweissystem für eine Sprache S. Wir nennen (P, V) **Perfekt Zero-Knowledge***, falls für jede polynomial zeitbeschränkte ITM V^* ein im Erwartungswert polynomial zeitbeschränkter, probabilistischer Algorithmus M_{V^*} existiert, so daß für jedes $x \in S$ die Zufallsvariablen*

- *$\langle P, V^* \rangle(x)$ (repräsentiert die Ausgabe der ITM V^* nach der Interaktion mit Maschine P bei gemeinsamer Eingabe x).*

- *$M_{V^*}(x)$ (repräsentiert die Ausgabe des probabilistischen Algorithmus M_{V^*} bei Eingabe x).*

die gleiche Verteilung haben.

Der Algorithmus M_{V^*} kann als probabilistische, polynomial zeitbeschränkte Maschine betrachtet werden. Die Maschine M_{V^*} heißt **Simulator**. Damit ist die Ausgabe $M_{V^*}(x)$ für jede Eingabe x eine andere Zufallsvariable. Jede dieser Zufallsvariablen hat die gleiche Verteilung wie die entsprechende Zufallsvariable aus dem Ensemble $\{\langle P, V^* \rangle(x)\}$. Der Simulator muß nur für die Eingaben, die in der Sprache sind, die Ausgabe des Verifiers simulieren.

Der Simulator muß für jeden Verifier V^* und nicht nur für den ehrlichen Verifier V in der Lage sein, die Interaktion simulieren zu können. Der Simulator soll die Interaktion zwischen Prover und Verifier ohne jede Interaktion mit dem Prover simulieren. Kann dies der Simulator, dann ist intuitiv klar, daß der Verifier kein Wissen vom Prover bekommen haben kann, denn er hätte sich, wie der Simulator, alles selbst berechnen können.

Damit ist auch klar, daß jede Sprache in der Klasse BPP einen Zero-Knowledge Beweis hat. Der Verifier ist in diesem Fall mit dem Simulator identisch, er braucht keine Interaktion mit dem Prover.

4.3.2 Computational Zero-Knowledge

Für praktische Zwecke ist es nicht nötig, daß die Ausgabe des Verifier V^* perfekt simuliert wird. Es reicht aus, wenn der Simulator eine Ausgabe erzeugen kann die nicht effizient von der Ausgabe der Verifiers, der mit dem Zero-Knowledge Prover kommuniziert hat, unterschieden werden kann. Diese Abschwächung ist immer noch konsistent zu unser intuitiven Forderung, daß alles was sich effizient, bei Eingabe x, nach der Interaktion mit dem Prover berechnen läßt auch genauso ohne die Interaktion mit dem Prover nur aus der Eingabe x effizient berechnet werden kann

Definition 4.5 (Computational Zero-Knowledge)
Sei (P, V) ein Interaktives Beweissystem für eine Sprache S. Wir nennen (P, V) **Computational Zero-Knowledge** *(oder Zero-Knowledge), falls für jede polynomial zeitbeschränkte ITM V^* ein im Erwartungswert polynomial zeitbeschränkter, probabilistischer Algorithmus M_{V^*} existiert, so daß die Ensembles von Zufallsvariablen*

- $\{\langle P, V^* \rangle(x)\}_{x \in S}$ *(repräsentiert die Ausgabe der ITM V^* nach der Interaktion mit Maschine P bei gemeinsamer Eingabe x).*
- $\{M_{V^*}(x)\}_{x \in S}$ *(repräsentiert die Ausgabe des prob. Algorithmus M_{V^*} bei Eingabe x).*

nicht effizient unterscheidbar sind.

Zwischen den beiden Definitionen von Perfekt Zero-Knowledge und Computational Zero-Knowledge gibt es noch eine Unterscheidung, die in der Literatur „Statistical“ Zero-

Knowledge oder manchmal auch „Beinahe Perkekt“ Zero-Knowledge genannt wird. In dieser Definition wird gefordert, daß die Ensembles:

- $\{\langle P, V^* \rangle(x)\}_{x \in S}$ (repräsentiert die Ausgabe der ITM V^* nach der Interaktion mit Maschine P bei gemeinsamer Eingabe x).
- $\{M_{V^*}(x)\}_{x \in S}$ (repräsentiert die Ausgabe des prob. Algorithmus M_{V^*} bei Eingabe x).

statistisch nicht unterscheidbar sind.

P, V und M sind dabei wie in den beiden anderen Definitionen oben definiert.

Allerdings hat die Definition von „Statistical“ Zero-Knowledge nicht die Bedeutung wie die beiden anderen Definitionen und wird deshalb in dieser Arbeit nicht explizit definiert. Jedes Protokoll, das die Perfekt Zero-Knowledge Eigenschaft hat, ist natürlich auch Statistical- und insbesondere Computational Zero-Knowledge. Wir kürzen im folgenden häufig Computational Zero-Knowledge durch CZK, Statistical Zero-Knowledge durch SZK und Perfekt Zero-Knowledge durch PZK ab. Die entsprechenden Klassen, die die PZK,SZK und CZK Sprachen enthalten, bezeichnen wir mit $\mathcal{PZK}, \mathcal{SZK}$ und $\mathcal{CZK}$. Die bis jetzt bekannten Beziehungen sind:

$$\mathcal{BPP} \subseteq \mathcal{PZK} \subseteq \mathcal{SZK} \subseteq \mathcal{CZK} \subseteq \mathcal{IP}$$

Es wird vermutet, daß die ersten drei Inklusionen echt sind. Wie in [BOGG$^+$88] gezeigt wurde, ist unter der Annahme, daß es nichtuniforme Einwegfunktionen gibt $\mathcal{CZK} = \mathcal{IP}$.

Die Zero-Knowledge Eigenschaft setzt auf das Konzept eines Interaktiven Beweissystems auf. Durch den Interaktiven Beweis wird eine gewisse Eigenschaft der gemeinsamen Eingabe x des Beweissystems bewiesen. Man kann alle x, die eine solche Eigenschaft haben, zu einer Menge zusammenfassen. Eine solche Menge, die eine Teilmenge aller möglichen Eingaben ist, heißt Sprache. Es wird also bewiesen, daß die gemeinsame Eingabe x Element einer Sprache S ist. Hat der Interaktive Beweis zusätzlich die Zero-Knowledge Eigenschaft, so wird von dem Wissen des Provers nur das eine Bit, das aussagt, ob $x \in S$ ist oder nicht, verraten. Zero-Knowledge Beweise der obigen Art nennt man deshalb auch Zero-Knowledge of membership. Aus der Sicht des Verifiers ist die Interaktion mit einem Zero-Knowledge Prover äquivalent zu der Anwort eines vertrauenswürdigen Orakels.

4.3.3 Eine alternative Formulierung der Zero-Knowledge Eigenschaft

Betrachten wir die Interaktion in einem Interaktiven Beweissystem (P, V). Was sieht der Verifier von der Interaktion? Eine Mitschrift der Interaktion aus der Sicht des Verifiers besteht aus allen Inhalten der Bänder, auf die er Lesezugriff hat. Dazu gehören:

- Das Zufallsband.
- Das Kommunikationsband.
- Das Eingabeband.

Mit der Zufallsvariablen $\text{view}^P_{V^*}(x)$ beschreiben wir die Inhalte des Zufallsbands und des Kommunikationsbands des Verifiers bei Eingabe x. Dabei reicht es, nur die Nachrichten des Provers auf dem Kommunikationsband zu betrachten. Die Zufallsvariable $\text{view}^P_{V^*}(x)$ nimmt also Werte der Form $(r, \overline{m})$ an. Mit r werden die Zufallsbits des Verifiers bezeichnet und $\overline{m}$ bezeichnet die Folge von Nachrichten, die der Prover während der Interaktion mit V geschickt hat.

Wir geben nun eine alternative Definition von Zero-Knowledge an, bei der der Simulator nicht die Ausgabe des Verifiers simuliert sondern die Sicht des Verifiers von der Interaktion mit dem Prover.

Definition 4.6 (Alternative Definition von CZK)
Sei (P, V) ein Interaktives Beweissystem für eine Sprache S. Wir nennen (P, V) **Computational Zero-Knowledge** *(oder Zero-Knowledge), falls für jede polynomial zeitbeschränkte ITM V^* ein im Erwartungswert polynomial zeitbeschränkter, probabilistischer Algorithmus M_{V^*} existiert, so daß für jedes $x \in S$ die Ensembles von Zufallsvariablen*

- $\{view^P_{V^*}(x)\}_{x \in S}$ *und*
- $\{M_{V^*}(x)\}_{x \in S}$

nicht effizient unterscheidbar sind.

Entsprechend läßt sich dies für PZK definieren. Es ist klar, daß die Ausgabe eines Verifers V^* durch die Mitschrift der Interaktion bestimmt ist, und somit ist es mindestens so schwer die Mitschrift der Interaktion zu simulieren wie die Ausgabe des Verifiers. Umgekehrt muß dies für den ehrlichen Verifier nicht gelten. Da es aber für jeden Verifier V^* einen Verifier V' gibt, dessen Ausgabe die Mitschrift des Verifiers V^*, bei Interaktion mit dem Prover P, ist folgt, daß wenn über alle Verifier quantifiziert wird, beide Definitionen äquivalent sind.

Die Motivation zur Einführung der alternativen Definition ist, daß es sich mit dieser Definition oft besser arbeiten läßt. Die erste Definition erscheint dabei die „natürlichere“ zu sein (in bezug auf unsere intuitive Zero-Knowledge Vorstellung). Im folgenden benützen wir deshalb immer die alternative Formulierung.

4.4 Zusammensetzung von Zero-Knowledge Protokollen

Zero Knowledge Beweise werden häufig als Unterprotokolle in größeren kryptographischen Protokollen angewendet. Dies ist eines der Hauptanwendungsgebiete von Zero-Knowledge Beweisen. Eine grundlegende Frage für die Anwendbarkeit von Zero-Knowledge Beweisen ist, ob die Zusammensetzung von vielen Zero-Knowledge Protokollen zu einem größeren Gesamtprotokoll wieder ein Protokoll ergibt, welches die Zero-Knowledge Eigenschaft hat. Man unterscheidet zwei Arten von Zusammensetzungen. Als erstes wollen wir die hintereinander Ausführung von Protokollen definieren.

Definition 4.7 (sequentielle Komposition)
Seien $(P_1, V_1), \ldots, (P_k, V_k)$ Interaktive Beweissysteme für die entsprechenden Sprachen $S_1, S_2, \ldots, S_k$. Eine **sequentielle Komposition** *der k Protokolle – mit (P, V) bezeichnet – ist wie folgt definiert: Die gemeinsame Eingabe x für (P, V) ist ein String der Form $x_1\%x_2\% \cdots \%x_k\%$. Das Symbol „%“ ist ein Trennzeichen. Die Ausführung des Protokolls (P, V) besteht darin, daß alle Protokolle $(P_1, V_1), \ldots, (P_k, V_k)$ hintereinander auf ihrer entsprechenden Eingabe, (d.h. (P_i, V_i) mit Eingabe x_i) als Unterprogramme ausgeführt werden. V akzeptiert nur, wenn alle V_i akzeptiert haben.*

Entsprechend wird die gleichzeitige Ausführung von Protokollen definiert.

Definition 4.8 (parallele Komposition)
Seien $(P_1, V_1), \ldots, (P_k, V_k)$ Interaktive Beweissysteme für die entsprechenden Sprachen $S_1, S_2, \ldots, S_k$. Wir nehmen O.B.d.A an, daß alle k Protokolle aus m Wechseln bestehen. Zwei Wechsel entsprechen einer Runde. Eine **parallele Komposition** *der k Protokolle – mit (P, V) bezeichnet – ist wie folgt definiert: Die gemeinsame Eingabe x für (P, V) ist ein String der Form $x_1\%x_2\% \cdots \%x_k\%$. Die i-te Nachricht in dem Protokoll (P, V) besteht aus der Konkatenation der i-ten Nachrichten aller k Unterprotokolle. V akzeptiert nur, wenn alle V_i akzeptiert haben.*

Der Fall, daß nur ein einziges Protokoll (P_1, V_1), bei möglicherweise gleicher Eingabe x, k-mal wiederholt ausgeführt wird, ist ein Spezialfall der obigen Definition, wobei $\forall i : (P_i, V_i) = (P_1, V_1)$ und $x_i = x$.

Wir wollen im folgenden Nachweisen, daß die Orginal Computational Zero-Knowledge Definition von Goldwasser, Micali und Rackoff [GMR85] nicht unter seqentieller Komposition abgeschlossen ist. Es kann somit nicht argumentiert werden, daß ein Protokoll Zero-Knowledge (in bezug auf die Orginaldefinition) ist, weil es sich nur aus kleineren Zero-Knowledge Protokollen die hintereinander ausgeführt werden zusammensetzt. Dies läßt sich sehr elegant mit Hilfe von pseudozufälligen Mengen mit ausweichender Eigenschaft beweisen. Diese Mengen werden wir deshalb im nächsten Abschnitt einführen.

4.4.1 Pseudozufällige Mengen mit ausweichender Eigenschaft

In [GK92] untersuchen Goldreich und Krawczyk pseudozufällige Mengen, welche interessante ausweichende Eigenschaften haben. Intuitiv ist die Bezeichnung dadurch motiviert, daß es für alle effizienten Algorithmen schwer ist, Elemente von einer solchen Menge zu berechnen. Die Menge weicht sozusagen jeder effizienten Berechnung aus. Wir wollen nur kurz die benötigten Mengen mit ihren Eigenschaften vorstellen. Eine ausführliche Darstellung, sowie den Beweis, daß solche Mengen existieren, findet man in [GK92].

Eine Verteilung heißt pseudozufällig, falls sie nicht effizient von der Gleichverteilung unterschieden werden kann.

Definition 4.9
Eine Menge $T \subset \{0,1\}^k$ heißt $(\tau(k), \epsilon(k))$-pseudozufällig, falls für jeden probabilistischen Schaltkreis C der Größe $\tau(k)$ mit k Eingaben und einer einzelnen Ausgabe gilt:

$$|p_C(T) \; - \; p_C(\{0,1\}^k)| \leq \epsilon(k).$$

Mit $p_C(T)$ bezeichnen wir die Wahrscheinlichkeit, daß der Schaltkreis 1 ausgibt, wenn er zufällig gleichverteilt gewählte Elemente von T als Eingabe bekommt.

Eine Folge $T_1, T_2, ...$ von $(\tau(k), \epsilon(k))$-pseudozufälligen Mengen bezeichnet man als pseudozufälliges Ensemble, falls die Funktionen $\tau(k)$ und ϵ^{-1} schneller als jedes Polynom wachsen.

Definition 4.10
Sei $T_1, T_2, ...$ eine Folge von (nichtleeren) Mengen, so daß für jedes n gilt: $T_n \subset \{0,1\}^{q(n)}$, für ein festes Polynom q. Eine solche Folge nennt man polynomiell ausweichendes Ensemble – wir schreiben P-ausweichend –, falls für jeden probabilistischen polynomial zeitbeschränkten Algorithmus A, jede Konstante c, jedes ausreichend große n, und jedes $x \in \{0,1\}^n$ gilt:

$$Prob(A(x) \in T_n) < n^{-c}.$$

Die Wahrscheinlichkeit ergibt sich durch die Zufallsbits des Algorithmus A. Jeder polynomial zeitbeschränkte Algorithmus findet also nur mit vernachlässigbarer Wahrscheinlichkeit ein Element einer ausweichenden Menge.

Satz 4.1 (ohne Beweis)
Es gibt ein P-ausweichendes pseudozufälliges Ensemble $T_1, T_2, ...$ mit dem Polynom $q(n) = 4n$. Zusätzlich gibt es auch eine Turingmaschine, welche bei Eingabe von 1^n die Menge T_n ausgibt.

Ein nichtuniformer Algorithmus könnte als Zusatzeingabe die Strings einer ausweichenden Menge einkodiert haben (entspricht einer polynomiellen Zusatzeingabe). Wir definieren deshalb eine stärkere Ausweicheigenschaft, welche diese Fähigkeit auch gegen nichtuniforme Algorithmen bewahrt. Dazu muß für jedes n eine Zusammenstellung von exponentiell vielen Mengen angegeben werden. Soviel läßt sich in den nichtuniformen Algorithmus nämlich nicht reinkodieren, so daß diesem seine polynomial lange Zusatzeingabe nicht mehr soviel nützt.

Definition 4.11 (P/poly-ausweichend)
Sei $q(\cdot)$ ein Polynom. Mit $T^{(n)}$ bezeichnen wir eine Zusammenstellung von 2^n Mengen $\{T_1^{(n)}, \ldots, T_{2^n}^{(n)}\}$. Jedes $T_i^{(n)}$ ist eine Teilmenge von $\{0,1\}^{q(n)}$. Eine Folge $T^{(1)}, T^{(2)}, \ldots$ heißt nichtuniformes polynomiell ausweichendes Ensemble – wir schreiben P/poly-ausweichend – Ensemble, falls für jede Konstante $c > 0$, ausreichend großes n, und jeden probabilistischen Schaltkreis C der Größe n^c (mit n Eingaben und q(n) Ausgaben) gilt:

$$Prob(C(i) \in T_i) \ < \ \frac{1}{n^c}.$$

Die Wahrscheinlichkeit ergibt sich über der zufälligen Auswahl der $i \in \{1, \ldots, 2^n\}$, sowie von den Zufallsbits des Schaltkreises C.

Das heißt also, daß jeder Schaltkreis C der Größe polynomiell in n, der einen zufällig ausgewählten Index i von einer der Mengen in $T^{(n)}$ bekommt, nur mit vernachlässigbarer Wahrscheinlichkeit ein Element dieser Menge berechnen kann.

Satz 4.2 (ohne Beweis)
Es gibt ein P/poly-ausweichendes Ensemble $T^{(1)}, T^{(2)}, \ldots$ mit Polynom $Q(n) = 4n$, so daß für alle n, jedes $T_i^{(n)}$ eine $(2^{n/4}, 2^{-n/4})$-pseudozufällige Menge der Mächtigkeit 2^n ist. Zusätzlich gibt es eine Turingmaschine, welche bei Eingabe von 1^n die Zusammenstellung von Mengen $T^{(n)}$ ausgibt.

Einen Beweis dieses Satzes befindet sich im Anhang von [GK96].

4.4.2 Sequentielle Komposition von CZK

Eine natürliche Forderung an ein Zero-Knowledge Protokoll, die sich auch mit unserer intuitiven Vorstellung von Zero-Knowledge deckt, ist, daß ein Zero-Knowledge Protokoll auch bei mehreren Wiederholungen ein Zero-Knowledge Protokoll bleibt. Wir untersuchen zunächst, ob die Original CZK-Definition diese Forderung erfüllt.

Satz 4.3
Computational-Zero-Knowledge ist nicht unter sequentieller Komposition abgeschlossen.

Beweisidee:

Die Definition von Computational-Zero-Knowledge fordert, daß die Information, die im Verlauf eines solchen Protokolls übertragen wird, für jede polynomial zeitbeschränkte Berechnung ohne Nutzen ist. Solch eine Information kann sich der Verifier ohne fremde Hilfe selbst berechnen. Dies schließt aber nicht aus, daß diese nutzlose Information es einem betrügerischem Verifier bei einer weiteren Ausführung des Protokolls ermöglicht, aus dem nicht polynomial zeitbeschränkten Prover nützliche Information[7] herauszubekommen. Die Schwachstelle der Computational-Zero-Knowledge Definition ist, daß diese Definition nicht berücksichtigt, daß der Verifier Zugriff auf weitere Informationen, die z.B. von einer früheren Protokoll-Iteration herrühren kann, hat. Dies entspricht dann einem nichtuniformen Verifier. Die original Computational-Zero-Knowledge Definition berücksichtigt nur uniforme Verifier.

Dies läßt sich ausnutzen, um ein Gegenbeispiel zu konstruieren. Das Gegenbeispiel soll bei einer einfachen Ausführung CZK sein, aber bei mehrfacher Ausführung kann ein Verifier Informationen aus dem Prover extrahieren, die er sich selbst nicht berechnen kann.

Beweis :
Wir nehmen ein möglichst einfaches Interaktives Beweissystem her und modifizieren es passend. Ein sehr einfaches Protokoll ist das Protokoll für die Sprache $S = \{0,1\}^*$ In diesem Protokoll akzeptiert der Verifier einfach jede Eingabe. Dazu ist sogar keine Interaktion mit dem Prover notwendig, und aus diesem Grund ist das Protokoll trivialerweise CZK. Wir modifizieren nun dieses Protokoll. Es wird um eine Interaktion erweitert, durch die Information vom Prover zum Verifier übertragen werden kann, aber so, daß trotzdem noch die gleiche Sprache $S = \{0,1\}^*$ akzeptiert wird.

Für die Protokollerweiterung benötigen wir noch folgendes:

- Ein P-ausweichendes pseudozufälliges Ensemble $\{T_i\}_{i\in\mathbf{N}}$
- Eine harte Boolsche Funktion K, so daß die Sprache $S_K := \{x \mid K(x) = 1\}$ nicht in der Klasse BPP enthalten ist.

Damit konstruieren wir folgendes Protokoll (P, V) für $S = \{0,1\}^*$.

[7] Das ist eine Information, die sich der Verifier nicht selbst berechnen kann.

- Die gemeinsame Eingabe sei x der Länge n.

  1. **V1:**
     $s \in_R \{0,1\}^{4n}$
     $V \longrightarrow P \;:\; s$

  2. **P1:**
     Falls $s \in T_n$ dann $P \longrightarrow V \;:\; K(x)$
     Sonst $s_0 \in_R T_n$ und $P \longrightarrow V \;:\; s_0$

  3. **V2:**
     Akzeptiere immer.

Der Verifier startet das Protokoll mit seiner ersten aktiven Phase (V1). In dieser Phase wählt er zufällig unter Gleichverteilung einen String s aus der Menge aller Bitstrings der Länge $4n$ aus. Der Verifier schickt diesen String s an den Prover.

Der Prover überprüft, ob der String s, den der Verifier ihm geschickt hat in der Menge T_n enthalten ist. Ist der String s in der Menge T_n enthalten, so schickt er dem Verifier den Wert $K(x)$, im anderen Fall schickt er dem Verifier einen String s_0 zu, den er zufällig und unter Gleichverteilung aus der Menge T_n wählt. Der Prover kann dies, weil er nicht polynomial zeitbeschränkt ist.

Der Verifier akzeptiert die Eingabe x, in seiner zweiten Phase (V2), in beiden Fällen.

In jeder Ausführung des Protokolls wird dasselbe P-ausweichende pseudozufällige Ensemble benützt. Dadurch hängt die Menge T_n anstatt von der speziellen Eingabe x des Protokolls nur von der Länge n von x ab.

Behauptung: Das Protokoll ist bei einmaliger Ausführung CZK.

Beweis :
Wir geben für jeden Verifier V^* einen Simulator M_{V^*} an, der

- in der Laufzeit polynomiell beschränkt ist und
- dessen Ausgabe nicht effizient von der view des Verifiers V^* bei Interaktion mit dem Prover unterschieden werden kann.

Der Simulator kann immer die Schritte des Verifiers simulieren. Es reicht deshalb, die Simulation des Provers genauer zu betrachten. Der Prover sendet während des Protokolls nur eine Nachricht an den Verifier.

Fall1:
Der Prover sendet den Wert von K(x).

> Wie kann der Simulator dies simulieren? Er müßte dazu $K(x)$ berechnen können. Dieser Fall tritt allerdings nur auf, falls der Strings s, den der Verifier sendet, Element der Menge T_n ist. Hier wird jetzt die ausweichende Eigenschaft

> der Menge T_n ausgenützt. Wegen dieser Eigenschaft der Menge T_n kann eine polynomial zeitbeschränkte Maschine[8] unabhängig von der speziellen Eingabe x, ein Element der Menge T_n nur mit vernachlässigbarer Wahrscheinlichkeit finden. D. h. dieser Fall tritt nur mit vernachlässigbarer Wahrscheinlichkeit auf und braucht deswegen vom Simulator gar nicht beachtet werden.

Fall2:
Der Prover schickt ein zufällig unter Gleichverteilung aus der Menge T_n ausgewähltes Element s_0. Der Simulator kann kein Element aus T_n zufällig auswählen, außer mit vernachlässigbarer Wahrscheinlichkeit. Er simuliert dies, indem er zufällig unter Gleichverteilung ein Element aus $\{0,1\}^{4n}$ auswählt. Diese Auswahl ist, aufgrund der Pseudozufälligkeit der Menge T_n effizient nicht von der Auswahl des Provers unterscheidbar.

In beiden Fällen kann der Simulator die view des Verifiers V^*, bei Interaktion mit P so simulieren, daß sie nicht effizient von der tatsächlichen view des Verifiers V^* unterschieden werden kann. Der Simulator braucht dafür nur polynomielle Laufzeit.

$\square$

Das Protokoll wird nun zweimal hintereinander ausgeführt.

Behauptung: Das Protokoll ist nicht CZK, falls es zweimal hintereinander ausgeführt wird.

Beweis :
Folgt der Verifier dem Protokoll, so bekommt er auch weiterhin keine Information, die er sich nicht selbst berechnet haben könnte.

Betrachten wir nun im folgenden einen beliebigen Verifier V^*.

Beobachtung:

> Egal wie dieser V^* in der ersten Iteration des Protokolls vom ehrlichen Verifier V abweicht, er bekommt trotzdem mit überwältigender Wahrscheinlichkeit einen String $s_0 \in T_n$ vom Prover.[9]

Wie kann der Verifier V^* nun in der zweiten Iteration des Protokolls betrügen, um eine wichtige Information vom Prover zu bekommen?

