Zero-Knowledge Arguments

Diplomarbeit von

Thomas Schwarze

angefertigt bei

Prof. Dr. Rutger Verbeek

Fachbereich Informatik
Theoretische Informatik II
FernUniversität -Gesamthochschule- Hagen

7. September 2003

Zero–Knowledge Proofs sind ein wesentlicher Bestandteil der modernen Kryptographie. Zur Verringerung kryptographischer Annahmen und Voraussetzungen werden Zero–Knowledge Arguments in der Praxis immer stärker erforscht und eingesetzt. Die Sicherheit der Zero–Knowledge Arguments ist dabei nur für die Nichtexistenz unendlich schneller Maschinen gewährleistet. Für die Zero–Knowledge Proofs existieren bereits einige deutschsprachige Arbeiten. Im Bereich der Zero–Knowledge Arguments fehlen diese jedoch bislang, so dass Interessierte der Einstieg in dieses Thema erschwert wird. Diese Lücke soll mit der vorliegenden Diplomarbeit geschlossen werden.

Inhaltsverzeichnis

1 Einleitung

1.1 Überblick

Interaktive Beweissysteme, bei dem eine Partei Alice einer anderen Partei Bob etwas beweist (z.B. dass sie Kenntnis von der Knotenfärbung eines 3-färbbaren Graphen hat oder allgemeiner die Zugehörigkeit eines Wertes x zu einer Sprache L), sind seit längerer Zeit in der theoretischen Informatik bekannt. So veröffentlichte Babai 1985 in [Bab85] ein heute noch als Grundlage verwendetes System, welches als Arthur–Merlin Game mit Merlin als beweisende und Artur als überprüfende Patei bekannt wurde.

Ebenfalls 1985 wurde von Goldwasser, Micali und Rackow in [GMR85] ein interaktives Beweissystem vorgestellt, bei dem die überprüfende Partei keine Informationen über die Tatsache des Beweises hinaus erhält. Diese Systeme wurden nach einer weiteren formalen Definition von Goldreich, Micali und Wigderson 1986 in [GMW86] Zero–Knowledge Proofs genannt und hatten in der Folgezeit einen bis heute ungebremsten Anstieg der Forschungen zur Folge. Die moderne, computergestützte Kryptographie hat davon erheblich profitiert.

Die vorliegende Diplomarbeit beschäftigt sich mit einem Teilgebiet der Zero–Knowledge Proofs, den Zero–Knowledge Arguments. Dabei handelt es sich um eine leichte Abschwächung der Sicherheit, die sich in der praktischen Anwendung jedoch nicht als Sicherheitslücke auswirkt. Da interaktive Beweissysteme mittels Kommunikation mindestens zweier Teilnehmer durchgeführt werden, besteht der Ansatz der Zero–Knowledge Arguments darin, die notwendige Komplexität des Kommunikationsprotokolls soweit wie möglich zu verringern, ohne der Sicherheit eines der beteiligten Teilnehmer zu schaden. Beim Zero–Knowledge Argument wird die Sicherheit der überprüfenden Partei darüber, dass die beweisende Partei nicht unbemerkt lügen oder täuschen kann, auf die Dauer der Kommunikation beschränkt. Gäbe es unendlich schnelle Computer als Beweiser, dann wäre die Sicherheit nicht mehr gewährleistet, was in der praktischen Anwendung nicht vorkommt.

1.2 Motivation

Da es eine fast schon unüberschaubare Anzahl von Veröffentlichungen über Zero–Knowledge Arguments sowie entsprechende Konstruktionen und Entwürfe gibt, musste eine Grenze gefunden werden, welche Verfahren vorgestellt und in welcher Tiefe diese erläutert werden.

Die Entscheidung fiel zu Gunsten eines sehr detaillierten, tiefgehenden Einstiegs, um ein paar ausgewählte Vertreter vorstellen und deren Technik exakt beschreiben zu können. Dieses Vorgehen ging zu Lasten eines generellen Überblicks über eine Vielzahl

von Protokollen. Die Motivation dafür entstand aus dem Wunsch, anhand ausgesuchter Beispiele die Technik der Konstruktion von Zero–Knowledge Arguments sowie die damit verbundenen Probleme und Lösungsansätze vorzustellen. Mit diesem Wissen ist es leichter, andere Veröffentlichungen zu finden, zu verstehen und zu verarbeiten. Mit einer bloß überblickhaften Vorstellung „aller“ Verfahren wäre die Vermittlung des tieferen Verständnisses nicht möglich gewesen.

Die wissenschaftliche Literatur zu diesem Thema wird fast ausschließlich in englischer Sprache veröffentlicht. Zum besseren Verständnis beim Einstieg in die Problematik der Zero–Knowledge Arguments werden in der vorliegenden Diplomarbeit soweit vorhanden deutsche Begriffe gewählt, die den Sinn der Fachausdrücke wiedergeben. Wie in weiten Bereichen der Informationstechnik existieren aber in einigen Fällen keine einprägsamen deutschen Entsprechungen, weshalb in diesen speziellen Fällen der weltweit übliche englische Fachausdruck verwendet wird. Ich hoffe, dass ich dabei die akzeptable Balance zwischen Verständlichkeit und Notwendigkeit im Umgang mit den Begriffen getroffen habe. Die jeweilige Entscheidung ist dabei subjektiv ausgefallen. Im Interesse eines „vernünftigen“ Leseflusses wurde aber versucht, sinnentsprechende deutsche Begriffe zu verwenden, wenn dieses möglich war bzw. sogar geboten erschien, wenn angloamerikanische Begriffe Verwendung fanden, deren tieferer Sinn sich erst nach eingehender Kenntnis der Fachsprache eröffnet.

1.3 Aufbau der Arbeit

Nach dieser Einleitung, die mit ein paar über den Inhalt dieser Arbeit hinausgehenden Literaturhinweisen endet, erfolgen in **Abschnitt 2** die Beschreibungen und Definitionen der verwendeten Grundlagen. Dabei wurde der Rahmen absichtlich etwas weiter gesteckt und unter anderem durch einige mathematischen Grundlagen der Zahlentheorie ergänzt, um so zum besseren Verständnis der späteren Algorithmen beizutragen.

In **Abschnitt 3** erfolgt der Einstieg in die Zero–Knowledge Beweissysteme, indem dort die Zero–Knowledge Proofs beschrieben werden. Dazu werden ausgesuchte Besonderheiten dieser Systeme vorgestellt, deren Kenntnis und Verständis im weiteren Verlauf notwendig ist.

In den daran anschließenden **Abschnitten 4** bis **7** werden die eigentlichen Zero–Knowledge Arguments entwickelt, wobei sich im **Abschnitt 4** die grundlegende Beschreibung der Zero–Knowledge Arguments im Unterschied zu den Zero–Knowledge Proofs befindet. In den **Abschnitten 5**, **6** und **7** werden konkrete interaktive Systeme und deren Protokolle, Algorithmen und Beweise vorgestellt.

1.4 Überblick der verwendeten Themen der Kryptographie

Hier soll ein kurzer Überblick über einige Arbeiten gegeben werden, die historisch gesehen von einiger Bedeutung sind, soweit die veröffentlichten Techniken in dieser Arbeit Verwendung finden. Es handelt sich dabei um keine abschließende Aufzählung aller Forschungsarbeiten.

Zero–Knowledge Proof. Diese Systeme wurden 1985 erstmals von Goldwasser, Micali und Rackoff in [GMR85] veröffentlicht. Goldreich, Micali und Widgerson zeigten in [GMW86], dass derartige Beweissysteme für jede Sprache in $\mathcal{NP}$ einen Zero–Knowledge Proof liefern können. Die ersten Protokolle, welche einen Zero–Knowledge Proof in *konstanter Rundenzahl* durchführten, wurden von Feige und Shamir in [FS90b], Brassard, Crépeau und Yund in [BCY89] und von Goldreich und Kahn in [GK96a] vorgestellt.

Black-Box Simulator. Die bisherigen Arbeiten verwendeten zum Beweis der Zero–Knowledge Eigenschaft einen sogenannten *Black-Box* Simulator, was erstamls von Goldreich und Krawczyk [GK96b] bemängelt wurde. Das erste Protokoll, welches keinen Black-Box Simulator verwendete, war die Arbeit von Hada und Tanaka in [HT98]. Einen anderen Weg mit praktikablen Annahmen schlug Barak in [Bar03] ein, um ein Verfahren mit einem nicht Black-Box Simulator zu erzeugen.