Er kann statt einen zufällig aus $\{0,1\}^{4n}$ gewählten String s den String $s_0 \in T_n$ aus der ersten Iteration des Protokolls an den Prover schicken. In diesem Fall bekommt er vom Prover den Wert $K(x)$ und zwar für alle x der Länge n. Bei Eingabe von $(x\%x)$

[8] Dies ist in unserem Fall der Verifier.

[9] Da jeder effiziente Algorithmus, in diesem Fall der (betrügerische) Verifier V^* es nur mit vernachlässigbarer Wahrscheinlichkeit schafft, dem Prover einen String s aus der P-ausweichenden pseudozufälligen Menge T_n zu schicken.

liefert der Prover, bei Interaktion mit dem betrügerischen Verifier V^*, den Wert $K(x)$ mit Wahrscheinlichkeit 1.

Ein effizienter Simulator M_{V^*} müßte also mit überwältigender Wahrscheinlichkeit $K(x)$ berechnen können. Solange aber die Sprache S_K nicht in BPP liegt, ist dies für einen effizienten Algorithmus nicht möglich. Somit kann ein effizienter Simulator die view des Verifiers, bei Interaktion mit P, nicht wie gefordert simulieren. □

Aus beiden Behauptungen zusammen folgt der Satz. □

Dieser Beweis stammt von Goldreich und Krawczyk [GK90]. Unabhängig davon hat D. Simon [Sim88] ein CZK-Protokoll angegeben, welches ebenfalls nicht unter sequentieller Komposition abgeschlossen ist. Simon verwendet für sein Beispiel die Voraussetzung, daß es sichere Verschlüsselungs-Verfahren gibt.

Bald nachdem Goldwasser, Micali und Rackoff [GMR85] ihre Computational-Zero-Knowledge Definition veröffentlicht haben, wurde vermutet,[10] daß diese Definition nicht unter sequentieller Komposition abgeschlossen ist.

Stärkere Computational-Zero-Knowledge Definitionen wurden von verschiedenen Autoren angegeben. Diese stärkeren Definitionen unterscheiden sich von der Original CZK-Definition dadurch, daß sie auch Verifier berücksichtigen, die eine polynomiell große Hilfseingabe haben können. Auch mit dieser Hilfseingabe darf es dem Verifier nicht möglich sein, nützliche Informationen aus dem Prover zu extrahieren.

Bei diesen stärkeren Definitionen ist diese Schwachstelle der Original CZK-Definition beseitigt, so daß es auch nicht möglich ist, ein Beispiel obiger Art zu konstruieren. Tatsächlich sind diese stärkeren Definitionen unter sequentieller Komposition abgeschlossen. In der Journalversion [GMR89] ihres bedeutenden Papers haben Goldreich, Micali und Rackoff ihre Original Definition durch eine stärkere Definition ersetzt.

4.4.3 Eine stärkere CZK-Definition

Wie wir im letzten Abschnitt gesehen haben, ist die CZK-Definition nicht unter sequentieller Komposition abgeschlossen. Dies liegt daran, daß diese Definition nicht berücksichtigt, daß ein Verifier außer der gemeinsamen Eingabe x zusätzliche Informationen haben kann.

Typischerweise wird ein Zero-Knowledge Protokoll als Unterprotokoll in größeren kryptographischen Protokollen eingesetzt. In dieser Situation kommuniziert der Prover mit einem Verifier, der aus dem bisherigen Protokollverlauf Zugriff auf zusätzliche Informationen besitzt. Diese Informationen könnten, wie im Beispiel aus dem vorigen Abschnitt, nützlich sein, um wichtiges Wissen aus dem Prover zu extrahieren. Besonders dann, wenn diese Informationen einen Bezug zur gemeinsamen Eingabe x besitzen.

Es wird nun, um Zero-Knowledge Protokolle in solchen Anwendungen einsetzen zu können, eine Definition gebraucht, die berücksichtigt, daß der Verifier eine beliebige polynomial

[10] Allerdings wurde es erst 1988 von Goldreich und Krawczyk [GK90] bewiesen. Stärkere Zero-Knowledge Definitionen kennt man schon seit 1987 z.B. [TW87].

lange private Zusatzeingabe z haben kann. Um dies zu modellieren, benützen wir das Erweiterte Interaktive Beweissystem und definieren in bezug auf dieses den Zero-Knowledge Begriff.

Definition 4.12 (CZK mit Zusatzeingabe)
*Sei (P, V) ein Erweitertes Interaktives Beweissystem für die Sprache S. Mit $P_S(x)$ bezeichnet man die Menge aller Strings y, mit denen der Prover den Verifier von $x \in S$ überzeugen kann. Wir nennen (P, V)*__Computational Zero-Knowledge mit Zusatzeingabe__*, falls für jede polynomial zeitbegrenzte ITM V^* ein probabilistischer, in der Länge seiner ersten Eingabe polynomial zeitbeschränkter Algorithmus M_{V^*} existiert, so daß bei beliebiger Wahl von $y \in P_S(x)$ die zwei Ensembles*

- $\{\langle P(y), V^*(z)\rangle(x)\}_{x \in L,\, y \in P_S(x),\, z \in \{0,1\}^*}$
- $\{M^*(x, z)\}_{x \in L,\, z \in \{0,1\}^*}$

nicht effizient unterscheidbar sind.

Das bedeutet hier, daß für jeden probabilistischen, in der Länge seiner ersten Eingabe polynomial zeitbeschränkten Algorithmus D, jedes Polynom $p(\cdot)$, alle ausreichend großen $x \in S$, alle $y \in P_S(x)$ und $z \in \{0,1\}^*$ gilt, daß

$$|Prob(D(x, z, \langle P(y), V^*(z)\rangle(x)) = 1) - Prob(D(x, z, M^*(x, z)) = 1)| < \frac{1}{p(|x|)}$$

Die Definition soll folgende intuitive Idee formal fassen: Ein Protokoll ist CZK mit Zusatzeingabe z und gemeinsamer Eingabe x, falls sich alles was sich effizient aus x und z bei Interaktion mit dem Prover berechnen läßt, auch nur aus x und z ohne jede Interaktion mit dem Prover berechnen läßt.

Der String z, in der obigen Definition, stellt die mögliche Information des Verifier vor Beginn des Protokolls dar. Dies wird hier dadurch modelliert, daß der Verifier dieses z als eine Zusatzeingabe bekommt. Auch der Prover kann in diesem Modell eine Zusatzeingabe in Form des Strings y bekommen. Beide Zusatzeingaben sind privat, d.h. sie sind der anderen Partei nicht bekannt.

Auch der Algorithmus D, der versucht die Ensembles zu unterscheiden, bekommt als Zusatzeingabe die Zusatzinformation z des Verifiers.

Zu beachten ist, daß selbst nichtuniforme, polynomial große Algorithmen, nicht effizient die Ausgabe des Simulator M_{V^*}, in obiger Definition, von der view des Verifiers V^* unterscheiden können.

Das heißt:
Für jede polynomial große Schaltkreisfamilie $\{C_n\}_{n \in \mathbf{N}}$, jedes Polynom $p(\cdot)$, für alle ausreichend großen n, alle $x \in S$ der Länge n, alle $y \in P_S(x)$ und beliebige $z \in \{0,1\}^*$

gilt:

$$|\mathrm{Prob}(C_n(x,z,\langle P(y),V^*(z)\rangle(x))=1)-\mathrm{Prob}(C_n(x,z,M_{V^*}(x,z))=1)|<\frac{1}{p(|x|)}$$

Beweisskizze:

Wir nehmen das Gegenteil an und konstruieren daraus einen Widerspruch. Es existiere also ein Verifier V^* und eine polynomial große Schaltkreisfamilie $\{C_n\}_{n\in\mathbf{N}}$, so daß es für unendlich viele n Tripel der Form (x,y,z) gibt, für welche der Schaltkreis C_n einen nicht vernachlässigbaren Unterschied feststellen kann.

Dazu beobachten wir folgendes. Der Verifier, der Simulator und der Algorithmus D sind alle polynomial zeitbeschränkt in bezug auf die gemeinsame Eingabe x. Alle drei können somit nur einen poly($|x|$) langen Präfix von der Zusatzeingabe z lesen.

V^* ist durch ein festes Polynom $Q_1(\cdot)$ in der Laufzeit beschränkt. Der zum Verifier V^* passende Simulator M_{V^*} ist nun auch fixiert und deswegen auch durch ein festes Polynom $Q_2(\cdot)$ in der Laufzeit begrenzt.

Da nun eine Aussage über alle polynomiell zeitbeschränkten Algorithmen D in der Definition gemacht wird, werden auch solche Algorithmen D betrachtet, deren Laufzeit durch ein größeres Polynom beschränkt ist. Das heißt, diese Algorithmen laufen länger als der Verifier und der Simulator und können deshalb auch einen größeren Teil der Zusatzeingabe z lesen, falls diese ausreichend lang ist.

Um einen Widerspruch zu bekommen kodieren wir die Beschreibung des Schaltkreises C_n zusammen mit der Zusatzeingabe z zu einer neuen Zusatzeingabe z''. Sei dazu $q(\cdot)$ ein Polynom, daß die Laufzeit des Simulators und des Verifiers, sowie die Größe des Schaltkreises C_n begrenzt. Dann erzeugen wir, aus z, einen String z', der genau die Länge von $q(n)$ besitzt.

1.Fall: $|z|<q(n)$
Der String z wird mit einer Standard Paddingtechnik auf die Länge $q(n)$ vergrößert.

2.Fall: $|z|>q(n)$
Der String z wird bei Länge $q(n)$ abgeschnitten.

Um den gewünschten String z'' zu bekommen, hängen wir an den String z' die Kodierung des Schaltkreises C_n an. Der String z'' ist also so konstruiert worden, daß der Verifier und der Simulator nicht genügend Zeit haben, um bei Zusatzeingabe von z'' die Kodierung von C_n zu lesen. Das bedeutet natürlich, daß:

$$\begin{aligned} M_{V^*}(x,z'') &= M_{V^*}(x,z) \text{ und} \\ \langle P(y),V^*(z'')\rangle(x) &= \langle P(y),V^*(z)\rangle(x) \end{aligned}$$

Ein Auswertungsalgorithmus für Schaltkreise ist ein Algorithmus, der als Eingabe einen Schaltkreis und dessen Eingabe bekommt und damit die Funktion des Schaltkreises auf dessen Eingabe berechnet. Mit Hilfe eines solchen Auswertungsalgorithmus D, der als Eingabe das z'' bekommt, erhalten wir einen uniformen Algorithmus D, so daß

$$D(x, z'') = C_n(x, z)$$

ist. Das gibt uns also einen uniformen Algorithmus D, welcher die Ensembles genau wie der nichtuniforme C_n unterscheiden kann. Einen solchen Algorithmus gibt es aber nach der Definition von CZK mit Zusatzeingabe nicht. Das ist ein Widerspruch, also ist die Annahme, daß es nichtuniforme Algorithmen gibt, die die Ensembles unterscheiden können, falsch.

Die Original CZK Definition aus [GMR85] hat sich aufgrund der Nichtabgeschlossenheit unter sequentieller Komposition als nicht ausreichend erwiesen. Die Definition CZK mit Zusatzeingabe hat nicht diesen Mangel der Original Definition und konnte sich deshalb durchsetzten. Wenn im folgenden von CZK die Rede ist bezieht sich dies auf die CZK mit Zusatzeingabe Definition.

Die Black Box Definition

Die stärkste bekannte Definition der Zero-Knowledge Eigenschaft ist die Black-Box Definition.

Definition 4.13 (Black Box Zero-Knowledge)
Ein Interaktives Protokoll (P, V) heißt **Black Box Zero-Knowledge** *für eine Sprache S, falls es einen polynomial zeitbeschränkten universellen Simulator M gibt, der jeden (nichtuniformen) Verifier V^* als Black Box benützend die Interaktion von V^* mit dem Prover simulieren kann, so daß die zwei Ensembles von Zufallsvariablen*

- $\{view^P_{V^*(z)}(x)\}_{x \in S, z \in \{0,1\}^*}$
- $\{M(x, z)\}_{x \in S, z \in \{0,1\}^*}$

nicht effizient unterscheidbar sind.

Es ist klar, daß die Sprachen, die ein Black Box Zero-Knowledge Protokoll haben auch ein Zero-Knowledge Protokoll nach der Originaldefinition von [GMR85] haben. Goldreich und Oren haben gezeigt, daß die Klasse der Black Box Zero-Knowledge Protokolle in der Klasse der Zero-Knowledge Protokolle mit Zusatzeingabe enthalten ist (siehe dazu [GO94]).

Ein Problem bei der obigen Black Box Definition ist folgendes:

Der universelle Simulator benutzt alle Verifier als eine Art Black Box, das heißt, er gibt dem jeweiligen Verifier bestimmte Eingaben vor und startet diesen dann. Dabei muß er dem Verifier auch die Zufallsbits liefern, die dieser für seine Berechnungen benötigt. Jeder Verifier ist zwar polynomiell in der Laufzeit begrenzt und kann deshalb auch nur eine durch dieses Polynom nach oben begrenzte Anzahl von Zufallsbits benützen, aber es gibt kein festes Polynom, das die Zufallsbits aller Verifier nach oben begrenzt. Unser universelle Simulator muß aber für jeden Verifier die Zufallsbits liefern können, will er ihn wirklich als Black Box benützen. Wie macht er das aber bei einem Verifier, der mehr Zufallsbits benötigt, als der Simulator aufgrund seiner Laufzeitbegrenzung liefern kann?

Um dieses Problem, das bei den anderen Definitionen nicht auftrat, weil dort jeder Verifier seinen spezifischen Simulator haben durfte, zu lösen, schwächt man die Definition etwas ab.

Für jedes Polynom $p(\cdot)$ fordert man die Existenz eines eigenen Simulators, der alle Verifier simuliert, die höchstens $p(|x|)$ viele Zufallsbits verwenden bei allen Eingaben x.

4.4.4 Abschlußeigenschaft bei sequentieller Komposition

Goldreich und Oren haben in [GO94] gezeigt, daß die stärkere Definition, CZK mit Zusatzeingabe, unter sequentieller Komposition abgeschlossen ist. Dieses bedeutende Resultat ist unter dem Namen „Sequential Composition Lemma“ bekannt geworden. Bevor wir dieses Resultat aus [GO94] nachvollziehen, wollen wir nochmal die Bedeutung dieses Resultats aufzeigen.

Die Bedeutung des Resultats

Das „Sequential Composition Lemma“ ist ein sehr wichtiges Werkzeug zur Konstruktion von Zero-Knowledge Protokollen. Ein typisches Zero-Knowledge Protokoll besteht aus der mehrfachen Wiederholung eines Zero-Knowledge Basisprotokolls. Dieses Basisprotokoll bietet eine Lücke zwischen den Akzeptanz-Wahrscheinlichkeiten von Strings die in der Sprache sind und Strings die nicht in der Sprache sind. Allerdings ist diese Lücke oft nicht groß genug, um der Definition eines Interaktiven Beweissystems zu genügen.

Durch das Wiederholen des Basisprotokolls wird die Lücke zwischen den Wahrscheinlichkeiten vergrößert. Dabei garantiert das „Sequential Composition Lemma", daß das Protokoll trotz der Wiederholungen die Zero-Knowledge-Eigenschaft beibehält, falls das Basisprotokoll die Zero-Knowledge Eigenschaft hatte.

In dieser Arbeit wird nur eine eingeschränkte Version des „Sequential Composition Lemma" verwendet. Wir beweisen aus diesem Grund nur die eingeschränkte Version. Die Einschränkung bezieht sich darauf, daß hier nur die sequentielle Komposition eines Basisprotokolls betrachtet wird, während in der allgemeinen Version die sequentielle Komposition beliebiger Basisprotokolle behandelt wird.

Lemma 4.1 (Sequential Composition Lemma)

Sei ein Prover P mit der CZK Eigenschaft für eine Sprache S gegeben. Die letzte Nachricht, die von P bei Eingabe x gesendet wird, sei ein spezielles Beweisende-Symbol. Sei $Q(\cdot)$ ein Polynom. Mit P_Q bezeichnen wir folgende ITM, welche bei Eingabe von x in $Q(|x|)$ Phasen abläuft: In jeder Phase startet P_Q die Maschine P auf der Eingabe x. Die Maschine P_Q benutzt in jeder der $Q(|x|)$ Phasen unabhängige Zufallsbits. Dann ist auch der Prover P_Q CZK. Falls der Prover P sogar PZK ist, dann ist auch der Prover P_Q PZK.

Das Beweisende Symbol wird für beweistechnische Zwecke eingeführt. Jeder Prover P^* läßt sich so erweitern, daß seine letzte Nachricht aus dem Beweisende Symbol besteht. Durch diese Erweiterung wird die Zero-Knowledge Eigenschaft nicht beeinflußt. Das Lemma berücksichtigt durch das Polynom $Q(\cdot)$ eine polynomiell große sequentielle Komposition eines Basisprotokoll.

Beweisidee:

> Sei V^* ein beliebiger, polynomial zeitbeschränkter Verifier, der mit dem Prover P_Q, aus obiger Definition, kommuniziert. Um zu zeigen, daß das Protokoll (P_Q, V^*) Zero-Knowledge ist, müssen wir einen im Erwartungswert polynomial zeitbeschränkten Simulator M_{V^*} angeben, der die view des Verifiers V^* bei Interaktion mit P_Q simulieren kann. Wie geht der Simulator M_{V^*} vor?
>
> Die Idee ist hier, den Simulationsprozeß auch in $Q(|x|)$ Phasen aufzuteilen, wobei diese zu den $Q(|x|)$ Phasen des Provers P_Q korrespondieren. Jede Phase wird durch den Simulator für den Prover P simuliert[11]. In jeder Phase kann der Verifier Informationen sammeln, die er in die nächste Phase, in Form einer Zusatzeingabe, mitnehmen kann.
>
> Dafür muß zunächst die Ausführung der Maschine V^* auch in Phasen aufgeteilt werden. Im Programm der Maschine V^* werden im allgemeinen solche Phasen nicht berücksichtigt sein. Diese Aufteilung in Phasen muß sozusagen

[11] Die Existenz des Simulators für P ist durch die Vorraussetzung, daß P Zero-Knowledge ist garantiert

von außen, durch die Interaktion geschehen. Dazu wird die Phasenaufteilung von P_Q verwendet. Hier sehen wir jetzt den Grund für die Erweitung des Provers P bezüglich des Beweisende-Symbols. Immer wenn dieses Beweisende-Symbol auf dem Kommunikationsband erscheint, ist eine Phase beendet. Daraus folgt, daß unabhängig davon, wie der Verifier V^* arbeitet, die Interaktion von V^* mit P_Q, bei gemeinsamer Eingabe x als $Q(|x|)$ Iterationen eines Verifiers V^{**} mit dem Prover P betrachtet werden kann.

Behauptung1:

Es existiert eine polynomial zeitbeschränkte ITM V^{**}, so daß für jede gemeinsame Eingabe x und Zusatzeingabe z gilt, daß:

$$\langle P_Q, V^*(z)\rangle(x) \;=\; Z^{(Q(|x|))}$$

$$\text{wobei } Z^{(0)} \stackrel{\text{def}}{=} z \text{ und } Z^{i+1} \stackrel{\text{def}}{=} \langle P, V^{**}(Z^{(i)})\rangle(x)$$

Die Zufallsvariable $Z^{(Q(|x|))}$ beschreibt die Ausgabe von V^{**} nach $Q(|x|)$ hintereinanderfolgenden Interaktionen mit dem Basisprover P, bei gemeinsamer Eingabe x. Der Verifier V^{**} bekommt in der ersten Iteration des Protokolls auch die Zusatzeingabe z der Maschine V^*. Am Ende einer Iteration des Protokolls liefert die Maschine V^{**} eine Ausgabe, die bei der nächsten Iteration von der Maschine V^{**} wieder als Zusatzeingabe verwendet wird.

Beweis :
Vorbetrachtungen: Betrachten wir eine Interaktion von $V^*(z)$ mit dem Prover P_Q, bei gemeinsamer Eingabe x. Die Maschine V^* bekommt die Zusatzeingabe z.

Wir modifizieren die Maschine V^* ein wenig. Die Maschine V^* soll zunächst die Inhalte ihres Eingabebands, ihres Zufallsbands und ihres Zusatzeingabebands auf einen speziellen Bereich ihres Arbeitsbandes kopieren. Danach soll sie während des Protokolls nicht mehr auf diese drei Bänder zugreifen.

In einer aktiven Phase soll die Maschine V^* als erstes immer den Inhalt des Kommunikationsbandes auf einen weiteren speziellen Bereich des Arbeitsbandes kopieren und danach in dieser aktiven Phase nicht mehr auf das Kommunikationsband zugreifen. Da die Maschine V^* polynomial (in der Länge der gemeinsamen Eingabe) zeitbeschränkt ist, kann sie nur jeweils einen polynomiell langen Präfix von den Inhalten der Bänder kopieren.

Konstruktion der Maschine V^{**}

Wie geht die Maschine V^{**} aus der Maschine V^* hervor? Betrachten wir dazu eine Phase i des Protokolls (P_Q, V^*). In einer solchen Phase bekommt die Maschine V^{**} eine Zusatzeingabe – bezeichnet mit z' – welche sie auf das Arbeitsband der Maschine V^* kopiert. Dann simuliert die Maschine V^{**} eine einzelne Phase der Interaktion des Verifiers V^* mit dem

Prover P_Q, auf der Eingabe x. Dies macht sie, indem sie mit dem Inhalt des Arbeitsbandes von V^* die Interaktion mit dem Prover P durchführt. Diese Phase ist beendet, sobald der Prover P das Beweisende-Symbol schickt. Dann gibt die Maschine V^{**} die aktuellen Inhalte des Arbeitsbandes von V^* aus und hält an. Diese Ausgabe bekommt die Maschine V^{**} in der Phase $i+1$ als Zusatzeingabe und so setzt sich dieser Prozeß über alle Phasen fort.

Dieser Prozeß muß nun leicht modifiziert werden um auch die erste Phase des Protokolls (P_Q, V^*) zu handhaben. Dazu nehmen wir an, daß sich die Kodierung einer Zusatzeingabe für die Maschine V^* von einer Kodierung der Inhalte eines Arbeitsbandes unterscheidet.

Dann kopiert die Maschine V^{**} nur dann die Zusatzeingabe z' auf das Arbeitsband der Maschine V^* falls z' einer Kodierung eines Arbeitsbandes entspricht. Im anderen Fall, wenn z' einer Kodierung einer Zusatzeingabe für die Maschine V^* entspricht, dann startet die Maschine V^{**} den Verifier V^* mit einem leeren Arbeitsband. In diesem Fall betrachtet der Verifier V^* alle Bänder mit Lesezugriff als seine eigenen und beginnt die Inhalte dieser Bänder auf sein Arbeitsband zu kopieren (wie oben beschrieben). □

Die Maschine V^{**} ist eine polynomial zeitbeschränkte ITM mit privater Zusatzeingabe. Da nach Vorraussetzung der Prover P die CZK Eigenschaft hat, gibt es einen im Erwartungswert polynomial zeitbeschränkten[12] Simulator M^{**}, der die Interaktion von V^* und P simulieren kann.
Das heißt formal:
Für jeden probabilistischen polynomial zeitbeschränkten (in$|x|$) Algorithmus D , jedes Polynom $p(\cdot)$, alle ausreichend langen $x \in S$ und allen $z \in \{0,1\}^*$, gilt:

$$|\text{Prob}(D(x, z, \langle P, V^{**}(z)\rangle(x) = 1) - \text{Prob}(D(x, z, M_{V^{**}}(x, z)) = 1)| \ < \ \frac{1}{p(|x|)}$$

Konstruktion des Simulators M_{V^*}

Wir werden im folgenden einen Simulator M_{V^*} konstruieren, der in der Lage ist die Interaktion der Maschine V^* mit dem Prover P_Q zu simulieren. Der Simulator M_{V^*} nutzt dazu den Simulator $M_{V^{**}}$, der nach Vorraussetzung existiert, aus. Auf die Eingabe (x, z) setzt der Simulator M_{V^*} $z^{(0)} = z$. In der i-ten Phase berechnet der Simulator M_{V^*} $z^{(i)}$, indem er den Simulator $M_{V^{**}}$ mit der Eingabe $(x, z^{(i-1)})$ startet und die Ausgabe, die dieser liefert, als $z^{(i)}$ setzt. Nach $Q(|x|)$ Phasen hält der Simulator M_{V^*} und gibt $z^{(Q(|x|))}$ aus.

Jetzt ist zu zeigen, daß

1. Die Laufzeit des Simulators M_{V^*} im Erwartungswert polynomiell beschränkt ist.
2. Die Ausgabe des Simulators nicht effizient unterscheidbar ist von der view des Verifiers V^* bei Interaktion mit dem Prover P_Q

[12] In der Länge seiner ersten Eingabe.

Zu1:
Der Simulator ist im Erwartungswert polynomial zeitbeschränkt in der Länge seiner ersten Eingabe. Er durchläuft polynomiell viele Phasen und in jeder Phase findet nur die Berechnung des im Erwartungswert polynomial zeitbeschränkten Simulators $M_{V^{**}}$ statt.

Zu2:

Behauptung2:

Für jeden probabilistischen Algorithmus D, dessen Laufzeit polynomiell in der Länge seiner ersten Eingabe ist, jedes Polynom $p(\cdot)$, alle ausreichend großen $x \in S$ und alle $z \in \{0,1\}^*$ gilt:

$$|\mathrm{Prob}(D(x, z, \langle P_Q, V^*(z)\rangle(x) = 1) - \mathrm{Prob}(D(x, z, M^*(x, z)) = 1)| \; < \; \frac{1}{p(|x|)}$$

Beweis :
Wir definieren uns wie folgt $Q(|x|) + 1$ Hybrids. Das i-te Hybrid, $0 \leq i \leq Q(|x|)$, korrespondiert zu folgendem Zufallsprozeß. Der Verifier V^{**} bekommt die Eingabe (x, z) und kommuniziert die ersten i Phasen mit dem Prover P. Nach der i-ten Phasen liefert er eine Ausgabe, die wir mit $Z^{(i)}$ bezeichnen. In den verbleibenden $Q(|x|) - i$ Phasen wird die Maschine M^{**} gestartet. Formal wird das i-te Hybrid $H^{(i)}$ wie folgt definiert:

$$H^{(i)}(x, z) \;\stackrel{\text{def}}{=}\; M^{**}_{Q(|x|)-i}(x, Z^{(i)})$$

$$\text{wobei } Z^{(0)} \;\stackrel{\text{def}}{=}\; \text{ und } Z^{(j+1)} \;\stackrel{\text{def}}{=}\; \langle P, V^{**}(Z^{(j)})\rangle(x)$$

$$M^{**}_0(x, z') \;\stackrel{\text{def}}{=}\; (x, z') \text{ und } M^{**}_j(x, z') \;\stackrel{\text{def}}{=}\; M^{**}_{j-1}(x, M^{**}(x, z'))$$

Betrachten wir das nullte Hybrid $H^{(0)}(x, z)$, das durch obigen Prozeß definiert ist, so fällt auf, daß dieses gleich $M^*(x, z)$ ist. Wegen Behauptung1 folgt, daß das $Q(|x|)$-te Hybrid $H^{(Q(|x|))}(x, z)$ gleich $\langle P_Q, V^*(z)\rangle(x)$ ist. Wenn wir jetzt zeigen können, daß alle benachbarten Hybrids nicht effizient unterscheidbar sind, so folgt daraus, daß auch die extremen (am weitesten auseinanderliegenden) Hybrids $H^{(0)}(x, z)$ und $H^{(Q(|x|))}(x, z)$ nicht effizient unterscheidbar sind.

Der Grund ist folgender: Der Unterschied zwischen nicht effizient unterscheidbaren Objekten ist vernachlässigbar. Eine vernachlässigbare Funktion bleibt vernachlässigbar auch wenn sie mit einem beliebigen Polynom multipliziert wird. Wir haben hier nur polynomiell viele Hybrids zwischen den extremen Hybrids und daraus folgt, daß diese auch nicht effizient unterscheidbar sein müssen.

Betrachten wir nun die beiden benachbarten Hybrids i und $i - 1$.

$$\begin{aligned} H^{(i)}(x, z) &= M^{**}_{Q(|x|)-i}(x, \langle P, V^{**}(Z^{(i-1)})\rangle(x)) \\ H^{(i-1)}(x, z) &= M^{**}_{Q(|x|)-i}(x, M^{**}(x, Z^{(i-1)})) \end{aligned}$$

Nehmen wir das Gegenteil an, also:

Es gibt einen probabilistischen polynomial zeitbeschränkten Algorithmus D, sowie ein Polynom $p(\cdot)$, so daß für unendlich viele $x \in S$ ein $z \in \{0,1\}^*$ existiert, so daß

$$|\mathrm{Prob}(D(x, z, \langle P_Q, V^*(z)\rangle(x) = 1) - \mathrm{Prob}(D(x, z, M^*(x, z)) = 1)| > \frac{1}{p(|x|)}.$$

Dann folgt, daß es für alle solche x und z es ein $i \in \{1, \ldots, Q(|x|)\}$ gibt, so daß

$$|\mathrm{Prob}(D(x, z, H^{(i)}(x, z)) = 1) \ - \ \mathrm{Prob}(D(x, z, H^{(i-1)}(x, z)) = 1)| > \frac{1}{Q(|x|) \cdot p(|x|)}$$

Sei $c(n) \stackrel{\text{def}}{=} 1/(Q(n) \cdot p(n))$ Mit einem Durchschnittsargument folgt, daß es ein festes z' gibt, so daß für jedes x, z und i wie oben gilt: (Einsetzen der Definition von $H^{(i)}$ und $H^{(i-1)}$)

$$|\mathrm{Prob}(D(x, z', M^{**}_{Q(|x|)-i}\langle P, V^{**}(z')\rangle(x)) = 1) - \mathrm{Prob}(D(x, z', M^{**}_{Q(|x|)-i}(M^{**}(x, z'))) = 1)| > c(|x|).$$

Jetzt haben wir fast den gewünschten Widerspruch. Die Zufallsvariablen $(x, z', \langle P, V^{**}(z')\rangle(x))$ und $(x, z', M^{**}(x, z')$ können mit den Algorithmen D und M^{**} effizient unterschieden werden, falls das i bekannt ist. Aus den Bemerkungen zur Definition von CZK mit Zusatzeingabe folgt, daß die Ausgabe der Maschine M^{**} von der Interaktion des Verifiers V^{**} mit dem Prover sogar von nichtuniformen polynomial großen Schaltkreisen nicht unterschieden werden kann. Das bedeutet daß es auf das i gar nicht mehr ankommt. Wenn die obigen Zufallsvariablen unterschieden werden können, so steht dies im Widerspruch dazu, daß M^{**} ein Simulator für (P, V^{**}) ist. Daraus folgt die Behauptung2.

Aus beiden Behauptungen folgt das Lemma. □

4.4.5 Abschlußeigenschaft bei paralleler Komposition

In diesem Abschnitt untersuchen wir die Frage, ob CZK-Protokolle unter paralleler Komposition abgeschlossen sind.

Um die Fehlerwahrscheinlichkeit eines Interaktiven Beweissystems zu senken wird das Basisprotokoll sequentiell oder parallel wiederholt. Allerdings steigt bei sequentieller Wiederholung die Anzahl der Runden des Protokolls. Dies ist vor allem aus praktischen Gesichtspunkten unerwünscht. Dieser Nachteil entfällt bei parallelen Wiederholungen des Basisprotokolls. Hier bleibt die Anzahl der Runden unverändert.

Natürlich möchte man diese Vorteile der parallelen Wiederholung auch gerne auf die speziellen Interaktiven Beweissysteme, die die Zero-Knowledge Eigenschaft haben, übertragen. In der ersten Zeit nach der Einführung des Zero-Knowledge Begriffs durch Goldwasser, Micali und Rackoff [GMR85] wurde vermutet, daß folgende Behauptung wahr ist:

Behauptung:
Sind (P_1, V_1) und (P_2, V_2) zwei Interaktive Beweissysteme. Hat jedes der zwei Beweissysteme die Zero-Knowledge Eigenschaft, so hat die parallele Komposition der beiden Protokolle ebenfalls die Zero-Knowledge Eigenschaft.

Diese Behauptung konnte nicht bewiesen werden, aber viele Forscher vermuteten trotzdem, daß diese Behauptung wahr ist. Diese berühmte Behauptung wurde unter dem Namen „Parallel-Composition-Conjecture" bekannt.

Goldreich und Krawczyk, die auch schon die Abgeschlossenheit unter sequentieller Komposition der CZK-Definition widerlegt haben, konnten wieder ihre Forschungsresultate über ausweichende, pseudozufällige Mengen verwenden, und damit auch die obige Behauptung widerlegen. Sie zeigen, daß auch die stärksten bekannten Computational-Zero-Knowledge Definitionen[13] nicht unter paralleler Komposition abgeschlossen sind.

Aufgrund des Resultats von Goldreich und Krawczyk kann nicht argumentiert werden, daß ein Protokoll Computational Zero-Knowledge ist, weil es sich nur durch parallele Komposition von einzelnen Computational Zero-Knowledge Protokollen zusammensetzt. Das Resultat schließt allerdings nicht aus, daß spezielle Protokolle bei parallelen Wiederholungen die Computational Zero-Knowledge Eigenschaft beibehalten.

Sie nützen in ihrem Beweis aus, daß die Rechenzeit des Provers nicht beschränkt ist. Ihr Beweis kommt dabei ohne jede unbewiesene Annahme aus. Feige und Shamir [FS90a] zeigten, allerdings basierend auf kryptographischen Annahmen, daß auch Perfekt ZK-Protokolle nicht unter paralleler Komposition abgeschlossen sind. Im Unterschied zu Goldreich und Krawczyk benötigt der Prover in ihrem Beweis nur polynomielle Rechenzeit.

Wir folgen im weiteren den Resultaten von Goldreich und Krawczyk [GK90].

Betrachten wir zunächst die Original CZK Definition. Wie wir bereits wissen, ist diese Definition nicht unter sequentieller Komposition abgeschlossen. Jedes Protokoll, welches nicht unter sequentieller Komposition abgeschlossen ist, kann leicht in ein Protokoll, das nicht unter paralleler Komposition abgeschlossen ist, umgeformt werden. Dazu müssen die Protokolle mit sogenannten „Dummy-Phasen" vergrößert werden. In einer Dummy-Phase macht eine Maschine nichts. Dies dient dazu, daß bei der parallelen Komposition der Protokolle, die ursprünglichen Basisschritte so zwischen die Dummy-Phasen verzahnt werden, daß eine Art sequentielle Ausführung der einzelnen Basisschritte simuliert wird. Dann wird wichtiges Wissen bei der parallelen Komposition genauso preisgegeben wie bei der sequentiellen Komposition.

Die stärkere CZK Definition, CZK mit Zusatzeingabe ist aber wie Goldreich und Oren [GO94] gezeigt haben unter sequentieller Komposition abgeschlossen. Dadurch läßt sich nicht analog wie oben argumentieren.

Um ein geeignetes Gegenbeispiel zur „Parallel-Composition-Conjecture" zu konstruieren, werden ähnliche Ideen wie im Beweis, daß CZK nicht unter sequentieller Komposition abgeschlossen ist, verwendet. Es werden auch wieder die ausweichenden und pseudozufälligen Eigenschaften gewisser Mengen ausgenützt. Hier haben wir es allerdings mit

[13]Das sind die CZK Definition mit Zusatzeingabe und die Black Box Definition.

nichtuniformen Algorithmen zu tun, und deshalb reicht die ausweichende Eigenschaft einer P-ausweichenden Menge nicht aus. Denn für jede Menge von Strings existiert ein nichtuniformer Verifier, der Elemente dieser Menge ausgeben kann. Der Grund ist, so ein nichtuniformer Verifier könnte solch einen Strings als Zusatzeingabe bekommen und kann ihn dann natürlich auch ausgeben.

Jeder solche nichtuniforme Verifier ist aber weiterhin polynomial zeitbeschränkt und kann deshalb auch nur polynomiell lange Zusatzeingaben nutzen bzw. den polynomiell langen Präfix einer größeren Zusatzeingabe. Dies werden wir nun ausnützen, indem wir nicht nur eine ausweichende Menge pro Eingabelänge betrachten, sondern eine Ansammlung von exponentiell vielen Mengen pro Eingabelänge. Selbst wenn ein nichtuniformer Verifier eine Zusatzeingabe bekommt in der Strings aus allen exponentiell vielen Mengen enthalten sind, kann es diese Zusatzeingabe nur zu einem polynomiellen Teil nutzen. Das heißt, die ausweichende Eigenschaft bleibt bei Benutzung von P/poly-ausweichenden pseudozufälligen Ensembles auch gegen nichtuniforme Verifier erhalten.

Satz 4.4
CZK mit Zusatzeingabe ist unter paralleler Komposition nicht abgeschlossen.

Beweis :
Um dies zu beweisen geben wir zwei Protokolle (P_1, V_1) und (P_2, V_2) an, welche die CZK mit Zusatzeingabe Definition erfüllen, solange sie unabhängig voneinander ausgeführt werden. Diese Protokolle sind, sobald sie parallel ausgeführt werden nicht mehr CZK. Beide Protokolle werden mit Hilfe von Dummy-Phasen synchronisiert. Mit diesen zusätzlich eingebrachten Dummy-Phasen besitzen beide Protokolle 5 Wechsel bzw. Schritte. Die Protokolle sind in Tabelle 4.1 dargestellt. Beide Protokolle benützen bei allen ihren Ausführungen dasselbe P/poly-ausweichende Ensemble $T^{(1)}, T^{(2)}, \ldots$ Sei K eine harte Boolsche Funktion, so daß die Sprache $S_K := \{x \mid K(x) = 1\}$ nicht in der Klasse BPP enthalten ist.

Betrachten wir zunächst das erste Protokoll (P_1, V_1).

Der Prover P_1 wählt ein i zufällig unter Gleichverteilung aus. Er schickt dieses i an den Verifier V_1. Nach zwei Dummy-Phasen in denen nichts passiert und die nur der Synchronisation dienen, wählt der Verifier V_1 einen String s der Länge $4n$ zufällig unter Gleichverteilung aus. Er schickt den String s an den Prover P_1. Der Prover überprüft, ob der String s Element der P-ausweichenden Menge $T_i^{(n)}$ ist. Falls ja, so schickt der Prover den Wert von $K(x)$ an den Verifier, falls nein, so schickt er nichts.

Der Simulator kann den Verifier simulieren. Hier könnte er ihn als Black Box benutzen. Wie simuliert der Simulator die zwei Schritte des Provers P_1? Der erste Schritt des Provers P_1 ist einen Index i aus $\{1, \ldots, 2^n\}$ zu wählen. Der Simulator macht genau dasselbe. Im zweiten Schritt schickt den Prover den Wert $K(x)$ nur dann, wenn es dem Verifier gelingt einen String s zu schicken, der Element der Menge $T_i^{(n)}$ ist. Durch die ausweichende Eigenschaft des P/poly ausweichenden Ensemles $T^{(1)}, T^{(2)}, \ldots$ gelingt es dem Verifier nur mit vernachlässigbarer Wahrscheinlichkeit einen solchen String s zu finden. Das heißt, der Simulator braucht immer nur den Fall $s \notin T_i^{(n)}$ zu simulieren, um eine Ausgabe zu erzeugen, die nicht effizient von der view des Verifiers unterschieden werden kann.

Betrachten wir nun das zweite Protokoll (P_2, V_2)

Die erste Phase des Provers P_2 ist eine Dummy-Phase zur parallelen Synchronisation. Im zweiten Schritt des Protokolls wählt der Verifier zufällig unter Gleichverteilung einen Index j aus der Menge $\{1, \ldots, 2^n\}$ aus und schickt diesen dem Prover P_2. Der Prover wählt unter Gleichverteilung einen String r der P-ausweichenden Menge $T_j^{(n)}$ aus und schickt diesen an den Verifier. Diese Auswahl kann er aufgrund seiner nicht beschränkten Rechenzeit treffen. Es kommen noch zwei Dummy-Phasen, dann endet das Protokoll.

Im zweiten Protokoll ist der einzige Schritt des Provers die Auswahl des Strings r aus der P-ausweichenden Menge $T_j^{(n)}$. Der Simulator kann dies simulieren, indem er zufällig einen String r' aus $\{0,1\}^{4n}$ auswählt. Durch die pseudozufällige Eigenschaft der Menge $T_j^{(n)}$ kann diese Wahl nicht effizient von der Wahl des Provers unterschieden werden.

Der Simulator benötigt in beiden Protokollen nur polynomiell viel Zeit und erzeugt in beiden Protokollen eine Ausgabe, die von der view des jeweiligen Verifiers nicht effizient unterschieden werden kann. Daraus folgt, daß beide Protokolle CZK sind.

Die parallele Komposition

Wir konstruieren nun ein neues Protokoll (P, V) durch die parallele Komposition der Protokolle (P_1, V_1) und (P_2, V_2). Hält der Verifier V sich an das Protokoll, so ist dieses weiterhin CZK. Wie kann nun ein betrügerischer Verifier V^* vom Protokoll abweichen, um wichtiges Wissen aus dem Prover P zu extrahieren? Nun in dem ersten Schritt des Protokolls (siehe Tabelle 4.1) bekommt der Verifier V vom Prover einen Index i. Anstatt den Index j zufällig auszuwählen, schickt er den Index j = i an den Prover im zweiten Schritt des Protokolls. Im dritten Schritt des Protokolls bekommt er daraufhin einen String $r \in T_i^{(n)}$. Jetzt betrügt der Verifier V^* nochmal, indem er im vierten Schritt des Protokolls anstatt den String s zufällig zu wählen, den String s = r an den Prover schickt. Der Prover überzeugt sich davon, daß der String s, den er bekommen hat, tatsächlich ein Element der ausweichenden Menge $T_i^{(n)}$ist und schickt dann dem Verifier die Information $K(x)$. Dieser letzte Schritt kann von keiner probabilistischen polynomial zeitbeschränkten Maschine, wie der Simulator eine ist, simuliert werden, solange die Sprache S_K nicht in der Klasse BPP enthalten ist. Nach unserer Voraussetzung ist S_K nicht in BPP und deshalb kann das Protokoll nicht CZK sein. □

P_1	V_1	Stufe	P_2	V_2
$i \in_R \{1, \ldots, 2^n\} \rightarrow$		1	Mache nichts	
	Mache nichts	2		$\leftarrow j \in_R \{1, \ldots, 2^n\}$
Mache nichts		3	$r \in_R T_j^{(n)} \rightarrow$	
	$\leftarrow s \in_R \{0,1\}^{4n}$	4		Mache nichts
Falls $s \in T_i^{(n)}$ dann $K(x) \rightarrow$		5	Mache nichts	

Tabelle 4.1: Die beiden Protokolle (P_1, V_1) und (P_2, V_2)

Kapitel 5

Einige klassische Zero-Knowledge Beweise.

5.1 Einleitung

In diesem Kapitel sollen einige Beispiele für Interaktive Beweissysteme, die die Zero-Knowledge Eigenschaft haben, vorgestellt werden. Durch die Beispiele werden die Definitionen der letzten Kapitel, insbesondere die Definitionen von Interaktiven Beweissystemen und die Zero-Knowledge Eigenschaft gut veranschaulicht. Außerdem werden an diesen Beispielen die Techniken, die man benützt, um von einem Protokoll die geforderten Eigenschaften nachzuweisen, vorgeführt. Die vorgestellten Beispiele können aus heutiger Sicht als „klassische“ Zero-Knowledge Protokolle betrachtet werden.

5.2 Ein Zero-Knowledge Beweissystem für GI

In diesem Abschnitt wird ein interaktives Beweissystem für Graph-Isomorphie beschrieben, welches die Zero-Knowledge Eigenschaft hat. Die Sprache GI ist in NP[1] und damit ist klar, daß sie ein effektives (traditionelles) Beweissystem hat. Interessant ist also nur, wie sich dies in Zero-Knowledge Manier durchführen läßt.

5.2.1 Ein Interaktives Beweissystem für GI mit der Perfekt Zero-Knowledge Eigenschaft

Sind die Graphen G_0 und G_1 isomorph, so bezeichet ϕ einen Isomorphismus zwischen den beiden Graphen, so daß $G_1 = \phi(G_0)$.

[1] Man rät nichtdeterministisch den Isomorphismus und überprüft diesen in Polynomialzeit.

- Gemeinsame Eingabe: Zwei Graphen G_0 und G_1, jeweils mit Knotenmenge $N = \{1, \ldots, n\}$
  Wiederhole die folgenden Schritte n mal:

  1. **P1:**
     $\pi \in_R \text{Perm}(N)$
     $H = \pi(G_1)$
     $P \longrightarrow V \ : \ H$
  2. **V1:**
     $i \in_R \{0, 1\}$
     $V \longrightarrow R \ : \ i$
  3. **P2:**
     Falls i $= 1$, dann $\rho = \pi$
     Falls i $\neq 1$, dann $\rho = \pi \circ \phi$
     $P \longrightarrow V \ : \ \rho$
  4. **V2:**
     Falls $H \neq \rho(G_i)$, dann verwerfe und halte
     sonst weiter.

- War für alle n Iterationen $H = \rho(G_i)$, dann akzeptiere den Beweis.

Erläuterung des Protokolls

- zu P1:
  Der Prover wählt zufällig eine Permutation π aus der Menge Perm(N) aller Permutationen auf der Knotenmenge N der Graphen. Danach berechnet er den Graphen H als Permutation von G_1 mit Hilfe von π. D.h. der Graph H ist ein beliebiger isomorpher Graph von G_1. Der Prover schickt den Graphen H an den Verifier.

- zu V1:
  Der Verifier wählt zufällig ein Bit i aus. Dieses Bit repräsentiert den Wunsch des Verifiers, entweder den Isomorphismus zwischen H und G_0, oder den Isomorphismus zwischen H und G_1 sehen zu wollen. Dieser Wunsch ist eine Herausforderung an den Prover, der ihm diesen Wunsch erfüllen muß, um sich die Chance zu erhalten den Verifier am Ende zum Akzeptieren zu bringen. Der Verifier schickt das Bit i an den Prover.

- zu P2:
  Falls i $= 1$, dann will der Verifier einen Isomorphismus zwischen G_1 und H sehen. Der Prover schickt den Isomorphismus $\rho = \pi$. Falls i $= 0$, dann will der Verifier einen Isomorphismus zwischen H und G_0 sehen. Der Prover nimmt den Isomorphismus ϕ (um G_0 in G_1 abzubilden) und komponiert mit π einen Isomorphismus ρ zwischen

G_0 und H. Er schickt ρ an den Verifier. Der Fall $i \notin \{0,1\}$, wird wie der Fall i = 0 behandelt[2].

- zu V2:
  Der Verifier überprüft, ob der Prover ihm seinen durch das Bit i geäußerten Wunsch erfüllt hat.
  Falls ja, also ρ ein Isomorphismus zwischen H und G_i ist, setzt er das Protokoll fort. Falls nicht, beendet der Verifier das Protokoll und verwirft.

Satz 5.1
Obiges Protokoll ist ein PZK-Beweissystem für GI.

Beweis :
Teil1: Zu zeigen: Das Protokoll ist ein Interaktives Beweissystem.

- *Durchführbarkeit:*
  Sind die Graphen G_0 und G_1 isomorph zueinander, so kann der im Protokoll beschriebene Prover immer die passende Permutation liefern. Das bedeutet, der Verifier akzeptiert mit Wahrscheinlichkeit 1. $\Rightarrow$ das Protokoll ist durchführbar.

- *Korrektheit:*
  Sind die Graphen G_0 und G_1 nicht isomorph, so gibt es keinen Graphen H, der zu beiden Graphen G_0 und G_1 isomorph sein kann.
  Egal wie der Prover nun diesen Graphen H konstruiert, es gibt ein $i \in \{0,1\}$, so daß H und G_i nicht isomorph sind. Also verwirft der Verifier in jeder Runde mindestens mit Wahrscheinlichkeit $\frac{1}{2}$. Somit akzeptiert der Verifier in n Runden höchstens mit Wahrscheinlichkeit $(\frac{1}{2})^n$. $\Rightarrow$ das Protokoll ist korrekt. Damit ist das Protokoll ein Interaktives Beweissystem.

Teil2: Zu zeigen das Beweissystem ist PZK.

Betrachten wir das Protokoll mit dem ehrlichen Verifier V. Dieser bekommt falls die beiden Graphen isomorph sind, in jeder Runde einen zu den Eingangspraphen zufälligen isomorphen Graphen H. Dann wählt er zufällig ein Bit i aus und bekommt daraufhin einen Isomorphismus von G_i und H zu sehen. Dies bringt ihm keinen Berechnungsvorteil ein, denn er kann nun auch nicht mehr berechnen als vor der Interaktion und zudem kann er sich selbst ohne Hilfe des Provers zufällige isomorphe Kopien der Eingangsgraphen erzeugen. Intuitiv ist das Protokoll für diesen Fall Zero-Knowledge. Was passiert eigentlich, falls der Prover zufällig den Isomorphismus zwischen G_1 und G_0 auswählt? Dann kann sich der Verifier diesen Isomorphismus zeigen lassen.

[2] Der Prover könnte in diesem Fall auch das Protokoll beenden, aber nach obiger Vorgehensweise erspart man sich die Betrachtung eines weiteren Falls $i \notin \{0,1\}$ in der Simulation.

Ist das Protokoll dann noch Zero-Knowledge? Der Verifier bekommt doch den Isomorphismus zwischen G_0 und G_1 mitgeteilt. Das Protokoll ist trotz dieser Möglichkeit Zero-Knowledge, denn die Wahrscheinlichkeit, daß der Prover gerade diesen Isomorphismus auswählt, ist äußerst gering. Genausogut hätte der Verifier zufällig einen der Eingangsgraphen permutieren können. Er hätte dann mit derselben Wahrscheinlichkeit den Isomorphismus gefunden und davon kann er ja nicht abgehalten werden.

Die Zero-Knowledge Eigenschaft muß auch gegen Verifier, die versuchen zu betrügen, erhalten bleiben. Ein solcher Verifier könnte z.B. seine Auswahl, zu welchem Eingangsgraph er den Isomorphismus zu H sehen will, nicht zufällig auswählen. Seine Auswahl könnte von Graphen, die er in früheren Runden gesehen hat, in irgendeiner Weise abhängen. Es ist erforderlich, daß über alle Verifier V^* auch über die, die beliebig aber polynomial zeitbeschränkt vom ehrlichen Verifier abweichen, argumentiert wird. Dies läßt sich durch einen formalen Beweis erreichen.

Wir zeigen dazu, daß es für alle Verifier V^* einen Simulator M_{V^*} gibt, so daß die Ensembles

- $\{view_{V^*}^P(x)\}_{x \in GI}$
- $\{M_{V^*}(x)\}_{x \in GI}$

identisch sind. Der Simulator muß außerdem im Erwartungswert polynomial zeitbeschränkt sein.