Bit–Commitment–Scheme Um bei theoriebasierten Überlegungen nicht von der Existenz konkreter z.B. algebraischer Verfahren abhängig zu sein, wird eine Abstraktion existierender oder vermuteter Verfahren vorgenommen. Aufbauend auf Arbeiten u.a. von Blum [Blu81], Chaum [Cha87] aber auch [GMR85, GMW86] entwickelten Brassard, Chaum und Crépeau [BCC88] den Begriff des Bit–Commitments. Formal definiert und beschrieben wurden Bit–Commitment–Schemes unter anderem von Meyer in [Mey92], Naor in [Nao91] und Goldreich in [Gol01]. Darüber hinaus gibt es immer wieder Veröffentlichungen neuer Ansätze zu Bit–Commitment–Schemes, die gerade für den Einsatz in Zero–Knowledge Systemen von Interesse sind, so z.B. von Halevi und Micali in [HM96].

Witness Indistinguishing. Die Technik der Zeugnis–ununterscheidbarkeit (engl. *witness indistinguishing und witness hiding*) wurde von Feige und Shamir in [FS90a] vorgestellt. Sie ermöglicht eine neue, eventuell schwächere aber in vielen Anwendungsfällen ausreichende Formen des Zero–Knowledge Proofs zu konstruieren.

Zero–Knowledge Proof of Knowledge. Das Grundkonzept dieser interaktiven Beweissysteme stammt aus [GMR85] und wurde von Feige, Fiat und Shamir in [FFS87] sowie von Tompa und Woll in [TW87] formal definiert. Von Bellare und Goldreich wurden in [BG92] einige Lücken dieser Definitionen geschlossen.

Zero–Knowledge Argument In der Literatur auch als Computationally Sound (CS) Proof bezeichnet. Zero–Knowledge Arguments wurden von Brassard, Chaum und Crépeau in [BCC88] vorgestellt, wobei ein perfektes Zero–Knowledge Verfahren mit Hilfe des Faktorisierungsproblems entwickelt wurde. Naor, Ostrovsky, Venkatesan und Yung zeigten in [NOVY92] einen perfekten Zero–Knowledge Algorithmus, der nur noch auf der Existenz beliebiger Einweg–Permutationen beruht.

1.5 Literaturhinweise

Als grundsätzlicher Einstieg in die Thematik der modernen Kryptographie unter Berücksichtigung der interaktiven Beweissysteme werden hier einige Bücher, Artikel und sonstige Veröffentlichungen vorgestellt. Mit Ausnahme eines Artikels befassen sich alle hier genannten Werke nicht nur mit dem Spezialgebiet der Zero–Knowledge Arguments, sondern sind als Einstieg, Nachschlagewerke oder hilfreiche Literatur zu verstehen, um die Zero–Knowledge Arguments im Gesamtkontext der modernen Kryptographie und theoretischen Informatik zu verstehen bzw. die notwendigen Grundlagen dafür erarbeiten zu können.

- Bellare und Goldwasser [BG01b]: Ein sehr umfassendes Vorlesungsskript über die Kryptographie.
- Goldreich [Gol99]: Erstes Buch von Goldreich über die moderne Kryptographie.
- Goldreich [Gol01]: Standardwerk von Goldreich über Grundlagen der modernen Kryptographie. Es ist eines der am meisten zitierten Werke und von der Wissenschaft als das Grundlagenwerk überhaupt auf diesem Gebiet anzusehen.
- Goldreich [Gol02]: Sehr interessante Abhandlung über das gesamte Gebiet der Zero–Knowledge Systeme als Rückschau über die Forschungsergebnisse der letzten 20 Jahre.
- Mattern [Mat01]: Teil einer deutschsprachigen Vorlesung über Sicherheit in verteilten Systemen. Anschaulicher und leicht verständlicher Überblick über die Problematik.
- Menezes, van Oorschot und Vanstone [MvV01]: Sehr verständliches Buch über die angewandte Kryptographie. Zu Beginn werden sogar die mathematischen Grundlagen erklärt, so dass dieses Buch als Nachschlagewerk sehr empfehlenswert ist.
- A. Pfitzmann [Pfi00]: Sehr umfangreiches, deutschsprachiges Vorlesungsmaterial zur Kryptographie und Datensicherheit.

- B. Pfitzmann [Pfi98]: Umfangreiche, deutschsprachige Folien zur Vorlesung über Sicherheit in praktischer Informatik mit dem Schwerpunkt kryptographischer Systeme.
- B. Pfitzmann [Pfi99]: Ein weiteres, englischsprachiges Vorlesungsskript über „Higher cryptographic protocols“.
- Quisquater, Guillou und Berson [QGB90]: „How to explain zero-knowledge protocols to your children“ ist eine nette nicht-digitale und nicht-formale Geschichte über das Prinzip des Zero–Knowledge Proofs und dessen Nachweis.
- Stammnitz [Sta02]: Kurzer, deutschsprachiger Überblick über einige mathematische Grundlagen, die in Verschlüsselungsverfahren Anwendung finden.

2 Definitionen

2.1 Grundlegendes

Für die Diskussion der Zero–Knowledge Arguments werden in diesem Kapitel die grundlegenden Definitionen und Vereinbarungen dargelegt.

Diese Arbeit beschäftigt sich mit Beweissystemen, in denen von einer Partei eine Behauptung aufgestellt wird, z.B. dass sie im Besitz eines Wortes w aus der Sprache L ist, die von einer anderen Partei akzeptiert oder als falsch erkannt wird. Die Partei, die die Behauptung aufstellt und den Beweis antritt, wird im Nachfolgenden *Prover* (P) genannt. Die Partei, der die Behauptung bewiesen wird bzw. diese Behauptung prüft, wird als *Verifier* (V) bezeichnet. Häufig wird in der Literatur der Prover auch A wie *Alice* und der Verifier B wie *Bob* genannt. Bei der Untersuchung einer Maschine wird diese oftmals mit allen anderen, eventuell täuschenden bzw. von dem vorgegebenen Protkoll abweichenden Maschinen verglichen. Diese werden in der Diskussion mit einem * versehen, bei Untersuchung von P entsprechend P^*.[1]

Die Anzahl der Eingaben einer Maschine A wird wie folgt beschrieben. Bezeichne $|x|$ die Länge des Eingabewertes x und $t_A : \mathbb{N} \to \mathbb{N}$, $t_A(|x|)$ die Anzahl der Schritte der Maschine A, dann heißt A polynomiell beschränkt in Bezug auf die Länge der Eingabe x oder einfach nur polynomiell, wenn es ein Polynom $p_A(|x|)$ gibt, so dass für jedes x aus dem Definitionsbereich von A gilt: $t_A(|x|) \leq p_A(|x|)$. Existiert kein entsprechendes Polynom, dann heißt A polynomiell nicht beschränkt oder einfach nicht-polynomiell.

Wahrscheinlichkeit

Im weiteren Verlauf spielt die zufällige Wahl eines Wertes eine besondere Rolle, ohne dass hier das Problem diskutiert wird, wie von einer Maschine ein echter Zufall erzeugt bzw. genutzt werden kann. Diesbezüglich wird auf die Literatur z.B. von Håstad, Impagliazzo, Levin und Luby [HILL95], Goldreich [Gol99], Babai [Bab85] oder Naor [Nao91] verwiesen.

Sei S eine Wahrscheinlichkeitsverteilung, dann wird mit $x \in_R S$ ein zufällig gewähltes Element aus S bezeichnet, so dass x gleichverteilt über der Menge S ist.

Für den Wahrscheinlichkeitsraum S bezeichnet

$$\Pr[p(x) : x \in_R S]$$

die Wahrscheinlichkeit, dass das Prädikat $p(x)$ wahr ist bei der Zuordnung von $x \in_R S$. Ist die Zuordnung der Variablen x aus dem Zusammenhang ersichtlich, wird manchmal nur $\Pr[p(x)]$ geschrieben.

[1] In der Literatur werden diese Maschinen auch häufig mit $\widehat{P}$ oder auch $\overline{P}$ bezeichnet.