Die Laufzeit des Verifiers V^* sei durch das Polynom p beschränkt. Der Simulator wählt zu Beginn einen zufälligen Bitstring $\overline{r} \in \{0,1\}^{p(n)}$ aus. Diesen Bitstring schreibt er auf das Zufallsband des Verifiers V^*. Beschreibung des Simulators M_{V^*} bei Iteration j:

- **S1:**
  $\beta \in_R \{0,1\}$
  $\psi \in_R \mathrm{Perm}(N)$
  $H := \psi(G_\beta)$
- **S2:**
  $i = V^*(x, (H_1, \psi_1), \ldots, (H_{j-1}, \psi_{j-1}), H)$
  - Falls $i = \beta$, dann schreibe (H, i, ψ) auf das Ausgabeband und setze die Simulation mit der Runde j + 1 fort.
    $j := j + 1$
  - Falls $i \neq \beta$ dann wiederhole diese Iteration nochmal.

Der Simulator versucht in Stufe (S1) den Isomorphismus, den V^* sehen will, durch das Bit β zu erraten. Entsprechend berechnet er H so, daß er V^* einen gültigen Isomorphismus zeigen kann. In Stufe (S2) benutzt er den Verifier V^* als Unterprogramm. Er gibt ihm x und $(H_1, \psi_1), \ldots, (H_{j-1}, \psi_{j-1}), H)$ als Eingabe und bekommt ein Bit i als Ausgabe. Sollte der bösartige Verifier V^* kein Bit $\in \{0,1\}$ liefern, so wird dies behandelt, als hätte er 0 geantwortet.

Nun testet der Simulator, ob er die Herausforderung von V^* richtig erraten hat. Falls ja, dann kann er eine gültige Permutation ψ vorzeigen. Damit hängt er das Tripel (H, i, ψ) an die j-1 Tripel auf seinem Arbeitsband an und geht in die nächste Iteration.

War allerdings $i \neq \beta$, so hat er den Wunsch von V^* falsch vorausgesagt und kann somit keinen gültigen Isomorphismus vorweisen. Deshalb muß er die Iteration wiederholen.

Nach der Simulation der n-ten Iteration gibt der Simulator die Mitschrift $(x, \overline{r}, \gamma)$ aus. Mit γ bezeichnen wir die Folge $(H_1, \psi_1), \ldots, (H_n, \psi_n)$.

Jetzt ist noch zu zeigen, daß der angegebene Simulator die geforderten Eigenschaften hat.

Behauptung1: Der Simulator ist im Erwartungswert polynomial zeitbeschränkt.

Beweis :
In jeder Wiederholung einer Iteration z.B. j + 1, sind β und H stochastisch unabhängig voneinander. Denn unabhängig vom Wert β ist H immer eine zufällige isomorphe Kopie von G_1[3]. Sei $\gamma_j = (H_j, \psi_j)$ und $\gamma^{(j)}$ bezeichne $gamma_1, \ldots, \gamma_j$.

Dann gilt:

$$\text{Prob}[V^*(x, \gamma^{(j)}, H) = \beta] \;=\; \frac{1}{2}$$

Das heißt, der Erwartungswert der Anzahl der Wiederholungen pro Runde ist 2 und da auch V^* nach Voraussetzung polynomial zeitbeschränkt ist, folgt daraus, daß M_{V^*} im Erwartungswert polynomial zeitbeschränkt ist. □

Behauptung2: $\{view^P_{V^*}(x)\}_{x \in GI}$ und $\{M_{V^*}(x)\}_{x \in GI}$ sind identisch.

Beweis :
Eine Mitschrift der Interaktion in diesem Protokoll enthält folgendes:

- Die Graphen G_0 und G_1.
- Alle ausgetauschten Nachrichten.
- Die von V^* benutzten Zufallszahlen.

Also hat eine Mitschrift T des obigen Protokolls folgende Form:

$$T = ((G_0, G_1); (H_1, i_1, \rho_1); \ldots; (H_n, i_n, \rho_n)).$$

Ein typischer Wert der Zufallsvariablen $M_{V^*}(x)$ und $view^P_{V^*}(x)$ besteht aus der Eingabe x und einem Zufallsbitstring $\overline{r}$, sowie einer Folge von n Tripeln $(H_1, i_1, \rho_1), \ldots, (H_n, i_n, \rho_n)$.

[3]Gilt im positiven Fall, d.h. falls beide Eingangsgraphen isomorph sind und nur in diesem Fall muß der Simulator, nach Definition, das Gewünschte liefern.

Beide Verteilungen hängen von $\overline{r}$ ab, da $\overline{r}$ in beiden Fällen jedoch unter Gleichverteilung ausgewählt wird, reicht es zu zeigen, daß unter einem festgehaltenen $\overline{r}$ die bedingten Verteilungen gleich sind. Dann nehmen die Zufallsvariablen als Werte eine Mitschrift T an.

Wir bezeichnen mit $view^P_{V^*}(x)^j$ und $M_{V^*}(x)^j$ die Wahrscheinlichkeitsverteilungen, die durch die ersten j Iterationen definiert sind.

Durch Induktion wird gezeigt, daß die obigen Wahrscheinlichkeitsverteilungen für jede Iteration identisch sind.

IA: $j = 0$:

Die Mitschrift enthält nur die Eingabegraphen. Die Verteilung ist also gleich.

IH: Für ein $j \geq 1$

Die Verteilungen $view^P_{V^*}(x)^{j-1}$ und $M_{V^*}(x)^{j-1}$ sind identisch.

IS: $j - 1 \rightarrow j$

Was passiert in Runde j?
Wir wissen nur, daß der Verifier ein i_j auswählt. Da er es nicht gleichverteilt auswählen muß, sondern möglicherweise in irgendeiner Abhängigkeit zu früheren Iterationen, können wir dem Ereignis $i_j = 1$ nur eine Wahrscheinlichkeit p_1 zuordnen. Damit ist die Wahrscheinlichkeit für $i_j = 0$ gerade $1 - p_1$. Die Wahrscheinlichkeit p_1 hängt von dem Zustand von V^* zu Beginn der j-ten Iteration ab.

Der Prover wählt zufällig einen einen isomorphen Graph H aus. Damit ist auch die Permutation ρ zufällig gleichverteilt ausgewählt. Jede Permutation ist gleichwahrscheinlich für jede mögliche Wahl von i_j und damit unabhängig von p_1. Daraus folgt für die Wahrscheinlichkeit das (H, i, ρ) das j-te Tripel der Mitschrift ist, für

$$i_j = 1 \quad : \quad \frac{p_1}{n!}$$

und für

$$i_j = 0 \quad : \quad \frac{1 - p_1}{n!}$$

Wie ist der Fall bei der Simulation? Der Simulator wählt jeden Graphen gleichverteilt mit Wahrscheinlichkeit $\frac{1}{n!}$ aus. Die Wahrscheinlichkeit, daß $i'_j = 1$ und V^* mit 1 antwortet ist $\frac{p_1}{2}$. Entsprechend ist die Wahrscheinlichkeit, daß i'_j $= 0$ und V^* mit 0 antwortet $\frac{1-p_1}{2}$. In beiden Fällen wird (H, i, ρ) zum j-ten Tripel der simulierten Mitschrift. Mit Wahrscheinlichkeit $\frac{1}{2}$ wird kein Tripel angehängt.

Fall $i'_j = 1$: Mit Wahrscheinlichkeit p_1 wählt V^* „1“ aus. Damit beträgt die Wahrscheinlichkeit, daß ein Tripel $(H, 1, \rho)$ als j-tes Tripel an die Mitschrift angehängt wird in dem k-ten Durchlauf der Iteration:

$$\frac{p_1}{2^k \cdot n!}$$

Addiert man über alle möglichen Werte von k auf, so erhält man die Wahrscheinlichkeit, daß $(H, 1, \rho)$ das j-te Tripel der Mitschrift ist.

$$\frac{p_1}{2 \cdot n!}\left(1 + \frac{1}{2} + \frac{1}{4} + \ldots\right) = \frac{p_1}{n!}$$

Fall $i'_j = 0$: Die gleiche Analyse liefert für den Fall $i'_j = 0$: Die Wahrscheinlichkeit, daß $(H, 0, \rho)$ als das j-te Tripel der Mitschrift angehängt wird, ist $\frac{1-p_1}{n!}$

Daraus folgt, daß die Verteilungen am Ende der j-ten Iteration gleich sind. Durch die Induktion gilt: Die Verteilungen sind nach n Iterationen identisch.

□

Aus den beiden Behauptungen folgt der Satz. □

5.3 Ein PZK-Beweissystem für GNI

Das im Kapitel drei beschriebene Interaktive Beweissystem hat wahrscheinlich nicht die Zero-Knowledge Eigenschaft. Ein betrügerischer Verifier könnte einen Graphen H schicken, von dem er nicht weiß, ob er isomorph zu G_0 oder G_1 ist. Durch das Interaktive Protokoll würde er es aber erfahren, so daß in diesem Fall das Beweissystem intuitiv nicht die Zero-Knowledge Eigenschaft hat. Es ist unwahrscheinlich, daß ein Simulator eine solche Mitschrift der Interaktion nachmachen kann.

Falls der Verifier aber immer weiß, zu welchem Eingangsgraph sein Herausforderungsgraph H isomorph ist[4], so lernt er nichts durch die Antwort des Provers.

Die Idee ist nun folgende, der Verifier muß erst gegenüber dem Prover beweisen, daß er einen Isomorphismus zwischen H und einem der Eingangsgraphen kennt. Der Prover antwortet nur, wenn er von dem Verifier davon überzeugt werden konnte. Dieser Beweis des Verifiers gegenüber dem Prover wird als Unterprotokoll in das frühere Interaktive Beweissystem für GNI (siehe Seite 32) eingebaut. Das Unterprotokoll ist eine parallele Version des ZK-Protokolls für GI aus dem letzten Abschnitt. Es darf den Isomorphismus nicht verraten, aber es muß nicht die Zero-Knowledge Eigenschaft haben.

[4]Das ist z.B. beim ehrlichen Verifier immer der Fall.

5.3.1 PZK-Protokoll für GNI

- Eingabe: Zwei Graphen G_0 und G_1 jeweils mit Knotenmenge $N = \{1, \ldots, n\}$
  Die folgenden 5 Schritte werden n mal hintereinander ausgeführt, wobei jedesmal neue unabhängige Zufallsbits verwendet werden.

1. **V1:**
   Der Verifier wählt zufällig ein Bit $\alpha \in_R \{0,1\}$ und eine Permutation $\pi \in_R Perm(N)$.
   Mit diesen Werten konstruiert er den Graphen $H := \pi(G_\alpha)$. Damit ist der Graph H so konstruiert worden, daß er eine zufällige isomorphe Kopie von G_1 oder G_0 ist. Zusätzlich konstruiert der Verifier n^2 Paare von Graphen. Jedes Paar besteht aus einer zufälligen isomorphen Kopie des Graphen G_0 und einer zufälligen isomorphen Kopie des Graphen G_1. Solch ein Paar heißt *Testpaar.* Die zufälligen isomorphen Kopien werden auch zufällig in dem Paar plaziert. Die Testpaare werden vom Prover benützt um zu testen, ob der Verifier den Graphen H nach den Protokoll-Spezifikationen konstruiert hat. Der Verifier geht konkret so vor:
   (a) für alle i, mit $1 \leq i \leq n^2$, wählt er ein bit $\gamma_i \in_R \{0,1\}$ und zwei Permutationen $\tau_{i,0}, \tau_{i,1} \in \text{Perm}(N)$.
   (b) für j = 1 und j = 2 berechnet er einen Testgraphen T
   $T_{i,j} = \tau_{i,j}(G_{j+\gamma_i \; mod 2})$.
   (c) Der Verifier schickt den Graphen H und die Folge von Testpaaren an den Prover.
2. **P1:**
   Der Prover wählt zufällig eine Teilmenge I von $\{1, 2, \ldots, n^2\}$ unter Gleichverteilung aus. Der Prover schickt I an den Verifier.
3. **V2:**
   Falls I keine Teilmenge von $\{1, 2, \ldots, n^2\}$ ist, dann halte und verwerfe. Sonst antworte mit
   $$\begin{cases} (\gamma_i, \tau_{i,1}, \tau_{i,0}) & \text{falls} \quad i \in I \\ (\alpha + \gamma_i \; mod \; 2, \tau_{i,(\alpha+\gamma_i mod 2)}\pi^{-1} & \text{falls} \quad i \in \overline{I} \end{cases}$$
   Für i $\in$ I zeigt der Verifier den Isomorphismus zwischen den Eingangsgraphen und dem i-ten Testpaar. Dazu schickt er die Permutationen $\tau_{i,1}, \tau_{i,0}$ sowie das zufällige Bit γ_i, um die Ordnung innnerhalb des Paares zu bestimmen. Damit beweist der Verifier gegenüber dem Prover, daß er die Testpaare richtig konstruiert hat. Für i $\in \overline{I}$ zeigt der Verifier einen Isomorphismus zwischen dem Graphen H und einem Graphen des Testpaars. D.h. unter der Vorraussetzung, daß das i-te Testpaar richtig konstruiert wurde, wurde auch H den Spezifikationen entsprechend konstruiert.
4. **P2:**
   Der Prover muß nun folgendes überprüfen: Für alle $i \in I$ und $j \in \{0,1\}$ prüfe,

ob $\tau_{i,j}$ tatsächlich ein Isomorphismus zwischen $T_{i,j}$ und $G_{(j+\gamma_i \bmod 2)}$ ist und für alle $i \in \overline{I}$ überprüfe ob $\tau_{i,(\alpha+\gamma_i mod\ 2)}\pi^{-1}$ auch ein Isomorphismus zwischen H und $T_{i,(\alpha+\gamma_i\ mod 2)}$ ist. Falls irgendeine Bedingung verletzt ist, stoppt der Prover. Sind alle Bedingungen erfüllt, so ist der Prover überzeugt davon, daß der Verifier den Herausforderungsgraphen H entsprechend den Spezifikationen des Protokolls konstruiert hat. Er antwortet deshalb mit dem Bit $\beta \in \{0,1\}$, so daß G_β isomorph zu H ist.[5]

5. **V3:**
   Der Verifier überprüft, ob $\alpha = \beta$. Falls ja, dann nimmt er weiter am Protokoll teil.
   Falls nein, dann hält er an und verwirft.

- Hat der Verifier alle n Runden durchgehalten, dann akzeptiert er.

Satz 5.2
Das Protokoll ist ein PZK-Beweissystem für GNI.

Beweis :
Teil1: Zu zeigen: Das Protokoll ist ein Interaktives Beweissystem.

- *Durchführbarkeit:*
  Sind beide Graphen nicht isomorph und folgen Prover und Verifier dem Protokoll, so akzeptiert der Verifier immer. Denn der ehrliche Verifier kann den Prover im Unterprotokoll immer überzeugen, daß er den Status des Graphen H kennt. Andererseits kann der Prover immer feststellen, zu welchem Eingangsgraphen der Graph H isomorph ist, und damit kann er immer das passende Bit β schicken welches den Verifier zum Akzeptieren bringt.

- *Korrektheit:*
  Sind die Eingangsgraphen dagegen isomorph, so sind die Graphen, die der Verifier schickt, zufällige zu beiden Eingangsgraphen isomorphe Kopien. Der Prover bekommt dadurch keine Information über das Bit α und kann nur ein passendes Bit β raten. Dadurch ist die Wahrscheinlichkeit, daß der Prover ein passendes β schickt, in einer Iteration höchstens $\frac{1}{2}$. In n Iteration ist die Wahrscheinlichkeit, daß der Verifier zwei isomorphe Graphen nicht verwirft dann höchstens $\frac{1}{2^n}$.

  Damit ist das Protokoll ein Interaktives Beweissystem.

Teil2: Zu zeigen: Das Protokoll ist PZK.

Um zu zeigen, das das Protokoll PZK ist, geben wir wieder für jeden Verifier V^* einen Simulator M_{V^*} an, der wieder

[5]Falls kein solches β existiert, antwortet er mit $\beta = 0$.

- eine im Erwartungswert polynomial zeitbeschränkte Laufzeit hat und
- dessen Ausgabe $M_{V^*}(x)$ mit der view$_{V^*}^{P}(x)$ übereinstimmt.

Der Simulator verwendet dabei V^* als Unterprogramm.

Der Simulator beginnt mit der Wahl eines Zufallsbitstrings $\overline{r} \in \{0,1\}^{q(n)}$. Das Polynom $q(\cdot)$ beschränkt die Laufzeit von V^*. Der Simulator schreibt dieses $\overline{r}$ auf das Zufallsband von V^* und beginnt mit der Simulation.

Genaue Beschreibung der Arbeitweise des Simulators in einer Iteration t:

- **S1:**
  Der Simulator benötigt vom Verifier V^* den Graphen H und die n^2 Paare $(T_{1,1}, T_{1,0}, \ldots, (T_{n^2,1}, T_{n^2,0})$. Dazu startet er den Verifier V^* mit folgenden Eingaben:
  - Die Graphen G_0, G_1.
  - Mit dem Zufallsbitstring $\overline{r}$ auf dem Zufallsband des Verifiers.
  - Und der Mitschrift T des bisherigen Protokoll. In der ersten Runde gibt es noch keine Mitschrift.

  Der Simulator simuliert nun den Prover, indem er eine Teilmenge I $\in \{1, 2, \ldots, n^2\}$ zufällig auswählt. Er schreibt die Menge I auf das Kommunikationsband mit Lesezugriff des Verifiers V^*. Der Simulator speichert die Menge I auf sein Arbeitsband.

- **S2:**
  Der Simulator M_{V^*} liest vom Kommunikationsband mit Schreibzugriff des Verifiers V^* folgende Ausgaben:

$$\begin{cases} (\gamma_i, \tau_{i,1}, \tau_{i,0}) & : \quad i \in I \\ & : \\ (\alpha + \gamma_i \; mod \; 2, \tau_i, (\alpha + \gamma_i \; mod \; 2) \circ \pi^{-1} & : \quad i \in \overline{I} \end{cases}$$

  Die Maschine M_{V^*} überprüft nun folgendes:
  - für alle $i \in I$ und $j \in \{0,1\}$ muß gelten
    $\tau_{i,j}$ ist ein Isomorphismus zwischen $T_{i,j}$ und $G_{(j+\gamma_i \; mod 2)}$
  - für alle $i \in \overline{I}$ muß gelten
    $\tau_{i,(\alpha+\gamma_i \; mod 2)} \circ \pi^{-1}$ ist ein Isomorphismus zwischen
    $T_{i,(\alpha+\gamma_i \; mod 2)}$ und H.

  Falls eine der Bedingungen nicht gilt, so gibt M_{V^*} sein Arbeitsband aus und stoppt.

- **S3:**
  In dieser Stufe versucht der Simulator einen Isomorphismus zwischen H und G_α zu finden.

- **S3.1:**
  Der Simulator wählt eine Teilmenge K von $\{1, 2, \ldots, n^2\}$ zufällig aus. Er startet sein Unterprogramm V^* mit den Eingaben wie oben nur mit dem Unterschied, daß er statt I die Menge K auf das Kommunikationsband mit Lesezugriff von V^* geschrieben hat. Nachdem V^* seine Berechnungen abgeschlossen hat, liest M_{V^*} folgendes von dem Kommunikationsband mit Schreibzugriff von V^*:

$$\begin{cases} (\delta_i, \sigma_{i,1}, \sigma_{i,0}) & : \quad i \in K \\ & : \\ (\alpha + \delta_i \ mod\ 2, \sigma_i, (\alpha + \delta_i \ mod 2) \circ \pi^{-1} & : \quad i \in \overline{K} \end{cases}$$

- **S3.2:**
  Für jedes $i \in (I \cap \overline{K})$ gilt: Ist $\sigma_{i,(\alpha+\delta_i \ mod\ 2)}\pi^{-1}$ ein Isomorphismus zwischen $T_{(\alpha+\delta_i \ mod\ 2)}$ und H, dann ist die Komposition $\tau^{-1}_{i,(\alpha+\delta_i \ mod\ 2)}\sigma_{i,(\alpha+\delta_i \ mod\ 2)}\pi^{-1}$ ein Isomorphismus zwischen dem Eingangsgraphen G_α und dem Graph H. Analog gilt dies für alle $i \in (\overline{I} \cap K)$ Falls kein solches Element i gefunden wurde gehe zu Stufe S3.1
- **S3.3:**
  Versuche parallel durch die Brute-Force Methode (Alle Möglichkeiten durchprobieren) einen Isomorphismus zwischen dem Graph H und einem der Eingangsgraphen zu finden. Sollte der Simulator an dieser Stelle feststellen, daß der Graph H zu keinem der Eingangsgraphen isomorph ist, so behandelt er diesen Fall wie wenn er einen Isomorphismus zwischen den Graphen H und G_0 gefunden hätte.

- **S4:**
  Wurde ein Isomorphismus zwischen dem Graph H und dem Graph G_β gefunden, so speichere das Bit β auf einen speziellen Bereich des Arbeitsbandes und beende diese Iteration.

Hat der Simulator M_{V^*} alle Iterationen des Protokolls beendet so gibt er sein Arbeitsband aus.

□

Die Idee, die der Konstruktion des Simulators M_{V^*} zugrundeliegt ist folgende. Falls es der Verifer schafft, durch die Prüfungen des Provers in der Stufe P2 des Protokolls mit einer Wahrscheinlichkeit von ϵ zu kommen, so schafft der Simulator es mit dieser Wahrscheinlichkeit in die Stufe S3 zu kommen.

In dieser Stufe muß der Simulator nun einen Isomorphismus zwischen dem Graphen H und einem der Eingangsgraphen präsentieren. Um einen solchen zu bekommen startet er wieder den Verifier V^*, mit demselben Zufallsband. Dies führt dazu, daß der Verifier dieselbe Frage stellt und vorallem auch die gleichen Test-Paare von Graphen schickt. Der Simulator fragt ihn nun aber über eine andere Teilmenge K aus. In der Stufe S2 wurden dem Simulator schon Isomorphismen zwischen den Test-Paar Graphen und den Eingangsgraphen geliefert. Bekommt der Simulator in dieser Stufe einen Isomorphismus zwischen dem Graphen H und einem der Test Graphen, so kann sich der Simulator durch die Komposition der beiden Isomorphismen einen Isomorphismus zwischen H und einem der Eingangsgraphen konstruieren.

Ist die Wahrscheinlichkeit ϵ größer als 2^{-n^2} so gibt es mehrere Teilmengen K mit denen der Verifier die Stufe P2 des Provers schaffen könnte und der Simulator hat dann fast die Wahrscheinlichkeit ϵ durch zufällige Wahl einer Teilmenge eine solche „gute“ Teilmenge zu bekommen. Dann ist der Erwartungswert der Anzahl der Wiederholungen der Stufe S3 in diesem Fall $O(1/\epsilon)$

Der sehr seltene Fall, daß es nur eine gute Teilmenge gibt, mit der der Verifier, ohne den geforderten Isomorphismus zu kennen, durch die Prüfung des Provers kommt wird vom Simulator in der Stufe S3.3 behandelt. Der Simulator versucht hier durch Brute-Force einen Isomorphismus zu finden. Dieser Fall tritt aber nur mit Wahrscheinlichkeit 2^{-n^2} auf.

Behauptung1:
Die Laufzeit des Simulators ist im Erwartungswert polynomial beschränkt.

Beweis :
Wir halten den Zufallsbitstring r fest und betrachten die Laufzeit in einer einzelnen Iteration. Eine Teilmenge $I \subseteq \{1, \ldots, n^2\}$ heißt *gut*, falls der Verifier für

- alle $i \in I$ einen Isomorphismus zwischen dem i-ten Test-Graphen Paar und den Eingangsgraphen angeben kann, und für
- alle $i \notin I$ einen Isomorphismus zwischen dem Graphen H und einem der Testgraphen angeben kann.

Mit g_r bezeichnen wir die Anzahl von guten Teilmengen. Es ist klar, daß g_r zwischen 0 und 2^{n^2} liegt.

Wie oft wird nun der Verifier V^* in der Iteration aufgerufen? Es gibt drei Fälle zu unterscheiden.

Fall1: $(g_r \geq 2)$
Wurde eine gute Teilmenge I in der Stufe S1 gewählt, wie groß ist dann die Wahrscheinlichkeit, daß eine zufällig ausgewählte Teilmenge $K \neq I$, gut ist?

Der Beitrag zum Erwartungswert der Aufrufe ist in diesem Fall

$$1 + \frac{g_r}{2^{n^2}} \cdot \frac{2^{n^2}}{g_r - 1} = 1 + \frac{g_r}{g_r - 1} \leq 3$$

Fall2: $(g_r = 1)$
Nur mit Wahrscheinlichkeit 2^{-n^2} ist die Teilmenge I gut. In diesem Fall wird der Verifier nur einmal aufgerufen und zusätzlich ist der Beitrag zum Erwartungswert der Laufzeit vom Simulator $n! \cdot 2^{-n^2} < 1$.

Fall3: $(g_r = 0)$
Der Simulator wird nur einmal aufgerufen.

In allen drei Fällen ist die Laufzeit des Simulators im Erwartungswert polynomial zeitbeschränkt, solange der Verifier polynomial zeitbeschränkt ist.

Behauptung2:
Die Verteilungen der Ausgabe des Simulators und der view des Verifiers sind gleich.

Beweis :
Beide Verteilungen bestehen aus einem zufälligen r und einer Folge von Elementen (I, β). Entweder ist I gut, dann ist in beiden Fällen β so, daß H isomorph zu G_β ist, oder I ist irgendeine nicht gute Teilmenge von $\{1, \ldots, n^2\}$ $\square$

Aus beiden Behauptungen folgt, daß das Protokoll PZK ist. $\square$

Kapitel 6

Alle Sprachen in NP haben einen Zero-Knowledge Beweis

6.1 Einleitung

Als im Jahr 1985 die Zero-Knowledge Beweise von Goldwasser, Micali und Rackoff [GMR85] entdeckt wurden, waren zunächst nur sehr wenige Beispiele von Zero-Knowledge Beweisen bekannt.

Die ersten Beispiele waren Zero-Knowledge Beweise für zwei Probleme aus der Zahlentheorie, für die Sprache Qadratischer Rest QR_N und für die Sprache Quadratischer Nichtrest QNR_n (siehe dazu auch [GMR89]).

Man vermutete damals, daß es – ebenso wie bei den Public-Key-Kryptosystemen – nur sehr wenige und zudem zahlentheoretische Probleme gebe, die sich zur Konstruktion von Zero-Knowledge Beweisen eignen.

Doch dann kam 1986 das überraschende Resultat von Goldreich, Micali, Wigderson [GMW91]. Sie zeigten, daß es unter der allgemein akzeptierten Annahme – es gibt sichere Verschlüsselungsfunktionen – für alle Sprachen in NP, und damit für eine riesige Menge von Problemen, Zero-Knowledge Beweise gibt.

Konkret zeigten sie, daß unter obiger Annahme das NP-vollständige Problem der 3-Färbbarkeit von Graphen einen Zero-Knowledge Beweis hat. Da sich jedes Problem in NP auf das NP-vollständige Problem polynomiell reduzieren läßt, haben sie damit auch gezeigt, daß jedes Problem in NP einen Zero-Knowledge Beweis besitzt. Kurz darauf hat M. Blum ein Zero-Knowledge Protokoll, für das NP-vollständige Problem, einen hamiltonschen Kreis in einem Graphen zu finden, (siehe [Blu86]) angegeben. Allerdings gibt er keinen formalen Beweis dafür an, daß sein interaktives Beweissystem tatsächlich die Zero-Knowledge Eigenschaft besitzt. In diesem Kapitel wollen wir das Protokoll von Blum nachvollziehen und einen formalen Beweis führen, daß dieses interaktive Beweissystem tatsächlich die Zero-Knowledge Eigenschaft hat. Damit haben wir auch gezeigt, daß alle Probleme in NP einen Zero-Knowledge Beweis haben.

6.1.1 Zusammenhang von Sprache und Beweis

Die Klasse NP besteht aus Sprachen L, die von einer nichtdeterministischen Turingmaschine in Polynomialzeit akzeptiert werden (siehe Grundlagenkapitel).

In einem Zero-Knowledge Beweis wird (wie in jedem Beweis) eine Behauptung z.B. ein Satz bewiesen. Was heißt es nun, wenn eine Sprache $L \in$NP einen Zero-Knowledge Beweis hat? Welcher Satz ist da zu beweisen bzw. wie ist der Zusammenhang von einem Satz, der bewiesen wird, und der Sprache $L \in$NP?

Nun, jeder Sprache L $\in$NP läßt sich ein Entscheidungsproblem zuordnen. Für eine Eingabe x ist das Entscheidungsproblem für L die Frage, ob x in L enthalten ist. Als Beispiel betrachten wir die Sprache SAT, welche aus allen erfüllbaren aussagenlogischen Boolsch'en Formeln besteht. Ein Entscheidungsproblem für SAT ist:

Gegeben: Eine aussagenlogische Formel x.
Frage: Ist x eine erfüllbare Formel?

Für jede Formel x, die tatsächlich in SAT enthalten ist, läßt sich ein Satz wie folgt formulieren:

Die Formel x ist eine erfüllbare Formel (d.h. Element von SAT).

Ein Beweis dieses Satzes ist eine erfüllende Belegung für x. Allgemein kann das Entscheidungsproblem für alle positiven Instanzen x $\in$ L, für alle L $\in$ NP, in einen entsprechenden Satz wie

„Die Eingabe x ist Element von L“

umformuliert werden. Damit haben wir für jede Sprache L $\in$ NP eine Menge von Sätzen die bewiesen werden können.

Hat eine Sprache L $\in$ NP ein Zero-Knowledge Beweissystem, so bedeutet dies, daß sich mit dem Zero-Knowledge Beweissystem die Menge der Sätze, welche durch die positiven Instanzen gegeben sind, beweisen lassen.

Das in diesem Kapitel vorgestellt Resultat ist deshalb so bedeutend, weil sich fast alle in der Praxis interessanten Probleme als Sprachen in NP formulieren lassen und damit einen Zero-Knowledge Beweis haben.

6.2 Commitment Schema

Für den Beweis, daß alle Sprachen in der Klasse NP einen CZK-Beweis haben, benötigen wir ein Bit Commitment Schema.

Bei einem Commitment geht es darum, daß sich eine Partei A gegenüber anderen Parteien auf einen Wert m festlegt. Partei A möchte sich dabei so festlegen, daß der Wert m zunächst geheim bleibt.

Ein Beispiel aus dem Alltag soll den Nutzen eines solchen Commitments verdeutlichen. Das Beispiel stammt aus [BSW95].

Auf eine Ausschreibung hin reichen mehrere Firmen ein Angebot ein. Jedes Angebot einer Firma muß einen Kostenvoranschlag enthalten. Dieser Kostenvoranschlag soll aber den anderen Firmen nicht bekannt werden, damit diese nicht ihren Preis entsprechend anpassen können.

Die Firmen stehen jetzt vor dem Problem ihren Kostenvoranschlag geheimzuhalten und sich trotzdem auf einen Wert festzulegen, den sie später nicht mehr ändern dürfen. Eine Lösungsmöglichkeit wäre, daß jede Firma den Preis für ihr Angebot in einen Tresor hinterlegt. Dieser wird verschlossen und der ausschreibenden Stelle übergeben, während der Schlüssel des Tresors bei der Firma bleibt. Damit haben sich die Firmen gegenüber der ausschreibenden Stelle so festgelegt, daß die Festlegung geheim ist (also von den anderen Firmen nicht lesbar), und nicht mehr verändert werden kann. Sobald die Frist der Ausschreibung vorbei ist, übersenden alle Firmen ihre Schlüssel an die ausschreibende Stelle.

Prinzipiell kann diese Lösungsmöglichkeit auf den Kontext eines kryptographischen Protokolls übertragen werden.

Eine kollisionsfreie Hashfunktion f übernimmt hier die Funktion des Tresors. Diese Funktion ist öffentlich bekannt. Eine Partei A legt sich nun gegenüber einer Partei B auf den Wert m fest, indem sie Partei B den Funktionswert $c = f(m)$ sendet. Zu einem späteren Zeitpunkt kann A die Festlegung öffnen, indem sie B den Wert m zuschickt. Partei B kann nun den Funktionswert $f(m)$ mit der Festlegung von Partei A vergleichen.

Durch die Einwegeigenschaft der Funktion f ist die Festlegung c von Partei A geheim. Andererseits kann Partei A, aufgrund der Kollisionsfreiheit der Funktion f, später kein m' schicken, das die gleiche Festlegung c wie m besitzt.

6.2.1 Bit-Commitment

Ein besonderer Fall ist die Festlegung auf nur ein Bit $b \in \{0, 1\}$. Partei B könnte hier, falls analog wie oben eine Festlegung c erzeugt wurde diese selbst öffnen, indem er einfach die beiden Werte $f(0)$ und $f(1)$ berechnet und mit der Festlegung c von Partei A vergleicht. Die Festlegung c von A ist nicht mehr geheim.

Eine Lösungsmöglichkeit ist, daß sich Partei A nicht nur auf ein Bit b, sondern zusätzlich noch auf eine ausreichend große Zufallszahl r festlegt. Dann wird der Funktionswert $c = f(b, r)$ als Festlegung veröffentlicht. Die Kollisionsfreiheit der Funktion f muß nur für das erste Argument gelten.

Ein Kandidat für die Realisierung der Funktion f ist z.B.

$$f(b,r) \ := \ y^b \cdot r^2 \quad mod\, n,$$

dabei ist n eine Blum-Integer. Die Zahl y ist ein festes Element der Menge QNR_n. Wir machen folgende Beobachtung: $f(0,r) = r^2$ ist immer Element der Menge QR_n während $f(1,r) = y \cdot r^2$ immer Element der Menge QNR_n ist. Die Mengen QR_n und QNR_n sind zueinander disjunkt und damit folgt die Kollisionsfreiheit der Funktion f bezüglich der ersten Komponente.

Definition 6.1 (Commitment Schema)
Ein **Commitment Schema** *ist ein effizientes Protokoll zwischen zwei Parteien. Dieses Protokoll kann in zwei Phasen aufgeteilt werden. Durch das Protokoll legt sich die eine Partei – wir bezeichnen sie als Sender – gegenüber der anderen Partei – diese bezeichnen wir als Empfänger – auf einen Wert m fest, so daß folgende zwei Bedingungen erfüllt sind.*

1. **Geheimhaltung:**
   Am Ende der ersten Phase hat sich der Sender gegenüber dem Empfänger auf einen Wert festgelegt, so daß der Empfänger nichts über den festgelegten Wert weis . Dies gilt selbst dann, wenn der Empfänger versucht, zu betrügen.

2. **Eindeutigkeit:**
   Es darf nur einen Wert geben, den der Empfänger in der zweiten Phase als Öffnung der Festlegung akzeptiert. Dann heißt die Festlegung eindeutig.

Die erste Phase, in der der Sender sich auf einen Wert festlegt, heißt Festlegungsphase. *Die zweite Phase, in der die Festlegung geöffnet wird, heißt* Öffnungsphase.

Das Protokoll ist effizient, das heißt, daß die Programme für Sender und Empfänger auf probabilistischen polynomial zeitbeschränkten Maschinen implementiert werden können. Die Öffnungsphase besteht o.B.d.A. nur darin, daß der Sender dem Empfänger den Wert auf den er sich festgelegt hat, sowie die Zufallsbits, die er zur Festlegung verwendet hat, sendet.

Definition 6.2 (Bit-Commitment Schema)
Ein **Bit Commitment Schema** *ist ein Paar (S, E) von polynomial zeitbeschränkten ITMs, so daß folgendes gilt:*

- *Die gemeinsame Eingabe für die ITMs ist eine natürliche Zahl n in unärer Kodierung. Diese Zahl heißt* **Sicherheitsparameter** *des Schemas. Der Sender bekommt als Zusatzeingabe ein Bit b.*

- **Geheimhaltung:**
  Der Empfänger darf am Ende der Festlegungsphase der Festlegung nicht ansehen, auf welches Bit b sich der Sender festgelegt hat. Diese Bedingung muß auch gelten, falls der Empfänger vom Protokoll abweicht. Das heißt, für jede polynomial zeitbeschränkte ITM E^* sind die Zufallsvariablen $\langle S(0), E^*\rangle(1^n)$ und $\langle S(1), E^*\rangle(1^n)$ nicht effizient unterscheidbar.

- **Eindeutigkeit:**

  - *Die view des Empfängers nimmt Werte der Form $(r, \overline{m})$ an. Dabei bezeichnet r die Zufallsbits des Empfängers und $\overline{m}$ ist die Sequenz von Nachrichten, die der Empfänger vom Sender bekommt.*
  - *Sei $\rho \in \{0,1\}$. Die view $(r, \overline{m})$ des Empfängers heißt eine mögliche Festlegung von ρ, falls es einen String s gibt, so daß der Sender bei Eingabe von $(\rho, 1^n)$ und Benutzung der Zufallsbits s die Sequenz $\overline{m}$ von Nachrichten erzeugen kann.*
  - *Ist die view $(r, \overline{m})$ des Empfängers sowohl eine mögliche Festlegung von 0 wie auch von 1, so heißt die view mehrdeutig.*

  Die view $(r, \overline{m})$ des Empfängers heißt **eindeutig**, *falls es für alle Zufallsbits $r \in \{0,1\}^{poly(n)}$ bis auf einen vernachlässigbaren Anteil davon, keine Sequenz $\overline{m}$ gibt, so daß die view $(r, \overline{m})$ mehrdeutig ist. Das heißt es gibt keinen String s, so daß, der Sender für beide Eingaben $(0, 1^n)$ und $(1, 1^n)$ die gleiche Sequenz $\overline{m}$ erzeugen kann, wenn er die Zufallsbits s benützt.*

Für den Rest dieses Kapitels benötigen wir ein Bit-Commitment Schema dessen Geheimhaltungsbedingung auch in bezug auf nichtuniforme polynomial große Schaltkreisfamilien gilt. Solche Bit-Commitment Schemas existieren, basierend auf der Annahme, daß es nichtuniforme Einwegfunktionen gibt. Mit der Hilfe dieser Einwegfunktionen lassen sich die nichtuniformen Bit-Commitment Schemas genauso konstruieren wie die uniformen Schemas.

Beispiele für die Realisierung finden sich bei [Gol95, Seite 178]. Wir wollen uns im weiteren auf keine spezielle Realisierung festlegen.

Die Schreibweise $c_i = C_{s_i}(b_i)$ bedeutet dabei, daß eine Festlegung c_i auf das Bit b_i durch das (nichtuniforme) Bit-Commitment Schema C berechnet wird. Dabei werden die Zufallsbits $s_i \in \{0,1\}^n$ benützt.

6.3 CZK-Protokoll für Hamiltonkreis

In diesem Abschnitt geben wir ein CZK-Protokoll für die Sprache HAM an.

Definition 6.3 (Hamiltonkreis)
Gegeben ist ein Graph $G = (N, K)$ mit n Knoten. Ein **Hamiltonkreis** *des Graphen G ist ein geschlossener Weg, entlang der Kanten von G, der jeden der n Knoten genau einmal passiert. Ein Graph heißt* **hamiltonsch**, *wenn er einen Hamiltonkreis besitzt. Die Sprache* HAM *ist wie folgt definiert:* HAM $:= \{G \mid G$ *besitzt einen Hamiltonkreis* $\}$

Gegeben ist ein Graph $G = (N, K)$. Das Problem zu entscheiden, ob der Graph G Element der Menge HAM ist, nennen wir das *Hamiltonkreis Problem.*

Satz 6.1 (ohne Beweis)
Das Hamiltonkreis Problem ist NP-vollständig.

Einen Beweis findet man in [GJ79, Seite 56 ff].

6.3.1 Das Protokoll

Die Idee des Beweisverfahrens.

Für einen hamiltonschen Graph G besitzt der Prover einen Beweis. Dieser Beweis besteht aus einem Hamiltonkreis für diesen Graphen. Der Hamiltonkreis ist dabei durch eine Folge von Knoten dargestellt. Damit der Prover den Verifier in Zero-Knowledge Manier überzeugen kann, teilt er den Beweis in mehrere Teile auf. Dabei muß er so vorgehen, daß

1. der Verifier für jeden Teil des Beweises überprüfen kann, ob dieser korrekt ist. Dabei muß klar sein, daß falls alle Teile korrekt sind, dann gilt der Beweis.

2. er kein Wissen über den Beweis verrät, falls er einen Teil des Beweises dem Verifier vorzeigt.

Der Prover verschließt dazu jeden Teil des Beweises in einen eigenen Tresor und übergibt alle Tresore dem Verifier. Der darf sich einen beliebigen Tresor aussuchen, dessen Inhalt er sehen möchte. Der Prover muß ihm diesen Tresor aufschließen. Alle anderen Tresore bleiben verschlossen. Der Verifier überprüft den Inhalt des Tresors. Ist dieser korrekt, dann akzeptiert er, sonst verwirft er. Um die Fehlerwahrscheinlichkeit zu senken kann dieses Protokoll mehrfach ausgeführt werden, wobei der Verifier insgesamt akzeptiert, wenn er bei jeder Ausführung akzeptiert hat. Es ist dabei wichtig, daß der Prover bei jeder Ausführung seinen Beweis wieder so in mehrere Teile zerlegt, daß der Verifier überhaupt kein Wissen aus den verschiedenen Ausführungen des Protokolls ansammeln kann.

In dem CZK-Beweissystem für HAM teilt der Prover seinen Beweis in zwei Teile auf. Die Funktion der Tresore übernimmt ein Bit-Commitment Schema.

- Die gemeinsame Eingabe des Protokolls ist ein Graph $G = (N, K)$.
  Hat dieser Graph G einen Hamiltonkreis, so bezeichnen wir diesen mit $\hat{\alpha}$. Der Prover bekommt als Zusatzeingabe eine Permutation ψ über der Menge N. Diese Permutation ψ gibt eine Reihenfolge der Knoten an, so daß diese den Hamiltonkreis $\hat{\alpha}$ im Graph G bilden.

  1. **P1:**
     Der Prover wählt zufällig unter Gleichverteilung eine Permutation π aus der Menge Perm(N). Diese Permutation wendet er auf den Eingangsgraphen G an und erzeugt dadurch einen zu G zufällig isomorphen Graphen $G' = (N, K')$. In der Adjazenzmatrix des Graphen G' steht eine 1 bei Element $e_{i,j}$, falls in der Menge K' eine Kante von Knoten i nach Knoten j enthalten ist. Falls $(i,j) \notin K'$, so steht eine 0 bei $e_{i,j}$ in der Adjazenzmatrix. Der Prover verschlüsselt nun jeden Eintrag der $n \times n$ Adjazenzmatrix einzeln mit einem Bit-Commitment Schema. Dazu wählt er zunächst zufällig und unabhängig voneinander für alle $(i,j) \in N^2$ ein $s_{i,j} \in \{0,1\}^n$. Diese $s_{i,j}$ sind die Zufallsbits, die er zum Festlegen der $e_{i,j}$ benötigt. Er berechnet für alle $(i,j) \in N^2$ die Festlegung $c_{i,j} = C_{s_{i,j}}(e_{i,j})$. Der Prover schickt alle n^2 $c_{i,j}$'s an den Verifier.
  2. **V1:**
     Der Verifier wählt zufällig einen Wert $b \in \{0,1\}$. Er schickt b an den Prover.
  3. **P2:**
     Sei σ die Nachricht, die der Prover vom Verifier bekommen hat.

     - Falls $\sigma = 1$, dann schickt der Prover an den Verifier alle $s_{i,j}$ und $e_{i,j}$ für $(i,j) \in N^2$, sowie die Permutation π.

     - Falls $\sigma = 0$, dann öffnet der Prover nur n Festlegungen, indem er $(s_{\pi(1),\pi(\psi(1))}, \pi(1), \pi(\psi(1))), \ldots, (s_{\pi(n),\pi(\psi(n))}, \pi(n), \pi(\psi(n)))$ an den Verifier schickt.
     - Der Fall $\sigma \neq \{0,1\}$ wird wie der Fall $\sigma = 0$ vom Prover behandelt.
  4. **V2:**
     Der Verifier überprüft folgendes:
     - Falls $b = 1$, dann vergleiche für alle $(i,j) \in N^2$, ob $C_{s_{i,j}}(e_{i,j})$ mit den $c_{i,j}$, auf die sich der Prover eine Runde vorher festgelegt hat, identisch sind. Zusätzlich prüft der Verifier, daß die Permutation π wirklich G in G' überführt, also ob $G' \stackrel{?}{=} \pi(G)$ ist.
     - Falls $b = 0$, dann prüfe, daß die geöffneten n Einträge der Adjazenzmatrix einen geschlossenen Weg bilden, auf den jeder Knoten genau einmal vorkommt.

- Akzeptiere, falls die jeweilige Überprüfung positiv ausfällt.
  Verwerfe, falls eine Überprüfung negativ ausfällt.

Bemerkungen zum Protokoll:

Wir bezeichnen das Programm des obigen Provers mit P_{HAM}.

Zu P1:

> Der Prover permutiert zufällig den Eingangsgraph G. Das hat den Zweck, daß der Prover dem Verifier einen Hamiltonkreis (im permutierten Graph G') aufdecken kann, der nichts über den Hamiltonkreis des Eingangsgraphen G verrät.

Zu V1:

> Der Verifier darf sich aussuchen, ob er sich vom Prover
>
> - die korrekte Konstruktion der $c_{i,j}$ oder
> - einen Hamiltonkreis aus den $c_{i,j}$
>
> zeigen lassen will.
>
> Dabei ist klar, falls der Prover beides zeigen kann, dann muß der Graph G einen Hamiltonkreis haben. Das heißt, falls der Graph G keinen Hamiltonkreis besitzt, so kann kein Prover beides zeigen.

Das Programm für den Verifier, und sogar das Programm für den Prover läßt sich in probabilistischer Polynomialzeit implementieren. Allerdings wird dies im Fall des Provers nur durch eine passende Zusatzeingabe ermöglicht. Diese Eigenschaft ist entscheidend für die Anwendbarkeit in kryptographischen Protokollen.

Behauptung: Das Protokoll ist ein Interaktives Beweissystem.

Beweis :
Hat der Graph G einen Hamiltonkreis so kann der Prover immer beides aufdecken. Er kann also jede Herausforderung der Verifiers erfüllen und somit akzeptiert der Verifier mit Wahrscheinlichkeit 1, wenn beide dem Protokoll folgen.

Besitzt der Graph G keinen Hamiltonkreis, so gibt es keinen Prover, der beide Herausforderungen des Verifiers erfüllen kann. Das heißt, egal wie der Prover vom Protokoll abweicht, er kann nur eine der beiden Herausforderungen erfüllen und somit ist die Wahrscheinlichkeit, daß der Verifier akzeptiert höchstens $\frac{1}{2}$.

Indem das Protokoll polynomial mal sequentiell wiederholt wird und der Verifier nur dann akzeptiert, wenn er in keiner Iteration des Protokolls verworfen hat, läßt sich die Fehlerwahrscheinlichkeit auf einen exponentiell kleinen Wert senken.

□

Die Sprache HAM besitzt also ein Interaktives Beweissystem. Das ist offensichtlich, da die Sprache HAM in NP liegt und die Klasse NP eine Teilmenge von IP ist. Interessanter ist hier, daß das obige Protokoll CZK mit Zusatzeingabe ist. Deswegen kann aufgrund des „Sequential Composition Lemma“ das Basisprotokoll polynomial oft, unter Beibehaltung der Zero-Knowledge Eigenschaft wiederholt werden.

Behauptung: Das obige Protokoll ist CZK.

Wir beweisen die Behauptung, indem wir einen Simulator M_{V^*} angeben, der den Verifier V^* als Unterprogramm verwenden kann und für den gilt:

- Die Laufzeit des Simulators ist im Erwartungswert polynomiell beschränkt.
- Die Ausgabe des Simulators ist von der view des Verifiers V^* nicht effizient unterscheidbar.

Der Simulator $\mathbf{M_{V^*}}$ für obiges Protokoll

- **S0:**
  Sei $q(\cdot)$ ein Polynom, das die Laufzeit des Verifiers V^* begrenzt. Der Simulator M_{V^*} wählt zufällig unter Gleichverteilung ein $r \in \{0,1\}^{q(n)}$ und schreibt dieses r auf das Zufallsband des Verifiers V^*.

- **S1:**
  Der Simulator wählt ein $b' \in_R \{0,1\}$. Er versucht damit, die Frage des Verifiers V^* zu raten.

  - Falls $b' = 0$, so wählt der Simulator eine zufällige Permutation ϕ aus Perm(N). Mit der Permutation ϕ berechnet er eine zufällige isomorphe Kopie G'' des Eingangsgraphen, also $G'' = \phi(G)$. Dann wählt er noch für alle $(i,j) \in N^2$ ein $s_{i,j} \in_R \{0,1\}^n$. Mit dem Bit-Commitment Schema C und den Werten $s_{i,j}$ verschlüsselt er die Adjazenzmatrix des Graphen G''. Er berechnet also für alle $(i,j) \in N^2$ ein
  $$d_{i,j} = C_{s_{i,j}}(e_{i,j}).$$
  Die $e_{i,j}$ bezeichnen die Elemente der Adjazenzmatrix.
  - Falls $b' = 1$, dann konstruiert der Simulator einen zufälligen Hamiltonkreis α. Die Permutation τ gibt eine Reihenfolge der Knoten entlang des Hamiltonkreises an. Er verknüpft den Hamiltonkreis mit einer „oder“ Verknüpfung in die Adjazenzmatrix des Eingangsgraphen. Der Simulator wählt wieder für alle $(i,j) \in N^2$ ein $s_{i,j} \in \{0,1\}^n$ und berechnet
  $$d'_{i,j} \;=\; C_{s_{i,j}}(e_{i,j}(\alpha) \vee e_{i,j}(G))$$

- **S2:** Falls $b' = 0$, dann schreibe $d_{i,j}$ auf das Kommunikationsband des Verifiers V^*. falls $b' = 1$, dann schreibe $d'_{i,j}$ auf das Kommunikationsband des Verifiers V^*. Starte den Verifier V^*. Der Verifier liefert als Antwort ein Bit b. Vergleiche b' mit b.

  - Falls $b' \neq b$, dann beginne die Simulation von vorne.

- Sonst
  - Ist $b = b' = 0$ so gebe folgendes aus $G, r, (s_{1,1}, e_{1,1}), \ldots, (s_{n,n}, e_{n,n}), \phi$.
  - Ist $b = b' = 1$ so gebe $G, r, (s_{\pi(1),\pi(\tau(1))}, \pi(1), \pi(\tau(1)), \ldots, (s_{\pi(n),\pi(\tau(n))}, \pi(n), \pi(\tau(n))$ aus.

Hätten wir ein Protokoll mit wirklich undurchsichtigen Boxen, statt des Bit-Commitment Schemas benutzt, so hätte der Verifier keinen Anhaltspunkt über den Inhalt der Boxen, dies könnte also seine Wahl des Bits b nicht beeinflussen.

Bei dem Bit-Commitment Schema, das wir statt der Boxen benützen, ist die Festlegung nicht ganz unabhängig von dem Inhalt, also dem Bit, auf das sich der Prover bzw. der Simulator festgelegt hat. Der Verifier trifft seine Entscheidung des Bits b nicht ganz unabhängig von dem festgelegten Bit.