Relation und Zeugnis

Sei $R \subseteq \{0,1\}^* \times \{0,1\}^*$ eine binäre Relation, dann ist $L(R) \stackrel{\text{def}}{=} \{x \mid \exists y; (x,y) \in R\}$. Wenn $x \in L(R)$ gilt und y eine Zeichenkette mit der Eigenschaft $(x,y) \in R$ ist, dann heißt y ein *Zeugnis* für $x \in L(R)$. Die Menge der Zeugnisse für x wird mit $R(x)$ bezeichnet, d.h. $R(x) = \{y \mid (x,y) \in R\}$. Bezeichne $T : \mathbb{N} \to \mathbb{N}$ eine beliebige Funktion. Für eine Sprache L gilt $L \in \mathcal{N}time(T(n))$, wenn eine Relation R mit $L = L(R)$ sowie eine Turing Maschine M existieren, so dass M bei Eingabe von (x,y) höchstens $T(|x|)$ Schritte durchläuft und eine „1“ ausgibt, wenn $(x,y) \in R$ gilt, und „0“ sonst.

2.2 Knowledge und Zero–Knowledge

Eine erste Frage im Zusammenhang mit Zero–Knowledge Arguments betrifft die Bedeutung und den Umfang des Begriffs *Knowledge*. Seit der Veröffentlichung von Goldwasser, Micali und Rackoff [GMR85] sowie darauf aufbauend von Goldreich, Micali und Widgerson [GMW86] wird mit diesem Begriff ein dynamischer Prozess in Beweisverfahren verbunden, der nur die Kommunikation zwischen Teilnehmern, hier Prover und Verifier genannt, betrifft. Im Gegensatz dazu stehen statische Beweisverfahren z.B. in mathematischen Beweisen, die für sich alleine Gültigkeit haben.

Mit „Knowledge“ werden hier nur solche Kommunikationsinhalte in den zu betrachtenden Beweisen bezeichnet, die seitens des Provers über die Korrektheit der behaupteten Tatsache hinaus Informationen offenbaren.

Der Begriff „Zero–Knowledge“ ist dabei die (relative) Sicherheit dafür, dass ein Verifier mit keinem (berechenbaren) Verfahren und mit keiner Täuschungshandlung weitere Informationen über die Behauptung hinaus erlangen kann. Einschränkend sind mit diesen Informationen jedoch nur solche gemeint, die von dem Verifier nicht selbst gewonnen bzw. berechnet werden können, für ihn also fremd und (fast) unmöglich allein zu erlangen sind. Die Idee, die hinter der Definition von Zero–Knowledge liegt, besteht darin, dass alles, was der Verifier aus einer Kommunikation lernen kann, auch dadurch erlangt werden kann, dass der Verifier lediglich seine Eingabe betrachtet.

Ein Beispiel aus [Gol01] soll dieses deutlich machen. Wenn Alice behauptet, sie wisse, was die Summe aus „$1+1$“ ist, könnte sich Bob dieses durch einen später noch zu klärenden Zero–Knowledge Beweis beweisen lassen. Aber selbst, wenn Alice Bob das Ergebnis direkt mitteilte, gewönne er keine neuen Informationen, da die übermittelte Information von Bob sehr einfach in polynomieller Zeit selbst ermittelt werden könnte. Somit sind in diesem Zusammenhang nur solche Informationen mit dem Begriff „Knowledge“ verbunden, die entweder überhaupt nicht berechenbar oder zumindest durch den Teilnehmer Bob nicht berechenbar sind.

Ein Teilbereich der Forschung befasst sich mit den Aspekten der *nichtinteraktiven Beweissystemen*. Auf diese wird in dieser Arbeit jedoch nicht eingegangen. Daraus folgt

eine weitere Notwendigkeit, dass mit Zero–Knowledge Verfahren nur solche gemeint sind, bei denen der Verifier dem Prover mindestens eine Information übergeben muss, auf die oder mit der der Prover reagieren muss.

2.3 Interaktive Beweissysteme

Die Zero–Knowledge Systeme bauen auf den in [GMR85] begründeten interaktiven Beweissystemen auf. Diese bestehen aus zwei probabilistischen Turing Maschinen, die zu interaktiven Turing Maschinen erweitert werden und zusammen ein interaktives System ergeben.

Nachfolgend werden diese aus den bekannten Turing Maschinen abgeleiteten Maschinen definiert. Dabei wird auf eine informationstheoretisch formale Darstellung der Turing Maschinen als 4- oder 5-Tupel mit Angabe des Bandalphabets Σ, der Konfigurationen usw. verzichtet. Für diese formale Beschreibung der Zero–Knowledge Proof Theorie wird z.B. auf die Arbeiten von Wellner [Wel90] oder Meyer [Mey92] verwiesen.

2.3.1 Probabilistische Turing Maschine

Die weitere Darstellung baut auf dem Modell der Turing Maschine, auch deterministische Turing Maschine genannt, auf.

Definition 2.1 (Probabilistische Turing Maschine)
Eine probabilistische Turing Maschine ist eine Turing Maschine mit einem zusätzlichen Nur-Leseband, genannt das Zufallsband*, welches o.B.d.A. links beschränkt und nach rechts unendlich ist sowie eine unendliche Folge von zufälligen Bits $\sigma \in \{0,1\}$ enthält. Nach jedem Lesezugriff wird der Lesekopf der Turing Maschine um eine Stelle nach rechts bewegt. Es ist das einzige Zufallsband der Turing Maschine.*

Der einem Lesezugriff auf das Zufallsband folgende Zustand ist abhängig davon, ob eine 0 oder eine 1 gelesen wurde. Der Eintritt eines Folgezustands nach einem Lesezugriff auf das Zufallsband aus einem Paar von Zuständen tritt mit der Wahrscheinlichkeit von genau $\frac{1}{2}$ ein. Somit ist nicht mehr eindeutig genau ein Folgezustand definiert, so dass auch die Ausgabe bzw. der Haltezustand der gesamten probabilistischen Turing Maschine bei Eingabe eines Wertes x nicht mehr eindeutig bestimmt ist.

Das Zufallsband kann dadurch ersetzt werden, dass die probabilistische Turing Maschine bei den Zuständen, die einem Zugriff auf das Zufallsband entsprechen, zufällig ein $\sigma \in_R \{0,1\}$ wählt. Der „Zugriff auf das Zufallsband“ wird auch Münzwurf und die Maschine *Münzwurf Maschine* genannt.

Sei A die probabilistische Turing Maschine und $x \in \{0,1\}^*$ die Eingabe. Dann wird mit

$$\Pr[A(x)]$$

die Ausgabe der probabilistischen Turing Maschine als Wahrscheinlichkeitsverteilung über alle x in Abhängigkeit der Münzwürfe angegeben.

2.3.2 Interaktive Turing Maschine

Die bisher aufgeführten probabilistischen Turing Maschinen werden um zwei weitere Bänder zur *interaktiven probabilistischen Turing Maschine* erweitert. Beide Bänder sind jeweils einseitig unendlich, wobei das *Sendeband* beschreibbar aber nicht löschbar und das *Empfangsband* nur lesbar ist. Die Bänder sind o.b.d.A. links beschränkt. Auf dem Empfangsband darf nur dann eine Rechtsbewegung erfolgen, wenn rechts mindestens ein Zeichen vorhanden ist. Auf dem Sendeband darf nur ein leeres Feld beschrieben werden.

2.3.3 Interaktive Systeme

Zwei interaktive probabilistische Turing Maschinen A und B werden zu einem gemeinsamen System zusammengeschlossen, so dass das Sendeband der einen das Empfangsband der anderen Maschine ist und vice versa.

Zur Bezeichnung der Kommunikation bzw. Interaktion zwischen den beiden Maschinen wurde in der Literatur der Begriff *Runde* (engl. round) und *Zug* (engl. move) eingeführt. In dieser Arbeit wird mit einem Zug *eine* vollständige und abgeschlossene *Interaktion* bzw. Nachricht zwischen Prover und Verifier bezeichnet. Eine Runde beschreibt zwei aufeinanderfolgende Züge also eine von A nach B und die darauffolgende Antwort von B nach A.[2]

Definition 2.2 (Interaktives System)
Zwei interaktive Turing Maschinen A und B ergeben zusammen ein interaktives System (A, B) wenn

1. *A und B das selbe Eingabeband teilen und*
2. *A's Sendeband das Empfangsband von B ist und umgekehrt.*

B startet die Kommunikation. Hält eine der beiden Maschinen A oder B, dann hält das interaktive System (A, B). Sei a_i der i-te String, den A auf sein Sendeband geschrieben hat und entsprechend b_i von B. Dann ist der Text der Kommunikation nach n Runden von (A, B) die Folge $\{b_1, a_1, \ldots, b_n, a_n\}$, wobei a_n leer ist, wenn B als erstes den Haltezustand erreicht hat.

[2] In der Literatur besteht keine Einigung über die Begriffe *Runde* und *Zug*, so dass die Gefahr einer Verwechselung besteht. Während einerseits in Arbeiten zwei Züge eine Runde beschreiben, wie in der vorliegenden Ausarbeitung, bezeichnen andere Autoren mit Runde nur eine Interaktion, also einen Zug.

Analog der probabilistischen Turing Maschine ist die Wahrscheinlichkeit jeder Berechnung, die von (A, B) bei Eingabe von x ausgeführt wird, abhängig von den Münzwürfen, die von den Maschinen A und B durchgeführt werden.

2.3.4 Interaktives Beweissystem

Von Goldwasser, Micali und Rackoff wurde in [GMR85] gezeigt, dass mit interaktiven Systemen Sprachen bewiesen werden können, so dass *interaktive Beweissysteme* entstehen. In diesen interaktiven Beweissystemen übernimmt A die Rolle der beweisenden Partei und wird daher mit Prover P bezeichnet. B kommt die Rolle der zu überzeugenden Partei, dem Prüfer, zu und wird Verifier V genannt.

Definition 2.3 (Interaktives Beweissystem)
Sei $L \subseteq \{0,1\}^*$ eine Sprache, P und V interaktive probabilistische Turing Maschinen, wobei V polynomiell–beschränkt ist, und sei (P, V) ein interaktives System, welches beim Akzeptieren einer Eingabe eine „1“ auf die Ausgabe schreibt. Dann ist (P, V) ein **interaktives Beweissystem für eine Sprache** *L, wenn gilt:*

- *Vollständigkeitsbedingung (engl. completeness):*

$$\forall k \in \mathbb{N}, \exists N \in \mathbb{N} : \forall x \in L, |x| \geq N : \Pr[(P, V)(x) = 1] \geq 1 - \frac{1}{|x|^k}$$

- *Korrektheits- oder Eindeutigkeitsbedingung (engl. soundness):*

  Für jede (möglicherweise täuschende) interaktive probabilistische Turing Maschine P^* gilt

$$\forall P^* : \forall k \in \mathbb{N}, \exists N \in \mathbb{N} : \forall x \notin L, |x| \geq N : \Pr[(P^*, V)(x) = 1] < \frac{1}{|x|^k}$$

Die erste Bedingung besagt somit, dass der Nachweis für $x \in L$ mit einer beliebig kleinen Fehlerwahrscheinlichkeit erbracht werden kann. Die zweite Bedingung sagt aus, dass es fast unmöglich ist, $x \in L$ zu beweisen, wenn $x \notin L$ gilt.

Anmerkung: Während im Regelfall die Vollständigkeitsbedingung relativ einfach nachweisbar ist, muss die Korrektheitsbedingung besonderes sorgfältig geprüft werden. Daher wurden von Micali und Reyzin Arbeiten veröffentlicht, die interaktive Beweissysteme nach dem Grad der Korrektheit unterteilen. Einzelheiten dazu können in [MR01b, Rey01] nachgelesen werden.

Definition 2.4 (Klasse $\mathcal{IP}$)
Sei (P, V) ein interaktives Beweissystem gemäß Definition 2.3, wobei V polynomiell zeitbeschränkt ist. Die Klasse $\mathcal{IP}$ (engl. Interactiv Proof) enthält alle Sprachen L, für die ein interaktives Beweissystem (P, V) existiert.

Definition 2.5 (Klasse $\mathcal{BPP}$)
Eine Sprache L wird von einer probabilistischen polynomiell–beschränkten Turing Maschine M akzeptiert, wenn gilt:

- $\forall x \in L : Pr[M(x) = 1] \geq \frac{2}{3}$
- $\forall x \notin L : Pr[M(x) = 0] \geq \frac{2}{3}$

Die Klasse $\mathcal{BPP}$ (engl. Bounded-Probability Polynomial-time) enthält alle Sprachen L, die durch eine probabilistische polynomiell–beschränkte Turing Maschine erkannt werden kann.

Die Wahl von $\frac{2}{3}$ ist dabei willkürlich. Es muss lediglich sichergestellt werden, dass die Wahrscheinlichkeit der Akzeptanz deutlich größer als $\frac{1}{2}$ ist. Somit erfüllt jede Konstante größer $\frac{1}{2}$ oder auch Funktion mit einer Funktionswert größer $\frac{1}{2}$ die Klassendefinition.

Anmerkung: In [GMR85] wurde erstmalig gezeigt, dass jede Sprache $L \in \mathcal{NP}$ ein interaktives Beweissystem hat, somit also $\mathcal{NP} \subseteq \mathcal{IP}$ gilt. Weiterhin kann in [Gol99] und [Gol01] nachgelesen werden, dass $\mathcal{NP} \cup \mathcal{BPP} \subseteq \mathcal{IP}$ gilt, wobei nicht bekannt ist, ob $\mathcal{BPP} \subseteq \mathcal{NP}$ gilt. Bekannt ist jedoch, dass jede Sprache $L \in \mathcal{PSPACE}$ ein interaktives Beweissystem hat, so dass $\mathcal{IP} = \mathcal{PSPACE}$ gilt.

Die Komplexität von interaktiven Systemen wird sowohl für den Zeit- als auch für den Rundenaufwand wie folgt angegeben.

Definition 2.6 (Zeitaufwand, Mächtigkeit des Provers)
Sei $T : \mathbb{N} \to \mathbb{N}$ eine Funktion, P eine interaktive probabilistische Turing Maschine. Mit $\mathcal{IP}_T$ wird die Menge der Sprachen L aus $\mathcal{IP}$ bezeichnet, in der für beliebige interaktive probabilistische Turing Maschinen V^* der Prover in dem interaktiven Beweissystem (P, V^*) für jede Eingabe $x \in L$ in $T(|x|)$ Schritten hält.

Definition 2.7 (Rundenaufwand)
Sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion, (P, V) ein interaktives Beweissystem L eine Sprache und $x \in L$ eine beliebige Eingabe. Dann wird mit $\mathcal{IP}[f]$ die Menge der Sprachen aus $\mathcal{IP}$ bezeichnet, für deren Beweis die Maschinen P und V höchstens $f(|x|)$ Runden benötigen.

Für die weitere Diskussion wird noch eine Möglichkeit benötigt, das Empfangs- und Zufallsband[3] einer einzelnen Maschine in einem interaktiven Beweissystem formal zu beschreiben. Dazu wird das für eine Maschine sichtbare Protokoll wie folgt definiert.

Definition 2.8 ($\mathbf{view_A}(\cdot)$)
Sei (A, B) ein interaktives Beweissystem. Dann bezeichnet $\mathbf{view}_A(A(\cdot), B(\cdot))$ oder kurz $\mathbf{view}_A^B(\cdot)$ eine Zufallsvariable, die den Inhalt des Empfangsbandes von A und des von A benutzten Teil des Zufallbandes der Kommunikation mit B enthält. Entsprechend gilt $view_B(\cdot)$ für die Sicht der Maschine B.

[3] Dabei werden nur die Münzwürfe des Zufallsbandes erfasst, die von der Maschine verarbeitet wurden.

Neben der Angabe des für die Maschine A sichtbaren Protokolls der Kommunikation wird die Ausgabe von A wie folgt definiert.

Definition 2.9 ($\mathbf{out_A}(\cdot)$)
Sei (A, B) ein interaktives Beweissystem. Dann bezeichnet $\mathbf{out}_A(A(\cdot), B(\cdot))$ oder kurz $\mathbf{out}_A^B(\cdot)$ eine Zufallsvariable, die die Ausgabe von A am Ende der Kommunikation enthält (rsp. $\mathbf{out}_B(\cdot)$ für die Maschine B).