Mit $p_i(G, r, (e_{1,1}, \ldots, e_{n,n}))$ bezeichnen wir die Wahrscheinlichkeit (genommen über die $s_{i,j}$), daß der Verifier V^* mit dem Bit i antwortet, falls er folgende Eingaben bekommt:

- Den Graph G
- Die Zufallsbits r
- Die Nachricht $C_{s_{1,1}}(e_{1,1}), \ldots, C_{s_{n,n}}(e_{n,n})$

Lemma 6.1 (Fast unbewußte Auswahl)
Für jedes Polynom $t(\cdot)$, jeden ausreichend großen Graphen $G = (N, K)$, jedes $r \in \{0,1\}^{q(n)}$, jedes Bit i, und für alle zwei Sequenzen $\beta, \gamma \in \{0,1\}^{n^2}$ gilt:

$$|p_i(G, r, \beta) \; - \; p_i(G, r, \gamma)| \; \leq \; \frac{1}{t(n)}$$

Beweis :

Um dieses Lemma zu beweisen wird die nichtuniforme Geheimhaltungs-Bedingung des Bit-Commitment Schemas verwendet.

Wir nehmen das Gegenteil der Aussage des Lemmas an und leiten daraus einen Widerspruch her.
Annahme: Es gibt ein Polynom $t(\cdot)$ und eine unendliche Folge von natürlichen Zahlen, so daß für jede natürliche Zahl n in der Folge, es einen Graphen G_n, einen String $r \in \{0,1\}^{q(n)}$, ein $i_n \in \{0,1\}$ und zwei Folgen $\beta_n, \gamma_n \in \{0,1\}^{n^2}$ gibt, wobei folgendes gilt:

$$|p_{i_n}(G_n, r_n, \beta_n) \; - \; p_{i_n}(G_n, r_n, \gamma_n)| \; > \; \frac{1}{t(n)}$$

Wir haben jetzt eine Folge von zwei Wahrscheinlichkeiten, deren Unterschied mehr als vernächlässigbar groß ist. Um einen Widerspruch zu bekommen konstruieren wir uns eine nichtuniforme Schaltkreisfamilie $\{A_n\}$, die Commitments der Folge β_n von Commitments der Folge γ_n aufgrund des Unterschieds in den obigen Wahrscheinlichkeiten unterscheiden kann. Dies führt aber zum Widerspruch gegen die nichtuniforme Geheimhaltungsbedingung des Bit-Commitment Schemas. Dem Schaltkreis A_n ist die ITM V^*, der Graph G_n und r_n, i_n, β_n und γ_n einkodiert. Bei einer Eingabe y, die entweder eine Festlegung der Folge β_n oder γ_n ist, startet der Schaltkreis A_n die Maschine V^* auf der Eingabe (G_n, r_n, y) und gibt nur dann eine Eins aus, falls V^* mit i_n antwortet. Nun gilt:

$$\text{Prob}(A_n(C_{U_{n^3}}(\delta)) = 1) \;=\; p_{i_n}(G_n, r_n, \delta)$$

, wobei die Zufallsvariable U_{n^3} gleichverteilt auf $\{0,1\}^{n^3}$ ist. Das bedeutet der Schaltkreis kann Festlegungen der Folge β_n von Festlegungen der Folge γ_n unterscheiden. Durch ein Standard-Hybridargument wird der Unterschied zwischen den Folgen auf eine bestimmte Bitposition innerhalb der Folgen festgelegt. Das heißt, der Schaltkreis A_n trifft seine Entscheidung aufgrund dieses Bits, was sofort einen Widerspruch zur Geheimhaltungsbedingung eines nichtuniformen Bit-Commitment Schemas ergibt, welche besagt, daß kein (nichtuniformer) polynomial zeitbeschränkter Algorithmus einen Unterschied zwischen einer Festlegung auf das Bit 0 und einer Festlegung auf das Bit 1 erkennen kann. □

Wir wollen nun das obige Lemma benutzen, um die Wahrscheinlichkeit, daß der Simulator die Simulation wiederholen muß, nach oben abzuschätzen.

Dazu führen wir zunächst einige Bezeichnungen ein. Mit

$$p_i(G, r, \phi)$$

bezeichnen wir die bedingte Wahrscheinlichkeit, daß der Verifier V^* Bit i auswählt, falls er als Eingabe den Graph G, die Zufallsbits r und eine Verschlüsselung der Adjazenzmatrix von $\phi(G)$ bekommt, wobei der Simulator $b' = 0$ gewählt hat. Entsprechend ist

$$q_i(G, r, \alpha)$$

die Wahrscheinlichkeit, daß der Verifier V^* Bit i auswählt, bei Eingabe des Graphen G, der Zufallsbits r und der Verschlüsselung der Adjazenzmatrix eines Graphen, der aus G durch Vereinigung mit Hamiltonkreis α hervorgeht. Der Simulator hat hier also $b' = 1$ gewählt.

Der Simulator muß die Simulation wiederholen, falls er ein anderes Bit b' geraten hat als der Verifier V^* wählt, denn in diesem Fall kann er die Herausforderung des Verifiers nicht erfüllen.

Hierbei können folgende zwei Fälle auftreten:

Der Simulator wählt Bit $b' = 0$, kann also eine Permutation des Eingangsgraphen aufdecken, der Verifier wählt aber i = 1, d.h. er will einen Hamiltonkreis sehen. In unserer Notation ist die Wahrscheinlichkeit hierfür

$$p_1(G, r, \phi).$$

Die Wahrscheinlichkeit $p_1(G, r, \phi)$ hängt von G, r, ϕ ab. Die Werte G und r können als fix betrachtet werden, so daß die Wahrscheinlichkeit nur von der Verschlüsselung der Adjazenzmatrix $\phi(G)$ abhängt. Wir müssen die Wahrscheinlichkeit über alle Permutationen aufsummieren, dies sind $n!$ viele und anschließend durch $n!$ teilen.

Im anderen Fall wählt der Simulator das Bit $b' = 1$, er kann also einen Hamiltonkreis aufdecken, der Verifier wählt aber $i = 0$, d.h. er will die Permutation des Eingangsgraphen sehen. In unserer Notation ist die Wahrscheinlichkeit hierfür

$$q_0(G, r, \alpha).$$

Entsprechend summieren wir hier über alle Hamiltonkreise.

Insgesamt ergibt sich somit die Gleichung:

$$\text{Prob(M muß wiederholen)} = \frac{1}{2}\left(\frac{1}{n!}\sum_{\phi} p_1(G, r, \phi) \quad + \quad \frac{1}{(n-1)!}\sum_{\alpha} q_0(G, r, \alpha)\right)$$

Nach dem „Fast unbewußte Auswahl“ Lemma gilt:

$$p_1(G, r, \phi) \ \leq \ p_1(G, r, \mathrm{id}) + \frac{1}{t(n)}$$

sowie

$$q_0(G, r, \alpha) \ \leq \ q_0(G, r, \hat{\alpha}) + \frac{1}{t(n)}$$

Dies führt zunächst zu folgender Abschätzung:

$$\leq \frac{1}{2}\left(\frac{1}{n!}\cdot n! \left(p_1(G, r, \mathrm{id}) + \frac{1}{t(n)}\right) \quad + \quad \frac{1}{(n-1)!}\cdot (n-1)!\left(q_0(G, r, \hat{\alpha}) + \frac{1}{t(n)}\right)\right)$$

$$= \frac{1}{2}\left(p_1(G, r, \mathrm{id}) + \frac{1}{t(n)}) \quad + \quad q_0(G, r, \hat{\alpha}) + \frac{1}{t(n)})\right)$$

Mit id bezeichnen wir die identische Abbildung. Jetzt sehen wir, warum der Graph G zusammen mit dem Hamiltonkreis verknüpft wurde. Mit $\hat{\alpha}$ bezeichnen wir den Hamiltonkreis des Graphen G. Jetzt bekommt der Verifier praktisch in beiden Fällen das gleiche zu sehen, und er muß sich für ein Bit entscheiden. Das heißt $p_1(G, r, \mathrm{id}) + q_0(G, r, \hat{\alpha}) = 1$.

Dadurch ist die Wahrscheinlichkeit nach oben durch

$$\leq \frac{1}{2} + \frac{1}{t(n)}$$

abgeschätzt, für alle Polynome $t(n)$. Für $t(n)$ kann z.B. $\frac{1}{6}$ gewählt werden, dann beträgt die Wahrscheinlichkeit, daß der Simulator die Simulation wiederholen muß , $\frac{2}{3}$. D.h. mit

Wahrscheinlichkeit $\frac{1}{3}$ ist der Simulator in einer Iteration erfolgreich. Dann muß der Simulator im Erwartungswert die Simulation 3 mal wiederholen (falls für $t(n)$ ein $\frac{1}{6}$ gewählt wurde).
Somit gilt:

Der Simulator M_{V^*} ist im Erwartungswert polynomiell beschränkt.

Jetzt ist noch zu zeigen:
Das Ensemble $\{M_{V^*}(G)\}_{G \in HAM}$ ist nicht effizient von dem Ensemble $\{\mathrm{view}_{V^*}^{P_{HAM}}(G)\}_{G \in HAM}$ unterscheidbar.

Das bedeutet, daß für jeden probabilistischen polynomial zeitbeschränkten Algorithmus A, jedes Polynom $t(n)$ und alle ausreichend großen Graphen $G = (N, K)$ gilt:

$$|\mathrm{Prob}(A(M_{V^*}(G)) = 1) \;-\; \mathrm{Prob}(A(\mathrm{view}_{V^*}^{P_{HAM}}(G) = 1)| < \frac{1}{t(n)}$$

Beweis :
Zunächst führen wir neue Bezeichnungen ein. Gegeben sei ein Graph $G = (N, K)$. Mit $p_0(G)$ bezeichnen wir die Wahrscheinlichkeit, daß die Zufallsvariable $M_{V^*}(G)$ Informationen zum Öffnen einer Permutation des Eingangsgraphen G enthält. Wir wollen eine weitere Zufallsvariable $\mu_0(G)$ einführen, die für genau diesen Fall die Ausgabe von $M_{V^*}(G)$ beschreibt. Das heißt, der Verifier V^* hat bei Aufruf durch den Simulator M_{V^*} Bit $b = 0$ gewählt, was bedeutet, daß er eine Permutation des Eingangsgraphen sehen will.

Mit $p_1(G) = 1 - p_0(G)$ bezeichnen wir die Wahrscheinlichkeit, daß die Zufallsvariable M_{V^*} Informationen zum Öffnen eines Hamiltonkreises enthält. Die Zufallsvariable $\mu_1(G)$ beschreibt in diesem Fall die Ausgabe von $M_{V^*}(G)$.

Entsprechend werden die Wahrscheinlichkeiten $q_0(G)$ und $q_1(G)$ in bezug auf die Zufallsvariable $\mathrm{view}_{V^*}^{P_{HAM}}(G)$ definiert. Die Zufallsvariablen $\mu_0(G)$ und $\mu_1(G)$ beschreiben in beiden Fällen die $\mathrm{view}_{V^*}^{P_{HAM}}(G)$.

Wir nehmen nun das Gegenteil der Behauptung an – die zwei Ensembles sind doch effizient unterscheidbar.

Dann muß einer der beiden folgenden Fälle auftreten.

Fall1:
Es gibt einen bemerkbaren Unterschied im Wahrscheinlichkeits-Profil zwischen den Herausforderungen des Verifiers V^*, wenn er mit dem Prover P_{HAM} kommuniziert und den Herausforderungen, falls er vom Simulator M_{V^*} aufgerufen wird.
Formal drücken wir dies wie folgt aus:

> Es existiert ein Polynom $t(\cdot)$ und eine unendliche Folge von natürlichen Zahlen, so daß für jede natürliche Zahl n in der Folge, ein Graph G_n mit n Knoten und eine Herausforderung i_n existiert, wobei gilt:
>
> $$|p_{i_n}(G_n) - q_{i_n}(G_n)| \;>\; \frac{1}{t(n)}$$

Bemerkung:
Dieser Unterschied kann nur von den Commitments, die der Verifier V^* vom Prover bzw Simulator bekommt, herkommen.

Im Fall, daß der Verifier $i_n = 0$ wählt, machen Simulator und Prover dasselbe. Beide verschlüsseln eine beliebige isomorphe Kopie des Eingangsgraphen G. Die Betrachtung reduziert sich somit auf den Fall, daß der Verifier $i_n = 1$ wählt.

$$|p_1(G_n) - q_1(G_n)| > \frac{1}{t(n)}$$

Der Verifier kann in diesem Fall ein Commitment des Provers von einem Commitment des Simulators unterscheiden. Dies führt mit dem Standard Hybrid Argument sofort zu einem Widerspruch zur nichtuniformen Geheimhaltungsbedingung des Bit-Commitment Schemas. Wenden wir uns nun dem zweiten Fall zu...

Fall2:
Im Wahrscheinlichkeitsprofil der Herausforderungen des Verifiers V^* besteht kein bemerkenswerter Unterschied, ob er mit dem Prover P_{HAM} kommuniziert oder von dem Simulator M_{V^*} aufgerufen wird. Trotzdem soll es einen Algorithmus A geben, der die Ensembles unterscheiden kann.

Es ist hier wieder nur der Fall interessant, daß der Verifier den Hamiltonkreis sehen will, weil sich nur in diesem Fall die Handlungsweise des Provers P_{HAM} und des Simulators M_{V^*} unterscheiden.

Wir können annehmen, daß der Verifier den Hamiltonkreis mit einer mehr als vernachlässigbaren Wahrscheinlichkeit sehen will.

Formal ausgedrückt heißt das:

Es existiert ein probabilistischer, polynomial zeitbeschränkter Algorithmus A, ein Polynom $t(n)$ und eine unendliche Folge von natürlichen Zahlen, so daß für jede natürliche Zahl n in der Folge ein Graph G_n existiert, wobei gilt:

- $q_1(G_n) > \frac{1}{t(n)}$
- $|p_1(G_n) - q_1(G_n)| < \frac{1}{3t(n)^2}$
- $\mathrm{Prob}(A(\mu_1(G_n)) = 1) - \mathrm{Prob}(A(\nu_1(G_n)) = 1| > \frac{1}{t(n)}$

Der obige Unterscheidungsalgorithmus A bekommt Zufallsvariablen als Eingabe. Um einen Widerspruch zur nichtuniformen Geheimhaltungsbedingung des Bit-Commitment Schemas zu bekommen, brauchen wir einen Algorithmus, der Commitments als Eingabe bekommt und diese unterscheiden kann.

Dazu konstruieren wir uns eine polynomial große Schaltkreisfamilie $\{A_n\}$, die Commitments von zwei unterschiedlichen Folgen von Werten unterscheiden kann.

Konstruktion der Schaltkreisfamilie $\{A_n\}$

Der Schaltkreis A_n hat folgendes einkodiert:

- Den Verifier V^*.
- Den Unterscheidungsalgorithmus A.
- Den Graph G_n.
- Den Hamiltonkreis $\hat{\alpha_n}$ des Eingangsgraphen.

Damit kann der Schaltkreis die Commitments folgender zwei festen Folgen unterscheiden:

1. Die Folge $0^{n^2}1^{n^2}$.
2. Die Folge, in der die Adjazenzmatrix von G_n zweimal hintereinander geschrieben wird.

Der Schaltkreis A_n bekommt ein Commitment einer $2n^2$ langen Folge von Bits als Eingabe und erzeugt daraus eine Folge von n^2 Commitments. Bekommt der Schaltkreis als Eingabe ein Commitment der ersten Folge, so erzeugt er daraus eine Folge, wie sie der Prover dem Verifier vorlegt. Bekommt der Schaltkreis dagegen als Eingabe ein Commitment der zweiten Folge, so erzeugt er daraus eine Folge, wie sie der Simulator erzeugt.

Arbeitsweise des Schaltkreises $\{A_n\}$

Der Schaltkreis bekommt als Eingabe eine Folge $y = (y_1, \ldots, y_{2n^2})$ von Commitments. Jedes y_i ist dabei ein Commitment eines Bits. Wir teilen diese Eingabe in $y_1^0, \ldots, y_{n^2}^0, y_1^1, \ldots, y_{n^2}^1$ auf. Der Schaltkreis wählt als erstes eine zufällige Permutation $\pi \in \mathrm{Perm}(N)$.

Der Schaltkreis erzeugt aus der $2n^2$ langen Eingangsfolge eine n^2 lange Folge wie folgt:

- für alle (i, j), für die $e_{i,j} \in \pi(\hat{\alpha_n})$ wähle zufällig $s_{i,j} \in \{0, 1\}^n$ aus und verschlüssle ein „1“, durch
$$wi, j \;:=\; C_{si,j}(1)$$
- für alle anderen (i, j) wähle
$$w_{i,j} \;:=\; y_{i,j}^b \quad , \; b = e_{i,j}(\pi(G_n)).$$

Das Bit b entscheidet, ob von der ersten Hälfte der Eingangsfolge oder von der zweiten Hälfte der Eingangsfolge das Element für die neue Folge verwendet wird.

Der Schaltkreis A_n wählt zufällig unter Gleichverteilung einen Bitstring $r \in \{0,1\}^{q(n)}$ und schreibt diesen auf das Zufallsband des Verifiers V^*. Dann schreibt der Schaltkreis noch den Graph G_n auf das Eingabeband und die Folge $w_{1,1}, \ldots, w_{n,n}$ auf das Kommunikationsband des Verifiers V^*. Der Schaltkreis startet den Verifier V^* und liest die Herausforderung i_n des Verifiers.

- Falls $i_n = 0$, dann gibt der Schaltkreis „1“ aus.
- Sonst starte den Algorithmus A und gebe

  $$A(G_n, r, (w_{1,1}, \ldots, w_{n,n})$$

  aus

Die Größe des Schaltkreises A_n ist polynomial beschränkt in n. Betrachten wir nun die Wahrscheinlichkeit, daß der Schaltkreis eine „1“ ausgibt, falls er ein zufälliges Commitment der festen Folge $0^{n^2}1^{n^2}$ bekommt. Der Schaltkreis erzeugt aus dieser Eingangsfolge eine Folge $c_{1,1}, \ldots, c_{n,n}$ die die gleiche Verteilung hat wie die Folge die der Prover im Protokoll schickt. Das liegt daran, daß bei Eingabe von $0^{n^2}1^{n^2}$ Prover und Schaltkreis gleich vorgehen. Mit $C(\delta)$ bezeichnen wir ein zufälliges Commitment zu einem Bitstring δ. Der Schaltkreis gibt also mit Wahrscheinlichkeit

$$\mathrm{Prob}(A_n(C(0^{n^2}1^{n^2})) = 1) = (1 - q_1(G_n)) + q_1(G_n) \cdot \mathrm{Prob}(A(\nu_1(G_n)) = 1)$$

eine „1“ aus.

Betrachten wir nun den anderen Fall, daß der Schaltkreis als Eingabe ein zufälliges Commitment der 2. Folge bekommt. Dann erzeugt der Schaltkreis aus dieser Eingangsfolge eine Folge $c_{1,1}, \ldots, c_{n,n}$ die die gleiche Verteilung hat, wie die Folge die der Simulator erzeugt. Die Wahrscheinlichkeit, daß der Schaltkreis nun eine „1“ ausgibt ist

$$\mathrm{Prob}(A_n(C(G_nG_n)) = 1) = (1 - p_1(G_n)) + p_1(G_n) \cdot \mathrm{Prob}(A(\mu_1(G_n)) = 1)$$

Wir nützen nun die drei Bedingungen des 2.Falls aus und nehmen G_n aus der Notation von p_1, q_1 raus und bekommen folgende Abschätzung:

$$|\mathrm{Prob}(A_n(C(0^{n^2}1^{n^2})) = 1) - \mathrm{Prob}(A_n(C(G_nG_n)) = 1)|$$

$$\begin{aligned}
&\geq q_1 \cdot |\mathrm{Prob}(A(\nu_1) = 1) - \mathrm{Prob}(A(\mu_1) = 1)| - 2 \cdot |p_1 - q_1| \\
&> \frac{1}{t(n)} \cdot \frac{1}{t(n)} - 2 \cdot \frac{1}{3 \cdot t(n)^2} \\
&= \frac{1}{3 \cdot t(n)^2}
\end{aligned}$$

Durch die obigen Abschätzungen bekommen wir die Aussage, daß die Schaltkreisfamilie Commitments von der ersten Folge von Commitments der zweiten Folge unterscheiden kann. Durch das Standard-Hybridargument folgt die Aussage, daß eine Schaltkreisfamilie polynomieller Größe existiert, die Commitments von Bits unterscheiden kann, was ein Widerspruch zur nichtuniformen Geheimhaltungsbedingung des Bit-Commitment Schemas ist.

□

Da die Sprache HAM NP-vollständig ist, folgt für jede Sprache S in NP, daß sie polynomial reduzierbar auf die Sprache HAM ist. Es gibt also eine in Polynomialzeit berechenbare Funktion f, so daß

$$x \in S \Longleftrightarrow f(x) \in HAM$$

Die Standardreduktion f_S von S nach HAM hat dabei noch zusätzlich folgende wichtige Eigenschaft:

> Es gibt eine in Polynomialzeit berechenbare Funktion g_S, so daß für alle $(x, w) \in R_S$ gilt, daß $g_S(w)$ ein Hamiltonkreis von $f_S(x)$ ist.

Der „witness“ w für $x \in S$ kann also in einen Hamiltonkreis transformiert werden. Für die normale many-one Reduktion ist die obige Eigenschaft nicht garantiert. Allerdings hat die Standardreduktion f_S, die eine Komposition aus der allgemeinen Reduktion der Sprache S nach SAT, der Reduktion von SAT nach 3SAT und der Reduktion von 3SAT nach HAM ist, diese zusätzliche Eigenschaft. (Die Reduktionen sind in [GJ79] und in [Sch92] dargestellt).

Jetzt können wir eine Konstruktion eines Zero-Knowledge Beweissystems für jede Sprache in NP angeben.

Ein CZK-Beweissystem für eine Sprache S in NP

- Gegeben sind zwei polynomial zeitbeschränkte ITMs (P, V). Die gemeinsame Eingabe ist ein String $x \in S$.
- Der Prover bekommt einen „witness“ w für den gilt $(x, w) \in R_S$ (Ein Zeuge für die Mitgliedschaft von $x \in S$.
- Beide, der Prover und der Verifier berechnen den Graphen $G \stackrel{\text{def}}{=} f_S(x)$. Der Graph G besitzt einen Hamiltonkreis. Der Prover berechnet sich den Hamiltonkreis ψ mit der Funktion g_S aus dem „witness“ w.
- Beide Parteien starten nun einen CZK-Beweis für die Sprache HAM mit der gemeinsamen Eingabe G und der Zusatzeingabe ψ für den Prover.

Behauptung:
Ist das Unterprotokoll für HAM, welches in obigem Protokoll benutzt wird, CZK mit Zusatzeingabe, so ist obiges Protokoll auch CZK mit Zusatzeingabe.

Beweis :
Das obige Protokoll ist ein Interaktives Beweissystem. Das folgt aus der Annahme, daß das Unterprotokoll für HAM ein Interaktives Beweissystem ist und das die Reduktion korrekt ist. Damit liefert diese einen Graph G, der einen Hamiltonkreis besitzt, den der Prover kennt.

Bezüglich der Zero-Knowledge Eigenschaft muß beachtet werden, daß der Verifier für HAM, welcher in obigem Protokoll aufgerufen wird, außer der gemeinsamen Eingabe G auch noch die Zusatzeingabe x besitzt. Diese Zusatzeingabe könnte den Verifier beim Versuch, Wissen aus dem Prover P_{HAM} zu extrahieren, unterstützen, solange der Prover P_{HAM} nicht CZK mit Zusatzeingabe ist.

Wir müssen einen Simulator M_{V^*} angeben, der die Interaktion des Verifiers V^* mit dem Prover simulieren kann. Der Verifier V^* ruft einen anderen Verifier V^{**} auf, der mit dem Prover P_{HAM} bei gemeinsamer Eingabe G und Zusatzeingabe x kommuniziert. Jetzt kommt die Vorraussetzung, daß das Protokoll für HAM CZK mit Zusatzeingabe ist ins Spiel, denn dann muß es einen Simulator $M_{V^{**}}$ geben der bei Eingabe von (G, x) die Interaktion von V^{**} mit P_{HAM} bei gemeinsamer Eingabe G und Zusatzeingabe x des Verifiers V^{**} simulieren kann.

Der Simulator M_{V^*} geht wie folgt vor: Bei Eingabe von x, berechnet sich der Simulator M_{V^*} den Graphen $G \stackrel{\text{def}}{=} f_S(x)$ und gibt dann $M_{V^{**}}$ aus. Damit hat er die Interaktion von V^* mit Prover P simuliert. Der Simulator M_{V^*} braucht dazu nur polynomielle Laufzeit.
□ Damit folgt der Satz:

Satz 6.2 Alle Sprachen in NP haben einen CZK mit Zusatzeingabe Beweis
Unter der Annahme, daß es Commitment Schemas gibt, die die nichtuniforme Geheimhaltungsbedingung, sowie die Eindeutigkeitsbedingung erfüllen hat jede Sprache in NP einen Computational Zero-Knowledge mit Zusatzeingabe Beweis.

Besonders wichtig für praktische Anwendungen ist es dabei, daß sogar der Prover in probabilistischer Polynomialzeit implementiert werden kann, wenn er als Zusatzeingabe einen witness für die Mitgliedschaft der Eingabe x in der Sprache $S \in \text{NP}$ bekommt.

Kapitel 7

Zero-Knowledge Beweissysteme mit mehreren Provern

7.1 Einleitung

In diesem Kapitel betrachten wir eine Erweiterung des Modells des Interaktiven Beweissystems. Dieses Modell kann man erweitern, indem man zuläßt, daß der Verifier mit mehreren Provern, anstatt mit nur einem Prover kommunizieren darf.