Zur Beschreibung des Kommunikationsprotokolls zwischen den beiden Turing Maschinen eines interaktiven Beweissystems dient nachfolgende Definition.

Definition 2.10 ($\mathbf{transcript}_{(AB)}(\cdot)$)
Sei (A, B) ein interaktives Beweissystem. Dann bezeichnet $\mathbf{transcript}(A(\cdot), B(\cdot))$ oder auch $\mathbf{transcript}_{(AB)}(\cdot)$ alle Nachrichten, die zwischen A und B ausgetauscht wurden.

Daneben existiert zur Beschreibung einer Turing Maschine noch folgende Definition.

Definition 2.11 (desc(M))
Sei M eine Turing Maschine. Dann bezeichnet $\mathbf{desc}(M)$ die Beschreibung (engl. description) von M.

2.3.5 Interaktives Beweissystem mit Zusatzeingabe

Eine Erweiterung stellen interaktive Beweissysteme dar, die zusätzlich zu der gemeinsamen Eingabe x jeweils ein *Geheimnis* s_P und s_V als Eingabe erhalten. Von Chaum, Evertse und von de Graaf wurde in [CEv88] ein interaktives Beweissystem vorgestellt, bei dem sowohl P als auch V polynomiell–beschränkte Münzwurf Maschinen sind und $s_P =$ leer sowie im Allgemeinen $s_V \neq$ leer sind. Diese Grundlage wird weiter entwickelt zum interaktiven Beweissystem mit Zusatzeingabe.

Definition 2.12 (Interaktives Beweissystem mit Zusatzeingabe)
Sei (P, V) ein interaktives Beweissystem gemäß Definition 2.3, wobei P und V beide polynomiell beschränkt sind. (P, V) heißt **interaktives Beweissystem mit Zusatzeingabe***, wenn wenigstens P oder V neben dem gemeinsamen noch mindestens ein jeweils eigenes Eingabeband haben, so dass bei gemeinsamer Eingabe x jeweils $P(x, \cdot)$ bzw. $V(x, \cdot)$ berechnet wird.*

Man beachte im Unterschied zum interaktiven Beweissystem, dass hier auch der Prover P polynomiell beschränkt ist. Bei frei gewähltem, aber festem x werden die Maschinen auch mit $A_x(\cdot) = A(x, \cdot)$ bezeichnet.

2.4 Unterscheidbarkeit der Wahrscheinlichkeitsverteilung

In der weiteren Diskussion, vor allem aber für die Definition der Zero–Knowledge Proof-Systeme in Kapitel 3 ab Seite 30 und darauf aufbauend der Zero–Knowledge Arguments spielt die Ausgabe der an einem solchen System beteiligten Maschinen eine besondere Rolle. Da i.d.R. alle Teilnehmer eines Beweissystems probabilistische Turing Maschinen sind, kann die jeweilige Ausgabe nur über deren Wahrscheinlichkeitsverteilung für eine Eingabe x betrachtet werden. Zur Überprüfung der übermittelten Informationen wird der Unterschied eines Wahrscheinlichkeitsraums in noch zu präzisierenden Verfahren mit anderen Wahrscheinlichkeitsverteilungen verglichen.

In der Literatur z.B. bei Meyer [Mey92], Goldreich [Gol01] u.a. werden mehrere Unterscheidungen von Wahrscheinlichkeitsverteilungen beschrieben. Für die vorliegende Arbeit ist die Einteilung in zwei Grade jedoch ausreichend. Es handelt sich dabei um *perfekt ununterscheidbar* (engl. perfect indistinguishable) und *algorithmisch ununterscheidbar* (engl. computational indistinguishable).[4] Einfach ausgedrückt sind zwei Familien von Wahrscheinlichkeitsverteilungen perfekt ununterscheidbar, wenn sie identisch sind. Sie sind algorithmisch ununterscheidbar, wenn die Unterscheidbarkeit kleiner ist als eine Funktion, die schneller klein wird, als der Kehrwert jedes Polynoms. Diese Funktion wird als *vernachlässigbare Funktion* (engl. negligible function) oder auch μ-Funktion bezeichnet.

Definition 2.13 (vernachlässigbare Funktion)
Eine Funktion $\mu : \mathbb{N} \to [0,1]$ *heißt* vernachlässigbar*, wenn gilt*

$$\forall k \in \mathbb{N}, \exists n_k \in \mathbb{N}, \forall n \geq n_k : \mu(n) \leq \frac{1}{n^k}$$

Es existieren noch andere Definitionen der vernachlässigbaren Funktion, so z.B. von Bellare, Jakobsson und Yung [BJY97]. Wie von Bellare in [Bel02] gezeigt wurde, sind diese jedoch equivalent.

Es sei darauf hingewiesen, dass es für die kryptographische Untersuchung nicht ausreichend ist, wie in einigen Arbeiten (z.B. [Wel90, Mey92]) zu fordern, dass

$$\lim_{\substack{|x|\to\infty \\ x\in L}} \mu(x) = 0$$

gilt, da z.B. $f(x) = \frac{1}{x}$ dieses erfüllte. Anders als in der theoretischen Informatik wird in der theoretischen Kryptographie eine Bearbeitung mit endlichen Zahlen betrachtet, um eine Berechenbarkeit mit gegebenen Maschinen zu erhalten. Es ist daher erforderlich, dass eine Funktion die erforderliche Ununterscheidbarkeit schon bei relativ kleinen

[4] Weitere Unterscheidungsgrade sind die *statistische* und die *Schaltkreis*-Ununterscheidbarkeit, wobei für die vorliegende Arbeit ausreichend ist, die statistischen den perfekten Verfahren und die schaltkreisunterscheidbaren Verfahren den algorithmischen zuzuordnen.

Zahlen beschreibt. Durch die Wahl der μ-Funktion als z.B. eine exponentiell kleine Funktion, die für genügend große x schneller klein wird, als jedes Polynom, ist diese Forderung erfüllt.

Damit kann die Definition der Ununterscheidbarkeit erfolgen.

Definition 2.14 (Ununterscheidbarkeit)
Sei $L \subseteq \{0,1\}^*$ eine Sprache, $U = \{U_x\}_{x\in L}$ und $V = \{V_x\}_{x\in L}$ Familien von Wahrscheinlichkeitsverteilungen, eine für jedes $x \in L$.

1. *Die Familien U und V heißen* **berechenbar ununterscheidbar**, *wenn es eine vernachlässigbare Funktion $\mu(\cdot)$ gibt, so dass für alle probabilistischen polynomiell–beschränkten Turing Maschinen A gilt*

$$\left|Pr[\mathsf{A}(U_x) = 1] - \Pr[\mathsf{A}(V_x) = 1]\right| \leq \mu(|x|)$$

2. *Die Familien U und V heißen* **statistisch ununterscheidbar**, *wenn es eine vernachlässigbare Funktion $\mu(\cdot)$ gibt, so dass für alle (nicht nur polynomiell–beschränkten) Turing Maschinen B gilt*

$$\left|Pr[\mathsf{B}(U_x) = 1] - \Pr[\mathsf{B}(V_x) = 1]\right| \leq \mu(|x|)$$

3. *Die Familien U und V heißen* **perfekt ununterscheidbar** *oder auch identisch, wenn $U = V$ gilt.*

2.4.1 Pseudo Zufallsgeneratoren

Da in der Kryptographie polynomiell–beschränkte Maschinen eingesetzt werden, spielt die Zufälligkeit eine entscheidende Rolle. Mehr noch als in der theoretischen Kryptographie kommt in der modernen praktischen Kryptographie der Nutzung von maschinenerzeugbaren Zufallsfolgen eine der entscheidenden Bedeutungen zu. Dabei wird in der Literatur oftmals die Abkürzung PRG für *Pseudo Random Generator* genutzt. Diese können wie folgt definiert werden:

Definition 2.15 (Pseudo Zufallsgenerator PRG)
Seien $l, n \in \mathbb{N}$ mit $l < n$ und U_l rsp. U_n gleichverteilte Wahrscheinlichkeitsräume über die Mengen $\{0,1\}^l$ rsp. $\{0,1\}^n$. Eine deterministische polynomiell berechenbare Funktion $\mathsf{PRG} : \{0,1\}^l \to \{0,1\}^n$ heißt ***Pseudo Zufallsgenerator****, wenn die Zufallsvariable $\mathsf{PRG}(U_l)$ berechenbar ununterscheidbar von U_n ist.*

Es gibt weitere Definitionen für Pseudo Zufallsgeneratoren insbesondere für statistische und perfekte Ununterscheidbarkeit. Diese werden in dieser Arbeit jedoch nicht benötigt. Das grundlegende Prinzip, aus einem Ausgangswert einen neuen zufälligen

bzw. zufällig erscheinenden Wert zu erzeugen, der mindestens ein Bit länger als der Ausgangswert ist, ist unabhängig vom Grad der Unterscheidbarkeit. Die Elemente des Definitionsbereich von PRG werden dabei als *Saat* (engl. seed) bezeichnet. Die Realisierung von Pseudo Zufallsgeneratoren erfolgt in der Regel über Einweg–Funktionen, die in Abschnitt 2.7 ab Seite 18 beschrieben werden.

2.5 Bit–Commitment–Scheme

Eine wichtige Grundlage der modernen Kryptographie ist die Existenz von „schweren" Problemen. Darunter werden in Abhängigkeit der Zweckerfüllung mathematische oder informationstheoretische Verfahren verstanden, die gar nicht, nicht mit existierenden Maschinen oder nicht in einer überschaubaren Zeit berechnet werden können. Um bei theoriebasierten Überlegungen nicht von der Existenz konkreter z.B. algebraischer Verfahren abhängig zu sein, wird eine Abstraktion existierender oder vermuteter Verfahren vorgenommen.

Aufbauend auf Arbeiten u.a. von Blum [Blu81], Chaum [Cha87] aber auch [GMR85, GMW86] entwickelten Brassard, Chaum und Crépeau [BCC88] den Begriff des Bit–Commitments, wobei ein Bit–Commitment–Scheme das Basisverfahren für ein einzelnes Bit ist, die wiederholt ausgeführt einen String m mit $|m| > 1$ in einem Commitment–Scheme verarbeiten. Nachfolgend wird ein Überblick über (Bit-)Commitment–Schemes gegeben, soweit es hier notwendig ist. Weiterführende Ausführungen sowie Grundlagen zu Bit–Commitment–Schemes können u.a. bei Meyer [Mey92] (formaltheoretische Definitionen und Beschreibungen), Naor [Nao91] oder auch Goldreich [Gol01] nachgelesen werden. Darüber hinaus gibt es immer wieder Veröffentlichungen neuer Ansätze zu Bit–Commitment–Schemes, die gerade für den Einsatz in Zero–Knowledge Proofs und Zero–Knowledge Arguments von Interesse sind, so z.B. von Halevi und Micali [HM96].

Bit–Commitment–Schemes bestehen aus zwei Unterprotokollen, der *Festlegungsphase* (commit stage) und der *Öffnungsphase* (opening stage). In der Festlegungsphase wird ein Bit $\sigma \in \{0, 1\}$ von dem sendenden Teilnehmer S so festgelegt, dass es hinterher nicht mehr verändert werden kann, während σ für jeden anderen nicht sichtbar ist. Das veröffentlichte Paket wird dabei *Commitment* genannt. In der Öffnungsphase wird das festgelegt Bit σ dem empfangenen Teilnehmer E bekannt gegeben, wobei durch das Bit–Commitment–Scheme garantiert ist, dass nur das von S ursprünglich festgelegte Bit σ und nicht $1 - \sigma$ herauskommen kann.

Commitment–Schemes sind das digitale Anlogon zu versiegelten, undurchsichtigen Briefumschlägen, bei denen sichergestellt ist, dass die Inhalte wirklich von den Absendern stammen und nach dem Versiegeln nicht mehr verändert werden konnten.

Commitment–Schemes haben zwei wesentliche Eigenschaften:

- Eindeutigkeit (engl. binding property): Der Sender kann den Wert nach der Festlegungsphase nicht mehr ändern.
- Geheimhaltung (engl. hiding property): Der Empfänger lernt vor der Öffnungsphase nichts über den Wert.

Ausgehend von Brassard, Chaum und Crépeau werden Commitments auch manchmal als *blobs* bezeichnet [BCC88, BCY89].

Damit kann jetzt die allgemeine Definition des Bit–Commitment–Schemes erfolgen.

Definition 2.16 (Bit–Commitment–Scheme)
Sei (S, E) ein interaktives Beweissystem (S für Sender und E für Empfänger) und $\sigma \in \{0,1\}$ das nur S bekannte Bit. (S, E) ist ein **Bit–Commitment–Scheme**, *wenn gilt*

- *Geheimhaltung (engl. hiding):*

  Der Empfänger E kann vor der Öffnung ein 0–Commitment nicht von einem 1–Commitment unterscheiden, selbst wenn er von dem Protokoll abweicht. Sei E^* eine beliebige (möglicherweise täuschende) Turing Maschine. Dann sind nach der Festlegungsphase $\{\mathsf{view}_{E^*}((S(\sigma = 0), E^*)(n))\}_{n\in\mathbb{N}}$ und $\{\mathsf{view}_{E^*}((S(\sigma = 1), E^*)(n))\}_{n\in\mathbb{N}}$ statistisch rsp. berechenbar ununterscheidbar.

- *Festlegung oder Eindeutigkeit (engl. binding):*

  Seien $\nu \in \{0,1\}$ das nur S bekannte Bit und $n \in \mathbb{N}$ beliebig. Nach der Festlegungsphase sei $\mathsf{view}_E((S(\nu), E)(n))$ ein mögliches ν-Commitment. Dann darf $\mathsf{view}_E((S(\nu), E)(n))$ berechenbar rsp. statistisch nicht gleichzeitig ein mögliches 0–Commitment und ein mögliches 1–Commitment sein.

Die gemeinsame Eingabe n kann dabei als ein beliebig gewählter Sicherheitsparameter angesehen werden.

Anmerkung: Alle nachfolgenden Protokolle von Bit–Commitment–Scheme sind entweder statistisch eindeutig oder statistisch geheim, aber nicht beides gleichzeitig, wie bei Goldreich [Gol01] oder Pfitzmann [Pfi99] nachgelesen werden kann. Intuitiv besteht der Grund darin, dass nach der Festlegungsphase entweder zwei Möglichkeiten zur Öffnung des Commitments bestehen, dann ist es nicht statistisch eindeutig, oder es existiert nur eine, dann ist es nicht statistisch geheim.

Um mittels eines Commitment–Schemes nicht nur ein einzelnes Bit sondern ganze Zeichenketten verschlüsseln zu können, ist es notwendig, das Bit–Commitment–Scheme mehrfach entweder sequenziell oder parallel auszuführen. Es wurde nachgewiesen (z.B. in [GMW86, Gol01]), dass die Geheimhaltung bei sequenzieller Ausführung aller Bit–Commitment–Scheme erhalten bleibt. Dem entgegen steht jedoch die Erkenntniss, dass dieses

im Allgemeinen bei der parallelen Ausführung nicht der Fall ist (z.B. [GK96b, BCY91]). Um die Anzahl der Interaktionen zu reduzieren, ist jedoch eine parallele Ausführung notwendig. Der Entwurf eines Commitment–Scheme, eventuell im Zusammenhang mit dem Protokoll, in dem das Commitment–Scheme eingesetzt wird, muss somit im Besonderen darauf gerichtet sein, die Geheimhaltung bei der parallelen Verarbeitung zu erhalten.

2.6 Schaltkreise (engl. Circuits)

Kryptographische Verfahren werden oftmals sowohl informationstheoretisch als auch verbal mit Hilfe von Schaltkreisen (engl. circuits) untersucht und beschrieben. Die Beschreibung der Schaltkreise ist zum Teil der Arbeit von Meyer [Mey92] entnommen. Schaltkreise sind eine weitere Möglichkeit, Berechnungen und Algorithmen zu modellieren. Im Gegensatz zur Turing Maschine, die Eingaben beliebiger Länger verarbeiten kann, ist ein Schaltkreis immer nur für eine feste Eingabelänge geeignet.