Eingeführt wurde das erweiterte IP Modell auf der 20. STOC durch Ben-Or, Goldwasser, Kilian und Wigderson [BGKW88]. Man nennt dieses Modell *Interaktives Beweissystem mit mehreren Provern.*

Beim Modell mit nur einem Prover beruhen die Protokolle auf mathematische Annahmen wie z.B., daß die Faktorisierung von großen zusammengesetzten natürlichen Zahlen nicht in Polynomialzeit (in der Länge der Zahlen) durchführbar ist oder daß Einwegfunktionen existieren. Diese mathematischen Annahmen konnten bisher leider noch nicht bewiesen werden. Die Vermutung ist aber, daß diese Annnahmen doch stimmen, denn sonst würden einige sehr unwahrscheinliche Resultate folgen. Beispielsweise existieren Einwegfunktion genau dann wenn $P \neq NP$ ist (siehe dazu [BDG88, Seite 63]).

Wünschenswert wäre es, korrekte Zero-Knowledge Protokolle konstruieren zu können, welche nicht von unbewiesenen mathematischen Annahmen abhängen. (Um sozusagen für alle Fälle gerüstet zu sein). Dies ist im Modell mit mehreren Provern möglich. Es wird hier allerdings vorausgesetzt, daß die Prover während des Protokolls nicht miteinander kommunizieren können. Mit Hilfe eines Bit-Commitment-Schemas konnten [BGKW88] sogar zeigen, daß alle Sprachen in NP ein 2-Prover interaktives Beweissystem mit der perfekten Zero-Knowledge Eigenschaft haben, ohne eine noch nicht bewiesene Annnahme zu verwenden.

Eine Frage, die sich aufdrängt ist: Ändert sich die Klasse von Aussagen, die mit diesem erweiterten Modell bewiesen werden können, gegenüber dem ursprünglichen Modell mit nur einem Prover. Man könnte ja meinen, daß ein allmächtigen Prover genügen könnte. Diese interessante Frage ist immer noch offen. Intuitive Argumente dafür, daß Beweissysteme mit mehreren Provern mächtiger sein könnten, sind :

- Betrachten wir das Verhör mehrerer Verdächtiger bei der Polizei. Die Verdächtigen werden getrennt befragt. Sie wissen also nicht, was ihre Komplizen gefragt wurden und vor allem nicht, was sie geantwortet haben.
  - Falls sie lügen, werden sich leicht Inkonsistenzen in ihren Aussagen finden lassen.
  - Falls sie unschuldig sind und die Wahrheit sagen, erhöht sich andererseits die Glaubwürdigkeit ihrer Aussagen, wenn alles übereinstimmt.

  Die Verdächtigen haben hier die Rolle der Prover, die Polizei die Rolle des Verifiers.
- Generell läßt sich sagen, daß wenn nur eine Person eine seltsame Aussage macht, (z.B. ein UFO gesehen zu haben) es weniger glaubwürdig ist, als wenn mehrere Personen unabhängig voneinander diese Aussage machen.

7.2 Das neue Modell

7.2.1 Definitionen und Bezeichnungen

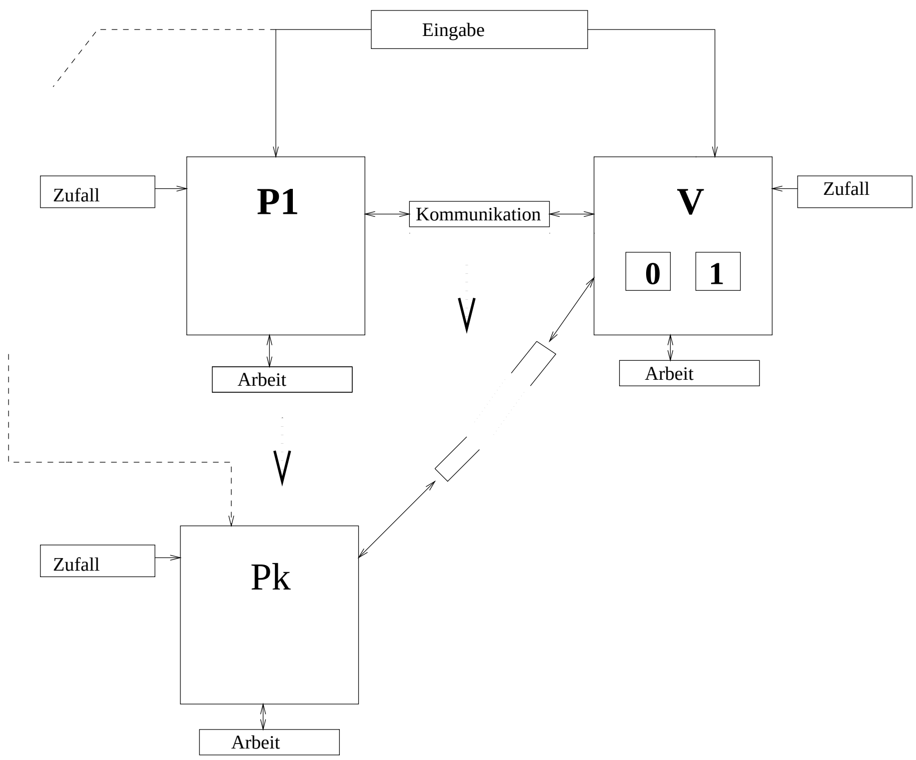

Abbildung7.1: Das neue Modell

Definition 7.1

Sei V eine probabilistische Polynomialzeit beschränkte ITM und $P_1, \ldots, P_k$ Maschinen mit unbeschränkter Rechenkraft. Die Anzahl k der Prover ist dabei durch ein Polynom in der Eingabelänge beschränkt. Alle Maschinen haben nur auf ein gemeinsames Eingabeband Lesezugriff. Der Verifier hat mit jedem Prover P_i ein Kommunikationsband gemeinsam. Zwei verschiedene Prover P_i und P_j können, während das Protokoll läuft, nicht miteinander kommunizieren. Jede Maschine hat noch ein Arbeitsband auf dem geschrieben und gelesen werden kann, sowie ein Zufallsband von dem nur gelesen werden kann. Dann nennt man das $k+1$ Tupel $(P_1, \ldots, P_k, V)$ ein **k-Prover Interaktives Protokoll.**

Definition 7.2

Sei $L \subseteq \{0,1\}^*$
L hat ein **k-Prover Interaktives Beweissystem**, *wenn ein probabilistischer Polynomialzeit beschränkter Verifier V existiert, so daß*

1. *Vollständigkeit gilt, d.h. $\forall x \in L$:*
   $\exists P_1, \ldots, P_k$, so daß $(P_1, \ldots, P_k, V)$ ein interaktives Protokoll ist und
   $Prob(\langle P_1, \ldots, P_k, V\rangle(x) = 1) \geq \frac{2}{3}$

2. *Korrektheit gilt, d.h. $\forall x \notin L$:*
   $\forall \hat{P}_1, \ldots, \hat{P}_k$, so daß $(P_1, \ldots, P_k, V)$ ein interaktives Protokoll ist
   und $Prob(\langle P_1, \ldots, P_k, V\rangle(x) = 1) \leq \frac{1}{3}$

Einige Erläuterungen zu den Definitionen:

- Die Wahl der Konstanten $\frac{2}{3}$ bzw. $\frac{1}{3}$ in der obigen Definition sind rein willkürlich. Genau wie im 1-Prover Fall gilt: Ein Protokoll mit den Wahrscheinlichkeiten $\frac{1}{2} + \epsilon, \frac{1}{2} - \epsilon$ für das Akzeptieren bzw. Verwerfen kann man in ein Protokoll mit den Wahrscheinlichkeiten $1 - \epsilon, \epsilon$ umformen, indem man das Protokoll polynomial oft hintereinander ausführt und dann einen Mehrheitsentscheid macht.

- Die k-Prover versuchen den Verifier zu überzeugen, daß die Eingabe x eine bestimmte Eigenschaft besitzt. Der Verifier fängt zunächst einmal an zu würfeln und stellt dann abhängig von der Eingabe x und seinen Zufallsbits Fragen an die Prover. Er kann an mehrere Prover gleichzeitig Fragen stellen, indem er auf das jeweilige

Kommunikationsband i die Frage für den Prover P_i schreibt. Die befragten Prover schreiben ihre Antwort auf ihr Kommunikationsband. Damit ist eine *Runde* beendet. Die Anzahl der Runden ist dabei durch ein Polynom in der Länge der Eingabe begrenzt.

- Falls sich die Prover während des Protokolls unterhalten können, dann wissen sie, was die anderen Prover gefragt wurden und könnten diese Information benutzten, um abhängig davon die nächste Frage zu beantworten. Ein Prover hat also alle Informationen, die die anderen (k-1)-Prover auch haben und so kann man alle k-Prover zu einem zusammenfassen. Deshalb haben wir in der Definition gefordert, daß sich die Prover während des Protokolls nicht unterhalten können.

  Die Prover können sich aber vor Beginn des Protokolls absprechen und auf eine gemeinsame Vorgehensweise einigen. Um sich Abzusprechen genügt es, daß sie, bei gegebener Eingabe x, ein gemeinsames Zufallsband benutzen. Dann kann jeder Prover die Vorberechnungen jedes anderen Provers durchführen.

MIP ist die Klasse von Sprachen, die ein Interaktives Beweissystem mit mehreren Provern haben. Mit **IP(j,k)** bezeichnet man die Klasse von Sprachen, die von einem Interaktivem Beweissystem mit maximal j Provern und in nicht mehr als k Runden akzeptiert werden. Dann ist :

$$\begin{array}{ll}\textbf{IP} & = \mathrm{IP}(1, poly) \text{ und} \\ \textbf{MIP} & = \mathrm{IP}(poly, poly)\end{array}$$

7.3 2-Prover Beweissysteme

7.3.1 Definitionen und Bezeichnungen

Wie wir im Anhang durch einen Ringschluß zeigen werden, genügt es, sich auf zwei Prover zu beschränken. Noch mehr Prover zu betrachten ist unnötig, denn jedes Protokoll mit polynomiell vielen Provern kann durch ein Protokoll /footnoteFalls die Rundenzahl nicht beschränkt ist mit nur zwei Provern simuliert werden.

Das 2-Prover Modell ist ein Spezialfall des 2-Partner Modells, welches zunächst definiert werden soll.

Definition 7.3 (2-Partner Modell)

Dieses Modell besteht aus drei ITMs. zwei ITMs – **die Partner** *genannt werden – kommunizieren mit der dritten ITM, dem* **Selbstständigen**.

Die Partner können sich, vor Beginn der Interaktion mit dem Selbstständigen, auf eine gemeinsame Strategie einigen. Dafür benutzen sie ein gemeinsames Zufallsband. Der

Selbstständige besitzt ein eigenes Zufallsband. Die ITMs sind wie folgt miteinander verbunden.

- *Die Eingabebänder aller drei ITMs sind miteinander verbunden.*
  Auf diesen Bändern lesen die ITMs ihre gemeinsame Eingabe ein.
- *Der Selbstständige besitzt zwei Kommunikationsbänder. Das erste Kommunikationsband ist mit dem Kommunikationsband des ersten Partners verbunden, das zweite mit dem des zweiten Partners.*
- *Der Selbstständige hat noch zwei Schaltbänder, welche wie die Kommunikationsbänder mit den Provern verbunden sind.*
- *Die Partner haben die gleiche Identität, der Selbstständige hat die Gegenidentität.*
- *Bei einer Eingabe x besteht die gemeinsame Berechnung des 2-Partner Modells aus einer Folge von Tripeln. Jede Komponente des Tripels stellt die Konfiguration einer ITM da.*

Mit $\langle P_1, P_2, S\rangle(x)$ bezeichnen wir die Ausgabe des Selbstständigen S nach der Interaktion mit den Partnern P_1, P_2 bei der gemeinsamen Eingabe x.

In dem 2-Partner Modell ist den ITMs bzgl. Berechnungsressourcen keine Beschränkung auferlegt. Wird der Selbstständige auf eine probabilistische Polynomialzeit beschränkte Maschine reduziert, so gelangt man zum 2-Prover Modell. Die 2 Partner heißen in diesem Modell Prover und der Selbstständige wird Verifier genannt.

Die Zero-Knowledge Eigenschaft beim 2-Prover Modell wird analog zum Modell mit nur einem Prover definiert. Weil wir im übernächsten Abschnitt zeigen wollen, daß alle Sprachen in NP ein 2-Prover interaktives Beweissystem mit der perfekten Zero-Knowledge Eigenschaft haben schreiben wir diese Definition hier explizit auf.

Definition 7.4 (perfektes Zero-Knowledge)

Ein 2-Prover Beweissystem (P_1, P_2, V) für eine Sprache L hat die **perfekte Zero-Knowledge** *Eigenschaft, falls für jeden probabilistisch Polynomialzeit begrenzten Verifier V^* ein probabilistischer im Erwartungswert Polynomialzeit begrenzter Simulator M^* existiert, so daß für alle $x \in L$ die Zufallsvariablen $\langle P_1, P_2, V^*\rangle(x)$ und $M^*(x)$ die gleiche Verteilung haben.*

7.3.2 Ein Bit Commitment Schema für das 2-Partner Modell

In diesem Abschnitt wird ein Bit Commitment Schema für das 2-Partner Modell definiert und eine konkrete Konstruktion für ein solches Schema angegeben. Dieses Bit-Commitment Schema, das ohne kryptographische Annahmen auskommt, wird im Beweis, daß alle Sprachen in NP ein 2-Prover Interaktives Beweissystem mit der perfekt Zero-Knowledge Eigenschaft haben, angewendet.

Ein Commitment Schema besteht aus zwei Phasen, der *Festlegungsphase* und der *Öffnungssphase*. Da wir in unserem Modell auch zwei Partner haben, verteilen wir die zwei Phasen auf die zwei Partner wie folgt.

Der erste Partner nimmt nur an der Festlegungsphase teil.
Der zweite Partner nimmt dagegen nur an der Öffnungssphase teil.

Die Partner werden hier als *Sender*, der Selbstständige als *Empfänger* bezeichet.

Prinzipiell läuft das Bit-Commitment Schema folgendermaßen ab :

Sender 1 legt sich gegenüber dem Empfänger in der Festlegungsphase auf ein Bit fest, und der Empfänger läßt sich in der Öffnungssphase dieses Bit von Sender 2 öffnen, so daß folgende Bedingungen gelten :

1. *Geheimhaltung* (informations theoretische) :
   Der Empfänger hat am Ende der Festlegungsphase überhaupt keine Information über das festgelegte Bit bekommen.
2. *Eindeutigkeit* (der Festlegung) :
   Falls der zweite Sender, den Wert 0 mit Wahrscheinlichkeit p öffnen kann, so kann er den Wert 1 mit nur vernachlässigbar größerer Wahrscheinlichkeit als 1-p öffnen. Kann er den Wert 1 nur mit genau der Wahrscheinlichkeit 1-p öffnen, dann heißt die Festlegung *total eindeutig.*

Beide obige Bedingungen sind unabhängig von der Rechenleistung der beteiligten Maschinen formuliert. Im Modell mit nur einem Prover ist es nicht möglich, diese beiden Bedingungen gleichzeitig zu erfüllen.

Definition 7.5 (Bit-Commitment-Schema mit zwei Sendern)

Betrachtet wird wieder das zwei Partner Modell. Ein Tripel (S_1, S_2, E) von Polynomialzeit begrenzten ITMs heißt **Bit-Commitment-Schema mit zwei Sendern***, falls folgendes gilt:*

- *Alle Maschinen bekommen eine Zahl n in unärer Kodierung (also als 1^n) auf das gemeinsame Eingabeband. Diese Eingabe heißt* Geheimhaltungs-Parameter. *Der Parameter n bestimmt auch die Wahrscheinlichkeit, mit der von der totalen Eindeutigkeit abgewichen werden darf. Die zwei Sender haben eine zusätzliche Eingabe $b \in \{0,1\}$. Die Eingabe b ist das Bit, auf das sich die Sender festlegen.*

- **Geheimhaltung:**
  Die Festlegungen für 0 bzw 1 haben die gleiche Verteilung. Dann gilt:
  Für jede beliebige probabilistische Maschine E^*, die mit dem ersten Sender kommuniziert, haben die Zufallsvariablen $\langle S_1(0), E^* \rangle (1^n) und \langle S_1(1), E^* \rangle (1^n)$ die gleiche Verteilung.

- **Eindeutigkeit:**
  Die Zufallsvariable view *bezeichnet die Sicht des Empfängers E von der Interaktion mit dem ersten Sender und besteht aus allen wesentlichen Bestandteilen der Kommunikation, die der Empfänger sehen kann.*
  Das sind die Zufallsbits $\overline{r}$, welche der Empfänger benützt hat, und natürlich die Folge von Nachrichten, die er vom ersten Sender erhalten hat. Diese Nachrichten stellen die Festlegung dar. Die View *läßt sich somit als Tupel $(\overline{r}, \overline{m})$ darstellen.*

  Sei $\sigma \in \{0, 1\}$. Man bezeichnet einen String s als mögliche σ-Öffnung von der View *$(\overline{r}, \overline{m})$ des Empfängers, falls der erste Sender die Zufallsbits s benützt, bei Eingabe von σ und 1^n, um sich bei der Interaktion mit E auf $\overline{m}$ festzulegen. Der Empfänger E benützt bei dieser Interaktion die Zufallsbits r.*

  Gegeben sei eine View *$(\overline{r}, \overline{m})$.*
  Interessant ist die Frage: Ist $\overline{m}$ eine Festlegung von 0 oder von 1 bzgl. $\overline{r}$.? Oder genauer: Mit welcher Wahrscheinlichkeit kann $\overline{m}$ als 0 bzw 1 geöffnet werden?

  Betrachten wir dazu eine beliebige Strategie S_1^* für den ersten Sender. Sei p eine reelle Zahl und $\sigma \in \{0, 1\}$ Alle möglichen Strategien von Sender S_2 können als Zufallsvariablen aufgefaßt werden. Da der Sender S_2 nie die Zufallsbits $\overline{r}$ des Empfängers zu Gesicht bekommt, sind die möglichen Strategien und damit die Zufallsvariablen unabhängig von den Zufallsbits $\overline{r}$.

  Gilt für alle solche Zufallsvariablen X, daß die Wahrscheinlichkeit das X eine mögliche σ-Öffnung von der View *des Empfängers (bei der Interaktion mit S_1^*), höchstens p ist, dann ist p eine obere Schranke der Wahrscheinlichkeit einer σ-Öffnung von dieser* View. *Für jedes $\sigma \in \{0, 1\}$ bezeichnet p_σ die obere Schranke der Wahrscheinlichkeit einer σ-Öffnung von der* View *des Empfängers bei Interaktion mit S_1^*.*

  Die View *des Empfängers bei Interaktion mit S_1^* (und damit die Festlegung $\overline{m}$) nennt man* **eindeutig**, *falls gilt:*
  $$p_o \; + \; p_1 \; \leq 1 \; + \; 2^{-n}$$
  Das heißt nur mit einer vernachlässigbar kleinen Wahrscheinlichkeit (in Abhängigkeit des Sicherheits-Parameters n) darf die Eindeutigkeit der Festlegung von der totalen Eindeutigkeit abweichen.

Konstruktion eines Bit Commitment mit zwei Sendern

Alle beteiligten Maschinen bekommen als Eingabe den Sicherheits-Parameter n unär (d.h. als 1^n) kodiert. Dieser begrenzt auch die Anzahl der Bits die festgelegt werden sollen.

Auf dem gemeinsamen Zufallsband der Sender werden je zwei Bits zusammengefaßt. Es ergeben sich mit zwei Bits vier verschiedene Möglichkeiten. Die Kombinationen $00, 01, 10$ werden als die Zahlen $0, 1$ und 2 interpretiert. Die Kombination 11 wird einfach überlesen. Diese Zusammenfassung von Bits führt man solange aus, bis sich n mal eine der Kombinationen $00, 01$ oder 10 ergeben hat. Nun kann davon ausgegangen werden, daß sich auf dem Zufallsband der Sender eine gleichverteilte Folge $\overline{s}$, der Länge n, von Elementen aus $\{0, 1, 2\}$ befindet. Also $\overline{s} = s_1 \cdots s_n$, wobei $s_i \in \{0, 1, 2\}$ für $1 \leq i \leq n$.

Die Festlegung geschieht mit den zwei Funktionen

$$\pi_0 := \begin{cases} 0 & \mapsto & 0 \\ 1 & \mapsto & 1 \\ 2 & \mapsto & 2 \end{cases}$$

$$\pi_1 := \begin{cases} 0 & \mapsto & 0 \\ 1 & \mapsto & 2 \\ 2 & \mapsto & 1 \end{cases}$$

Diese Funktionen sind allen Beteiligten bekannt.

- **Festlegungsphase:**

  1. Der Empfänger wählt unter Gleichverteilung eine Folge $\overline{r}$ von Bits der Länge n, indem er diese von seinem Zufallsband einliest.
     $\overline{r} = (r_1 \cdots r_n) \in \{0, 1\}^n$.
     Der Empfänger schickt $\overline{r}$ an den ersten Sender.
     $E \longrightarrow S_1 \;:\; \overline{r}$
  2. Der erste Sender legt sich auf ein Bit b_i fest, indem er folgendes berechnet

     $$c_i = \pi_{r_i}(s_i) + b_i \bmod 3.$$

     Der erste Sender legt sich auf n Bits fest und schickt die Folge $\overline{c} \in \{0, 1, 2\}^n$ an den Empfänger zurück.
     $S_1 \longrightarrow E \;:\; \overline{c}$

- **Enthüllungsphase:**

  1. Der zweite Sender schickt an den Empfänger die vom ersten Sender zum Festlegen benützte Folge $\overline{s}$, sowie die Folge von Festgelegten Bits $\overline{b}$.
     $S_2 \longrightarrow E \;:\; \overline{s}, \overline{b}$
  2. Der Empfänger überprüft die Festlegung, indem er mit den bekommenen Folgen $\overline{s}, \overline{b}$ und seinen Zufallsbits $\overline{r}$ eine Folge $\overline{d}$ berechnet und diese mit der Festlegung $\overline{c}$ vom ersten Sender vergleicht. Diese Berechnungen sind alle in Polynomialzeit durchführbar.

Der zweite Sender kann die Festlegung $\overline{c}$ als jede beliebige n Bit lange Folge öffnen sobald er weiß welche der beiden Funktion π_0, π_1 der erste Sender zum Festlegen des jeweiligen Bits verwendet hat. Nun könnten sich die beiden Sender vor Beginn des Commitment Schemas über die Benutzung der jeweiligen Funktion absprechen. Also muß der Empfänger auf die Benutzung der Funktionen π_0, π_1 so Einfluß nehmen können, daß eine vorherige Absprache der Sender die Eindeutigkeits-Bedingung der Festlegung nicht verletzt. Der Empfänger trifft mit dem Lesen seiner Zufallsbits eine Auswahl der Funktionen. Diese Wunschauswahl gibt er **nur** dem ersten Sender bekannt. Dieser muß nun eine zum Auswahlwunsch des Empfängers konsistente Festlegung berechnen, denn diese Festlegung muß der Überprüfung des Empfängers in der Öffnungsphase standhalten. Wichtig ist hier wieder die Forderung, daß die Sender, während des Commitment Schemas nicht miteinander kommunizieren dürfen. So bleibt die Wunschauswahl des Empfängers dem zweiten Sender verborgen.

Behauptung: Die obige Konstruktion ist ein Bit-Commitment-Schema für 2 Sender.

Beweis :

1. **Geheimhaltungs-Bedingung:**
   Unabhängig von dem Bit, das festgelegt werden soll liefern die beiden Funktionen π_0, π_1, bei gleichverteiltem $\overline{s}$ eine gleichverteilte Folge aus $\{0,1,2\}^n$, für jedes $\overline{r} \in \{0,1\}^n$. D.h. der Empfänger hat am Ende der Festlegungsphase überhaupt keine Information über die festgelegten Bits $\overline{b}$ bekommen, egal welches $\overline{r}$ er ausgewählt hat. Damit ist die Geheimhaltungs-Bedingung erfüllt.

2. **Eindeutigkeits-Bedingung:** Wir beweisen diese Bedingung durch einen Widerspruchsbeweis. Existiert eine beliebige probabilistische Strategie für den ersten Sender, so daß die Festlegung $\overline{c}$ von S_1 nicht eindeutig ist, bzgl. $\overline{r}$, dann existiert auch eine deterministische Strategie, welche auch nicht eindeutig ist. Das heißt diese Strategie ist eine Funktion f, die einen n-Bit langen Zufallsstring als Argument auf eine Folge $\overline{c}$ aus $\{0,1,2\}^n$ abbildet. Dann können auch die Strategien des zweiten Senders zum Öffnen der Festlegung als 0 bzw. 1 als deterministische Strategien angenommen werden. Die deterministischen Strategien bestehen aus konstanten Folgen $\overline{s}^0$ und $\overline{s}^1$, wobei $\overline{s}^0, \overline{s}^1 \in \{0,1,2\}$.

   Betrachten wir zunächst den Fall n = 1.

   Der Fall n = 1:
   Der Empfänger hat am Ende der Festlegungsphase eine Festlegung c bekommen. Wir wollen zeigen, daß es für die zwei Sender unmöglich ist, die Festlegung c immer in beide Richtungen (also als 0 und als 1) zu öffnen. Annahme: Es geht doch.

   > Der zweite Sender S_2 könnte zwei verschiedene Öffnungen von einer Festlegung c senden.
   > Er sendet in der Öffnungsphase also $(0, s^0)$ und $(1, s^1)$.
   > s^0 ist eine mögliche Öffnung der 0, s^1 eine mögliche Öffnung der 1. Beides in Bezug auf die *View* des Empfängers.