Definition 2.17 (Boolescher Schaltkreis)
Das Tripel $C = (G, \mathrm{In}, \Phi)$ *heißt deterministischer* **Boolescher Schaltkreis**, *wenn gilt:*

- $G = (V, E)$ *ist ein gerichteter azyklischer Graph mit einer endlichen Menge* $V = \{1, \ldots, n\}$ *von Knoten und dazugehörigen Kanten* $E \subseteq V \times V$.
- $\mathrm{In} \subseteq \mathrm{IN}(G) = \{v \in V | \mathrm{in}(v) = 0\}$ *mit* $\mathrm{in}(v) = |\{w|(w, v) \in E\}|$ *ist die Menge der Eingangsknoten.*
- $\Phi : V \backslash \mathrm{In} \rightarrow \{\vee, \wedge, \neg, 0, 1\}$ *ist eine Funktion, die jedem Knoten* $v \in V \backslash \mathrm{In}$ *eine Boolesche Funktion zuordnet. Dabei sind folgende Bedingungen erfüllt:*

$$\begin{aligned} \mathrm{in}(v) = 2 &\Rightarrow \Phi(v) \in \{\vee, \wedge\}, \\ \mathrm{in}(v) = 1 &\Rightarrow \Phi(v) = \neg, \\ \mathrm{in}(v) = 0 &\Rightarrow \Phi(v) \in \{0, 1\}. \end{aligned}$$

Eine Sprache $L \subseteq \{0,1\}^n$ wird von einem Schaltkreis C genau dann entschieden, wenn $x \in L \Leftrightarrow C(x) = 1$ gilt. Um Eingaben beliebiger Länge mit Schaltkreisen verarbeiten zu können, werden Familien von Schaltkreisen über einer abzählbaren Indexmenge untersucht. Sei $\{C_x\}_{x \in \{0,1\}^*}$ eine Familie von Schaltkreisen mit einem Ausgangsknoten. Dann heißt

$$L = \{x \in \{0,1\}^* | C_x(x) = 1\}$$

die von $\{C_x\}_{x \in \{0,1\}^*}$ entschiedene Sprache.

Definition 2.18 (Probabilistischer Schaltkreis)
Ein Tupel $C = (C', p)$ *heißt* **probabilistischer Schaltkreis**, *wenn gilt:*

- $C' = (G, \mathrm{In}, \Phi)$ *ist ein deterministischer Schaltkreis.*

- $\mathrm{In}_P \subseteq \mathrm{In}$ *ist die Menge der* probabilistischen *Eingangsknoten.*

  $\mathrm{In}_D = \mathrm{In} \backslash \mathrm{In}_P$ *ist die Menge der* deterministischen *Eingangsknoten von* C*.*

- $p : \mathrm{In}_P \to [0,1]$ *ist eine Funktion, die für jeden Knoten* $v \in \mathrm{In}_P$ *die Wahrscheinlichkeit* $p(v)$ *angibt, dass* v *den Wert 1 annimmt.*

In [Mey92] und [Wel90] sind weiterführende formale Darstellungen der Schaltkreise nachzulesen. Dort wird auch gezeigt, dass jede probabilistische Turing Maschine in eine Familie probabilistischer Schaltkreise überführt werden kann. Insbesondere wird in [Wel90] gezeigt, dass für zwei polynomiell–beschränkte Familien von Wahrscheinlichkeitsfunktion $U = \{U_x\}_{x \in L}$ und $V = \{V_x\}_{x \in L}$ folgende Beziehung gilt:

$$U, V \text{ identisch}$$
$$\Downarrow$$
$$U, V \text{ Schaltkreis - ununterscheidbar}$$
$$\Downarrow$$
$$U, V \text{ algorithmisch ununterscheidbar}$$

2.7 Einweg–Funktionen

Einweg–Funktionen (engl. one-way-function) sind einfach gesagt Funktionen, die leicht zu berechnen praktisch aber nicht zu invertieren sind. Die Einschränkung „praktisch“ ist dabei im Zusammenhang mit dem Einsatz innerhalb der Kryptographie zu sehen beim Einsatz genügend großer Zahlen. So kann z.B. die Funktion $y = f(x)$ eine Komplexität von $\mathrm{O}(n \log n)$ haben, die von $x = f^{-1}(y)$ jedoch $\mathrm{O}(2^n)$.

Einweg–Funktionen stellen dabei eines der bedeutendsten Primitive innerhalb der modernen Kryptographie dar. Dabei ist wichtig zu beachten, dass die gesamten Überlegungen bezüglich der Einweg–Funktionen auf $\mathcal{P} \neq \mathcal{NP}$ beruhen.

Es existiert noch kein mathematischer Beweis dafür, das es Einweg–Funktionen gibt. Trotzdem gibt es Funktionen wie die Multiplikation zweier großer Primzahlen mit dem Problem der Primfaktorzerlegung oder Restklassenringe mit dem Problem des diskreten Logarithmus,[5] die vermutete Einweg–Funktionen sind.

Die Angaben in diesem Abschnitt stammen unter anderem aus den Veröffentlichungen von Mattern [Mat01], Menezes, van Oorschot und Vanstone [MvV01] sowie insbesondere aus dem Buch von Goldreich [Gol01].

[5] Siehe dazu den Abschnitt 2.8 *Zahlentheorie* ab Seite 23.

Definition 2.19 (Einweg–Funktionen)
*Eine Funktion $f : \{0,1\}^* \to \{0,1\}^*$ wird **Einweg–Funktion** genannt, wenn die folgenden beiden Bedingungen erfüllt sind:*

1. *Einfach zu berechnen: Es existiert ein (deterministischer) polynomiell–beschränkter Algorithmus A, so dass dieser bei Eingabe von x $f(x)$ ausgibt ($\mathsf{A}(x) = f(x)$).*
2. *Schwer zu invertieren: Für jeden probabilistischen polynomiell–beschränkten Algorithmus A^*, alle genügend großen $n \in \mathbb{N}$ und alle $r, r' \in_R \{0,1\}^n$ gilt*

$$Pr[\mathsf{A}^*(f(r'), 1^n) \in f^{-1}(f(r))] < \mu(n)$$

mit μ als vernachlässigbarer Funktion.

Die Wahrscheinlichkeit der zweiten Bedingung wird dabei über alle n, r, r' und die Münzwürfe von A^* mit gleichverteilter Wahrscheinlichkeit berechnet.

Anmerkung: Es wird nicht gefordert, dass A^* das spezifische Urbild von $f(x)$ berechnet. Jedes Urbild, d.h. jedes Element aus $f^{-1}(f(x))$ ist ausreichend. Natürlich gilt $x = f^{-1}(f(x))$, falls f bijektiv ist, aber im Allgemeinen können andere Urbilder die Voraussetzung erfüllen.

Die zusätzliche Eingabe 1^n dient lediglich dem Zweck zu verhindern, dass eine Funktion nur deshalb als Einweg–Funktion gilt, weil deren Ausgabe so stark verkürzt wurde, dass der Invertierung keine Zeit verbleibt, die Ausgabe auszugeben. Sei z.B. $y = bin(x)$ die Funktion, um die binäre Darstellung von x auszugeben. Dann ist $|bin(x)| = \log_2 |x|$ und kein Algorithmus kann in Polynomialzeit bezüglich $|y|$ bin invertieren. Die Hilfseingabe 1^n verhindert, dass eine praktisch invertierbare Funktion als Einweg–Funktion bezeichnet wird.

Neben dieser Definition existieren noch weitere, die entweder auf die obige zurückgeführt werden können oder deren Unterschiede für die vorliegende Arbeit nicht relevant sind.

Von weiterer Bedeutung sind noch *injektive Einweg–Funktionen*, die in der Fachliteratur oft mit *1-1* oder *one-one* bezeichnet werden, da diese Funktionen in vielen Zero–Knowledge Protokollen Anwendung finden. Ist eine Einweg–Funktion sogar *bijektiv*, dann wird diese Funktion zur

Definition 2.20 (Einweg–Permutationen)
*Sei f eine Einweg–Funktion gemäß Definition 2.19 und bijektiv. Dann heißt f eine **Einweg–Permutation**.*

Die Bijektivität erscheint für eine Einweg–Funktion auf den ersten Blick sinnlos. Sie ist aber insbesondere dann von Wichtigkeit, wenn für die Invertierung eine Zusatzinformation vorhanden ist, die es erlaubt, f^{-1} effizient zu berechnen. Diese Umkehrfunktionen

sozusagen mit „Hintertür“ werden als *Trapdoor Funktion* bezeichnet.[6] Um die Trapdoor Funktionen genauer definieren zu können, werden Familien oder Kollektionen von Funktionen benötigt.