Da die beiden Öffnungen vom zweiten Sender gesendet werden, ohne die Funktionsauswahl des Empfängers zu kennen, müssen sie für jede Auswahl passen (also passend sein für alle möglichen *Views* des Empfängers). Das heißt für jedes $r \in \{0,1\}$ müssen $\pi_r(s^0)$ und $\pi_r(s^1)+1$ modulo 3 gerechnet, mit der Festlegung c übereinstimmen.
Also muß gelten:

$$\pi_0(s^0) \equiv_3 \pi_0(s^1) + 1 \equiv_3 c \tag{7.1}$$
$$\pi_1(s^0) \equiv_3 \pi_1(s^1) + 1 \equiv_3 c \tag{7.2}$$

Wegen 7.1 gilt: $s^0 \equiv_3 s^1 + 1$.
In 7.2 eingesetzt ergibt sich:

$$\pi_1(s^1 + 1 \bmod 3 \equiv_3 \pi_1(s^1) + 1 \tag{7.3}$$

Für jedes $s \in \{0,1,2\}$ gilt aber

$$\pi_1(s^1 + 1 \bmod 3 \not\equiv_3 \pi_1(s^1) + 1,$$

und damit kann Gleichung 7.3 nicht erfüllt werden. Das ist ein Widerspruch zur Annahme, also können die Sender die Festlegung nicht immer in beide Richtungen öffnen.

Der Empfänger wählt unter Gleichverteilung einen String $r \in \{0,1\}^n$. Damit läßt sich die *View* des Empfängers als $(U_n, f(U_n))$ darstellen. Für jedes $\sigma \in \{0,1\}$, bezeichnet p_σ die Wahrscheinlichkeit, daß die Folge $\overline{s}^\sigma$ eine mögliche σ-Öffnung der view des Empfängers ist.

Wir nehmen an, die Eindeutigkeitsbedingung gilt nicht. Dann folgt daraus

$$p_0 + p_1 \; > \; 1 + 2^{-n}$$

Wir wollen durch R^σ die Menge aller Strings $r \in \{0,1\}^n$ bezeichnen für die die Folge $\overline{s}^\sigma$ eine mögliche σ-Öffnung der view $(r, f(r))$ des Verifiers ist. Formal definiert ist R^σ:

$$R^\sigma = \{r : (\forall i) f_i(r) \equiv_3 \pi_{r_i}(s_i^\sigma) + \sigma\}$$

Es folgt also (durch die obige Wahrscheinlichkeiten)

$$|R^0| + |R^1| / geq 2^n + 2$$

Das heißt also mindestens zwei Elemente müssen in beiden Mengen enthalten sein.

Wir wollen nun zeigen, daß der Durchschnitt beider Mengen nicht mehr als ein Element enthalten kann.

Behauptung: $|R^0 \cap R^1| \leq 1$

Beweis :
Nehmen wir das Gegenteil an. Es gibt zwei Folgen $a, b \in R^0 \cap R^1$, $a \neq b$. Dann gibt es ein i, so daß $a_i \neq b_i$. Beispielsweise sei $a_i = 0$. Es folgt durch die Definition von R^σ:

$$\begin{aligned} f_i(a) &\equiv_3 \pi_0(s_i^0) \\ f_i(b) &\equiv_3 \pi_1(s_i^0) \\ f_i(a) &\equiv_3 \pi_0(s_i^1) \\ f_i(b) &\equiv_3 \pi_1(s_i^1) \end{aligned}$$

Der Widerspruch folgt jetzt genau wie im Fall n = 1. □ □

PZK für alle Sprachen in NP

In dem Modell mit zwei Provern ist es sogar möglich für jede Sprache in der Klasse NP einen Perfekt Zero-Knowledge Beweis anzugeben.

Es wird wieder ein NP-vollständiges Problem ausgwählt, z.B. das Hamiltonkreisproblem. In einem ZK Protokoll für ein NP-vollständiges Problem spielen die Bit-Commitment Schemas eine wichtige Rolle. Im zwei Prover Modell können wir das im vorigen Abschnitt eingeführte Bit-Commitment Schema verwenden. Im Gegensatz zum ein Prover Modell hat dieses Bit-Commitment Schema eine informationstheoretische Geheimhaltungsbedingung, wodurch sogar die Konstruktion eines PZK Protokolls für NP-Sprachen ermöglicht wird.

Im Fall des Hamiltonkreis Problems übernimmt hier Prover 1 die Verschlüsselung der Adjazenzmatrix mit dem obigen Bit Commitment Schema. Der Verifier wählt wieder ein Bit aus und schickt dieses an den zweiten Prover. Der zweite Prover muß nun die gewünschten Bits öffnen.

Anhang A

Äquivalente Charakterisierungen des MIP Modells

A.1 Ein Ringschluß

In diesem Abschnitt zeigen wir eine äquivalente Charakterisierung des Beweissystems mit mehreren Provern durch probabilistische Orakel-Maschinen. Es ist klar, daß IP $\subseteq$ MIP gilt. Offen ist immer noch, ob IP echt in MIP enthalten ist. Man könnte nun folgendes vermuten: Je mehr Prover zugelassen sind, desto größer wird die Klasse (siehe Einleitung zu Kapitel 7). Wir zeigen, daß man sich auf zwei Prover beschränken kann, d.h. IP$(poly, poly)$ = IP$(2, poly)$. Der Fall, daß polynomiell viele Prover 2 Prover simulieren, ist trivial. Die nächsten zwei Abschnitte beschäftigen sich mit den 2 anderen Schlüssen.

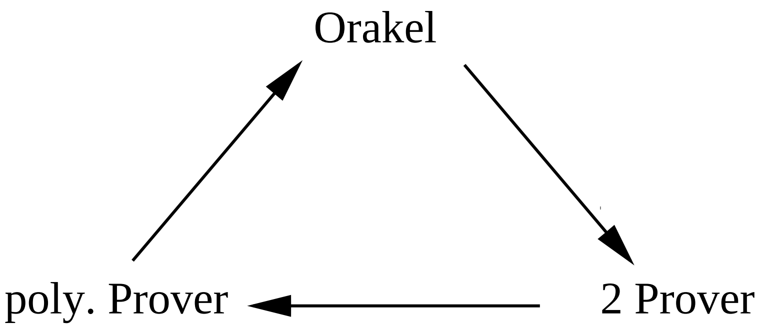

Abbildung A.1: Der Ringschluß

A.1.1 Polynomiell viele Prover $\Rightarrow$ Orakel

Definition 9.1
Sei M eine probabilistische Polynomialzeit beschränkte Turingmaschine mit Zugriff auf ein Orakel O. Eine Sprache L wird von M akzeptiert, falls :

1. $\forall x \in L \quad \exists$ *Orakel O, so daß* M^O *akzeptiert mit Wahrscheinlichkeit* $\geq \frac{2}{3}$.
2. $\forall x \notin L \quad \forall$ *Orakel* $\hat{O}$: $M^{\hat{O}}$ *akzeptiert mit Wahrscheinlichkeit* $\leq \frac{1}{3}$.

Wir haben definiert, was es bedeutet wenn eine Sprache von einer prob. Orakel TM akzeptiert wird. Jetzt wollen wir zeigen :

Satz 9.1
L wird akzeptiert von einem Beweissystem mit mehreren Provern $\Longrightarrow$ *L wird akzeptiert von einer prob. Orakelmaschine.*

Beweis :

Annahme: L wird von einem MIP-Beweissystem akzeptiert.
Definiere M wie folgt :

- M simuliert den Verifier V. Das kann er machen, weil beides probabilistische Polynomialzeit Maschinen sind. M merkt sich dabei alle Nachrichten.
- Falls V an den Prover i eine Nachricht schickt, stellt M seinem Orakel die Frage f := $(x, i, j, l, q_1^i, \ldots, q_j^i)$. f ist dabei passend kodiert. x ist die gemeinsame Eingabe, i bezeichnet den i-ten Prover, j die j-te Frage, $q_1^i, \ldots, q_j^i$ sind alle bisher gestellten Fragen an Prover i.
- Die Antwort des Orakels wird als l'tes Bit der Antwort auf die j-te Frage benutzt. Also muß M für jedes Bit eine Frage stellen, er stellt somit polynomiell viele Fragen, um die polynomiell lange Prover-Antwort zu bekommen.
- M akzeptiert genau dann, wenn V akzeptiert.

1.Fall x $\in$ L :
Dann gibt es Prover $P_1, \ldots, P_k$, die V mit einer Wahrscheinlichkeit von mindestens $\frac{2}{3}$ überzeugen. In das Orakel sind die Nachrichten der Prover kodiert $\Longrightarrow$ M^O akzeptiert $\forall x \in$ L mit derselben Wahrscheinlichkeit wie V.

2.Fall x $\notin$ L :

Annahme, es gibt ein Orakel $\hat{O}$, so daß $M^{\hat{O}}$ x mit größerer Wahrscheinlichkeit als $\frac{1}{3}$ akzeptiert.

Dann kann man Prover $P_1, \ldots, P_k$ konstruieren, welche V bzgl. x mit der gleichen Wahrscheinlichkeit überzeugen, indem sie $\hat{O}$ verwenden, um ihre Nachrichten zu erzeugen. Da keine solche Prover $P_1, \ldots, P_k$ nach Definition existieren, kann auch kein solches Orakel $\hat{O}$ existieren.

□

A.1.2 Orakel $\Rightarrow$ 2-Prover

Wir zeigen, wenn ein Sprache L durch eine probabilistische Polynomialzeit beschränkte Orakelmaschine akzeptiert wird, dann wird sie auch durch ein 2- Prover Interaktives Beweissystem akzeptiert.

Gegeben sei eine probabilistische Polynomialzeit beschränkte Orakelmaschine M, welche L akzeptiert. Dabei bezeichnet x die Eingabe. Dann akzeptiert folgendes 2-Prover Beweissystem (P_1, P_2, V) die Sprache L.

1. Der Verifier V rät ein r $\in_R \{0,1\}^{p(|x|)}$
   V $\longrightarrow$ P_1 : r

2. Prover P_1 berechnet alle Orakelantworten $b_1, \ldots, b_t$ auf die Fragen von M
   $P_1 \longrightarrow$ V : $b_1, \ldots, b_t$

3. V berechnet sich mit r und $b_1, \ldots, b_t$ alle Fragen $q_1, \ldots, q_t$
   V wählt i $\in_R [1, q]$
   V $\longrightarrow$ P_2 : q_i

4. P_2 berechnet Antwort a_i
   $P_2 \longrightarrow$ V : a_i

if $a_i = b_i$ then
   V simuliert M; bei i-ter Frage q_i verwende b_i als Antwort und
   V akzeptiert $\Leftrightarrow$ M akzeptiert
else output "Betrug"
end

Zu (1) :

V rät einen polynomiell langen Zufalls-Bitstring aus und schickt diesen an den Prover P_1.

Zu (2) :

P_1 kann sich mit den Zufallsbits vom Verifier die Fragen selbst generieren und berechnet nun die Antworten (entsprechend dem Orakel) darauf.
Er schickt die Antworten zurück an den Verifier.

Zu (3)(4) :

Der Verifier sucht sich unter Gleichverteilung eine Frage aus und überprüft die Antwort, die er von P_1 bekommen hat, mit Hilfe von P_2, indem er diesem auch diese Frage stellt.
Nur wenn die beiden Antworten von P_1 und P_2 übereinstimmen und wenn M bei den gegebenen Antworten akzeptiert, akzeptiert auch V.

Wir bezeichnen obiges Protokoll als A(x). Um die benötigte Wahrscheinlichkeit fürs Akzeptieren bzw. Verwerfen zu bekommen, führt man das Protokoll A(x) polynomiell mal seriell aus und nennt dieses Protokoll nun B(x).

```
for i := 1 to l do
   if A(x) = akzeptiert then
      z := z + 1;
   elsif A(x) = "Betrug" then
      verwerfe;
end
if z ≥ l/2 then
   akzeptiere
else
   verwerfe.
```

Wir setzen das Polynom l in der obigen Schleife auf den Wert 8tn. t ist die Anzahl von Fragen der Maschine M und n die Länge der Eingabe x. Es wird ein Mehrheitsentscheid durchgeführt, d.h. insgesamt akzeptiert das Protokoll nur, wenn mehr als 4tn Durchläufe von A(x) akzeptiert haben. Wird allerdings einmal ein Betrug von A(x) aufgedeckt, verwirft das Protokoll sofort. Ist das obige Protokoll nun ein 2-Prover interaktives Beweissystem?

Satz 9.2
Das Protokoll B(x) ist ein 2-Prover interaktives Beweissystem.

Beweis :

Fall x $\in$ L :
Die Prover verhalten sich wie im Protokoll beschrieben. Sie liefern also die Antworten so wie es das Orakel tut. Da es nach Vorraussetzung ein Orakel gibt, welches die Maschine M mit Wahrscheinlichkeit von mind. $\frac{2}{3}$ überzeugt, können die Prover den Verifier auch in jedem Schleifendurchgang mit mind. Wahrscheinlichkeit $\frac{2}{3}$ überzeugen. Betrachtet man jede Iteration von A(x) als Zufallsvariable X_i, dann ist, im schlechtesten Fall, $\Pr[X_i = 1] = p_i = \frac{2}{3}$. X sei $\sum_1^{8tn} X_i$ Der Erwartungswert E(X) ist dann $E(X) = 8tn \cdot \frac{2}{3}$.
Wir schätzen mit der *Chernoff-Bound* die Wahrscheinlichkeit ab, daß M nicht akzeptiert.
Zunächst berechnen wir die relative Abweichung δ.

$$\begin{aligned} X \ < \ (1-\delta)\cdot\frac{2l}{3} &= \frac{l}{2} \\ 1 - \delta &= \frac{3}{4} \\ \delta &= \frac{1}{4} \end{aligned}$$

$$\mathbf{Pr}[X \leq (1-\delta)E(X)] \leq \exp\left(\frac{-E(X)\delta^2}{2}\right)$$

$$\mathbf{Pr}\left[X \leq \left(1 - \frac{1}{4}\right)\frac{16tn}{3}\right] \leq \exp\left(\frac{-\frac{16tn}{3}\frac{1}{16}}{2}\right) \leq exp(-\Theta(n))$$

Das heißt mit Wahrscheinlichkeit 1 - $e^{-\Theta(n)}$ akzeptiert B(x).

Fall x $\notin$ L :

Falls sich die Prover wieder so verhalten, wie es das Protokoll vorgibt, so schaffen sie es, nur mit exponentiell kleiner Wahrscheinlichkeit M zum Akzeptieren zu bringen. Die Prover (O.B.d.A. Prover P_1) könnten aber versuchen zu betrügen (etwa indem sie die Tatsache, daß sie die Zufallsbits r kennen ausnutzen) um M mit höherer Wahrscheinlichkeit zum Akzeptieren zu bringen. Das Ereignis, daß M aufgrund von Betrugsversuchen von P_1 akzeptiert, läßt sich aufteilen in:

1. P_1 betrügt $<$ $(l/8)$ mal
2. P_1 betrügt $\geq$ $(l/8)$ mal

Natürlich versucht P_1 so wenig wie möglich zu betrügen, um die Wahrscheinlichkeit, daß der Betrugsversuch aufgedeckt wird, zu senken. Im Erwartungswert akzeptiert A(x) $\frac{8tn}{3}$ mal. In diesem Fall muß P_1 mehr als $(l/8)$ mal erfolgreich betrügen. Falls A(x) öfters als 3tn mal akzeptiert reicht es, weniger als $(l/8)$ mal zu betrügen.

$$\begin{aligned}\mathbf{Pr}[M(x) = \text{akz.}] \;&=\; \overbrace{\mathbf{Pr}\left[M(x) \text{ akz.} \wedge P_1 \text{ betrügt} < \frac{l}{8} \text{ erfolgreich}\right]}^{1} \\ &+\; \underbrace{\mathbf{Pr}\left[M(x) \text{ akz.} \wedge P_1 \text{ betrügt} \geq \frac{l}{8} \text{ erfolgreich}\right]}_{2}\end{aligned}$$

Es gilt offenbar :

$$\underbrace{[\text{M(x) akz.} \wedge P_1 \text{ betrügt} \leq (l/8) \text{ erfolgreich}]}_{A}$$

$$\Rightarrow \underbrace{[\text{mehr als(3/8) - mal akz. A(x), ohne daß } P_1 \text{ betrügt.}]}_{B}$$

Wegen A $\subseteq$ B gilt : Pr(A) $\leq$ Pr(B).

Zu (1) :

Berechnung der Wahrscheinlichkeit, daß A(x) mehr als $\frac{3l}{8}$ mal akzeptiert mit der *Chernoff-Bound* : Zunächst berechnen wir die Abweichung δ.

$$\begin{aligned} X \;>\; (1+\delta)\cdot\frac{l}{3} \;&=\; \frac{3l}{8} \\ 1 \;+\; \delta \;&=\; \frac{9}{8} \\ \delta \;&=\; \frac{1}{8}\end{aligned}$$

$$\Pr[X > (1+\delta)E(X)] \;<\; e^{-0{,}386E(X)\delta^2} \;\leq e^{-\Theta(n)}$$

Das heißt, nur mit exponentiell kleiner Wahrscheinlichkeit akzeptieren mehr als $(3l/8)$ Iterationen von A(x), falls n groß genug ist.

Zu (2) :

Bei einem Betrugsversuch muß P_1 mind. eine der t-vielen Antworten geändert haben. Mit einer Wahrscheinlichkeit von mind. $(1/t)$ wählt der Verifier eine Frage aus, deren Antwort verändert wurde. D.h. mit mindestens der Wahrscheinlichkeit $(1/t)$ merkt der Verifier den Betrug in einer Runde. Die Prover müssen mehr als tn Betrugsversuche machen. Die Wahrscheinlichkeit, daß sie mehr als tn-mal erfolgreich (also unbemerkt) betrügen können, ist somit $(1-\frac{1}{t})^{tn}$. Also wird mit Wahrscheinlichkeit

$$1 \ -(1-\frac{1}{t})^{tn} \geq \ 1-2^{-O(n)}$$

ein Betrugsversuch bemerkt, für genügend große n.

$$\begin{aligned}
&(2)\ \Pr[\mathrm{M(x)\ akz.}\ \wedge\ P_1\text{betrügt}\ >\ (l/8)\ \text{erfolgreich}]\\
&\leq \Pr[P_1\ \text{betrügt}\ >\ (l/8)\ \text{mal erfolgreich}]\\
&\leq 2^{-O(n)}
\end{aligned}$$

□

Literaturverzeichnis

[BDB92] M.V.D. Burmester, Y.G. Desmedt, T. Beth: Efficient zero-knowledge identification schemes for smart cards. In: *Computer Journal 35*, Seite 21-29, 1992

[BDG88] J. L. Balcázar, J. Díaz, J. Gabarró: *Structural Complexity I.* Springer-Verlag, 1988

[BDG90] J. L. Balcázar, J. Díaz, J. Gabarró: *Structural Complexity II.* Springer-Verlag, 1990

[BGKW88] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, Avi Wigderson : Multi-Prover Interactive Proff Systems : Removing Intractability Assumptions In: *Proceedings of the 20. STOC*, ACM, Seite 113-131, 1988

[Blu86] M. Blum: How to prove a theorem, so no one else can claim it. *Proceedings of the International Congress of Mathematicians Berkeley, CA* . Seite 1444-1451, 1986

[BOGG$^+$88] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, P. Rogaway: Everything provable is provable in zero-knowledge. In: *Proc. CRYPTO 88* Seite 37-56 , Springer Verlag, 1988. LNCS Nr. 435

[BSMP90] M. Blum, A. De Santis, S. Micali, G. Persiano: Non-interactive zero-knowledge, *MIT/LCS/TM-430.b*, 1990

[BSW95] Albrecht Beutelspacher, Jörg Schwenk, Klaus-Dieter Wolfenstetter : *Moderne Verfahren der Kryptographie.* Vieweg Mathematik, 1995

[CLR90] T. H. Cormen, C. E. Leiserson, R. L. Rivest: *Introduction to Algorithms*, MIT Press, 1990

[For91] L. Fortnow: Review of Goldwasser, Micali and Rackoff, " The Knowledge Complexity of Interactive Proof Systems„ , Goldreich, Micali and Wigderson, "Proofs that Release Minimum Knowledge„ and Goldreich, "Randomness, Interactive Proofs and Zero-Knowledge„. *Journal of Symbolic Logic, 56(3)*, Seite 1092-1094, 1991

[FS87] A. Fiat, A. Shamir: How to Prove Yourself - Practical Solutions to Identification and Signature Problems - In: *Proc. CRYPTO 86*, Seite 186-194, Springer Verlag 1987, LNCS NR. 263

[FFS88] U. Feige, A. Fiat, A. Shamir: Zero-knowledge proofs of identity. In: *Journal of Cryptology 1(2)*. Seite 77-94, 1988

[FS90] U. Feige, A. Shamir: Zero knowledge proofs of knowledge in two rounds. In: G. Brassard, editor, *Proc. CRYPTO 89*, Seite 526-545, Springer Verlag 1990. LNCS Nr. 435

[FS90a] U. Feige, A. Shamir: Witness indistinguishability and witness hiding protocols In: *Proc. 22nd ACM STOC*, 1990, Seite 416-426

[GJ79] M.R. Garey, D.S. Johnson: *Computers and Intractability – A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979

[GK90] O. Goldreich, H.Krawczyk: On the composition of zero-knowledge proof systems, *17th ICALP Proceedings, Lectures Notes in Computer Science, Vol. 443* Springer Verlag, New York, 1990, Seite 268-282.

[GK92] Oded Goldreich, Hugo Krawczyk: Sparse pseudorandom distributions. *Random Structures and Algorithms 3(2)*. Seite 163-174, 1992

[GK96] O. Goldreich, H. Krawczyk: On the composition of zero-knowledge proof system. In: *SIAM J. Comput. Vol 25 Nr. 1*, Seite 169-192, 1996

[GMR85] Shafi Goldwasser, Silvio Micali, Charles Rackoff: The knowledge complexity of interactive proof systems. In *Proc. 27th Annual Symposium on Foundations of Computer Science*. Seite 291-304, 1985

[GMR89] Shafi Goldwasser, Silvio Micali, Charles Rackoff: The knowledge complexity of interactive proof systems. In *SIAM J. Comput. Vol 18 Nr. 1*, Seite 186-208, 1989

[GMS87] O. Goldreich, Y. Mansour, M. Sipser: Interactive proof systems: Provers that never fail und random selection. In: *Proc of the 28th Annual IEEE Symposium on Foundations of Computer Science*, Seite 449-461, 1987.

[GMW91] O. Goldreich, S. Micali, A. Wigderson: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM 38(1)*, Seite 691-729, 1991

[GO94] O. Goldreich, Y.Oren: Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology 7(1)*, Seite 1-32, 1994

[Gol88] O. Goldreich: Randomness, Interaction, Proofs and Zero-Knowledge. *The Universal Turing Maschine: A Half-Century Survey*. Seite 377-406, Oxford University Press, 1988

[Gol89] S. Goldwasser: Interactive Proof Systems. In: *Proc. of Symposia in Applied Mathematics Vol 38*, 1989

[Gol95] O. Goldreich: *Foundations of Cryptography* zu finden unter: (http://www.eccc.uni-trier.de), 1995

[HU79] J. Hopcroft, J. Ullman: *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979

[Kar94] Christoph Karg: *Strukturfragen im Umfeld der Durchschnittskomplexität.* Diplomarbeit an der Universität Ulm, 1994

[Keh96] Guido Kehrle: *Die Bedeutung von kryptographischen Falltürfunktionen für Algorithmisches Lernen.* Diplomarbeit an der Universität Ulm, 1996

[Kil89] J. Kilian: *Uses of Randomness In Algorithms and Protocols* ACM Distinguished Dissertation 1989.

[Kob94] Johannes Köbler: *Vorlesungsskript Kryptographie.* Universität Ulm 1994.

[Kop95] Helmut Kopka: *LaTeX- Einführung Band 1.* Addison-Wesley, 1995

[KST93] J. Köbler, U. Schöning, J. Toràn: *The Graph Isomorphism Problem - It's Structural Complexity* -. Script der Universität Ulm, 1993

[Lan88] S. Landau: Zero knowledge and the Department of Defense. In: *Notices of the American Mathematical Society 35*, Seite 5-12, 1988

[MR95] Rajeev Motwani, Prabhakar Raghavan:*Randomized Algorithms.* Cambridge, 1995.

[Pau78] W. Paul: *Komplexitätstheorie.* Teubner Verlag, 1978

[QQQ^+90] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michael Quisquater, Louis Guillou, Marie Annick, Gaiid Guillou, Anna Guillou, G. Guillou, S. Guillou, T. Berson: How to explain zero-knowledge protocols to your children. In G. Brassard, editor, *Proc. CRYPTO 89*, Seite 628-631, Springer Verlag 1990. LNCS Nr. 435.

[Sch92] Uwe Schöning: *Vorlesungsskript Informatik IV*, Universität Ulm, 1992

[Sch95] Uwe Schöning: *Perlen der Theoretischen Informatik.* B· I · Wissenschaftsverlag, 1995

[Sim88] D. Simon: *Issues in the definition of zero-knowledge.* M.Sc. Thesis, University of Toronto, 1988.

[Sti95] Douglas R. Stinson: *Cryptography - Theory and Practice.* CRC Press, 1995

[TW87] M. Tompa, H. Woll: Random self-reducibility and zero-knowledge interactive proofs of possession of information, In: *Proc. 28th IEEE Symp. on FOCS*, 1987, Seite 472-482.

[Wut96] Andreas Wutzke: *Multi-Prover Beweissysteme*, Ausarbeitung zum Seminar Interaktive Beweissysteme SS96, Universität Ulm

Erklärung

Hiermit erkläre ich, daß ich diese Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den 19. September 1996

. . . .

Andreas Wutzke, Mat.Nr. 0235713