Definition 2.21 (Kollektion von Funktionen)
Eine Kollektion von Funktionen besteht aus einer endlichen Menge $\bar{I}$ von Indizees und einer dazugehörigen endlichen Menge von Funktionen $\{f_i\}_{i\in\bar{I}}$. D.h. für jedes $i \in \bar{I}$ existiert ein endlicher Wertebereich D_i, in der die Funktion f_i enthalten ist.

Weiterhin wird ein effizienter Algorithmus I benötigt, der nach Eingabe einer Zahl n zufällig einen Index i mit $|i| = poly(n)$ auswählt und damit eine Funktion f_i sowie den dazugehörigen Wertebereich D_i festlegt. Das führt zu folgender Definition.

Definition 2.22 (Kollektion von Einweg–Funktionen)
*Eine Kollektion von Funktionen $\{f_i : D_i \to \{0,1\}^*\}_{i\in\bar{I}}$ wird **Kollektion von Einweg–Funktionen** genannt, wenn drei probabilistische polynomiell–beschränkte Algorithmen I, D und F existieren, so dass die folgenden beiden Bedingungen erfüllt sind:*

1. *Leicht zu berechnen: Die Verteilung der Ausgabe von I bei Eingabe von 1^n ist eine Zufallsvariable über der Menge $\bar{I} \cap \{0,1\}^n$. Die Verteilung der Ausgabe von D bei Eingabe von $i \in \bar{I}$ ist eine Zufallsvariable über die Werte in D_i. Bei Eingabe von $i \in \bar{I}$ und $x \in D_i$ gibt F stets $f_i(x)$ aus.*

   Somit ist $D_i \subseteq \bigcup_{m\leq poly(|i|)}\{0,1\}^m$, wobei angenommen werden kann, dass $D_i \subseteq \{0,1\}^{poly(|i|)}$ gilt und dass F deterministisch ist.

2. *Schwer zu invertieren: Für jeden probabilistischen polynomiell–beschränkten Algorithmus A^*, und alle genügend großen n gilt*

$$Pr[A^*(I_n, f_{I_n}(X_n)) \in f_{I_n}^{-1}(f_{I_n}(X_n))] < \mu(n)$$

   wobei I_n eine Zufallsvariable ist, die die Ausgabe des Algorithmus I bei Eingabe von 1^n darstellt, und X_n eine Zufallsvariable ist, die die Ausgabe des Algorithmus D bei Eingabe von I_n darstellt.

Jede Kollektion von Einweg–Funktionen kann durch eine Einweg–Funktion repräsentiert werden und umgekehrt, wobei jede Formulierung Vorteile hat.

Trapdoor Funktionen können nun als Kollektionen von Einweg–Permutationen $\{f_i\}$ angesehen werden, die die Eigenschaft haben, dass f_i leicht zu invertieren ist, wenn erst einmal eine Hilfseingabe als „Abkürzung“ für den Index i gegeben ist. Diese Hilfseingabe

[6] Da die Übersetzung *Falltür Funktion* irreführend ist, sich jedoch kein besserer deutscher Begriff etabliert hat, wird hier weiterhin der Begriff Trapdoor verwendet.

für den Index i, bezeichnet mit $t(i)$ *kann nicht* effizient von i berechnet werden, obwohl es leicht möglich ist, ein korrespondierendes Tupel $(i, t(i))$ zu erzeugen.

Definition 2.23 (Kollektion von Trapdoor Permutationen)
Sei $I : 1^* \to \{0,1\}^* \times \{0,1\}^*$ ein probabilistischer Algorithmus und bezeichne $I_1(1^n)$ das erste Element des Tupels, welches von $I(1^n)$ ausgegeben wird. Ein Tripel von Algorithmen (I, D, F) wird als ***Kollektion von Trapdoor Permutationen*** *genannt, wenn die beiden folgenden Bedingungen erfüllt sind.*

1. *Das Tripel (I_1, D, F) bildet eine Kollektion von Einweg–Permutationen. (Dabei gilt insbesondere $F(i, x) = f_i(x)$.)*
2. *Es existiert ein polynomieller Algorithmus F^{-1}, so dass für jedes (i, t) im Indexbereich von I und für jedes $x \in D_i$ gilt, dass $F^{-1}(t, f_i(x)) = x$ ist.*

Eine nützliche Abschwächung der Bedingungen ist die Forderung, dass die Bedingungen mit sehr hoher Wahrscheinlichkeit erfüllt werden. Insbesondere wird dem Index–generierenden Algorithmus I gestattet, mit vernachlässigbarer Wahrscheinlichkeit Tupel (i, t) als Ausgaben zu erzeugen, für die sowohl f_i keine Permutation ist als auch $F^{-1}(t, f_i(x)) = x$ nicht für alle $x \in D_i$ erfüllt ist. Weiterführende Abhandlungen zu Trapdoor Funktionen sind unter anderem von Yao in [Yao82] veröffentlicht worden.

Eine weitere für die moderne Kryptographie bedeutsame Klasse von Funktionen sind die *Claw-Free* Funktionen.[7] Bei einer Claw-Free Kollektion handelt es sich um eine Menge von Paaren von Funktionen, die leicht zu evaluieren sind, bei dem jedes Paar von Funktionen denselben Bildbereich haben, und für die es unmöglich ist, ein Element aus dem Bildbereich zu finden, welches in beiden Funktionen denselben Ursprungswert hat.

Definition 2.24 (Claw-Free Kollektion)
Eine Kollektion von Paaren von Funktionen besteht aus einer unendlich großen Menge von Indizes $\bar{I}$, zwei endlichen Mengen D_i^0 und D_i^1 für jedes $i \in \bar{I}$ und zwei Funktionen f_i^0 und f_i^1, definiert über D_i^0 rsp. D_i^1. Eine solche Kollektion wird ***Claw-Free*** *genannt, wenn drei probabilistische polynomiell–beschränkte Algorithmen I, D und F existieren, so dass die folgenden drei Bedingungen erfüllt sind:*

1. *Leicht auszuwählen und zu berechnen: Die Zufallsvariable $I(1^n)$ ist Werten der Menge $\bar{I} \cap \{0,1\}^n$ zugeordnet. Für jedes $i \in \bar{I}$ und $\sigma \in \{0,1\}$ ist die Zufallsvariable $D(\sigma, i)$ verteilt über D_i^σ und für jedes $x \in D_i^\sigma$ gilt $F(\sigma, i, x) = f_i^\sigma(x)$.*
2. *Identische Bildverteilung: Für jedes i in der Indexmenge $\bar{I}$ sind die Zufallsvariablen $f_i^0(D(0, i))$ und $f_i^1(D(1, i))$ identisch verteilt.*

[7] Ähnlich wie bei den Trapdoor Funktionen existiert für die Claw-Free Funktionen keine sinnvolle deutsche Übersetzung. Claw, im Englischen *Klaue* oder *Kralle*, meint hier die Bedeutung von Aneinanderklammern oder wie mit einer Klaue zusammengehalten.

3. *Schwer, Claws zu erzeugen: Ein Paar (x, y), welches $f_i^0(x) = f_i^1(y)$ erfüllt, wird als* Claw *für den Index i bezeichnet. Bezeichne C_i die Menge der Claws für den Index i. Es ist erforderlich, dass für jeden probabilistischen polynomiell–beschränkten Algorithmus A^* und alle ausreichend großen n*

$$Pr[\mathsf{A}^*(I_n) \in C_{I_n}] < \mu(n)$$

gilt, wobei I_n eine Zufallsvariable ist, die die Ausgabeverteilung des Algorithmus I bei Eingabe von 1^n beschreibt.

Die erste Forderung in Definition 2.24 entspricht derjenigen in Definition 2.22, wohingegend die beiden anderen Forderungen sich widersprechen. Auf der einen Seite wird gefordert, dass Claws existieren. Auf der anderen Seite sollen diese Claws nicht effizient berechnet werden können. Es ist klar, dass eine Kollektion von Claw-Free Funktionen eine Kollektion von Einweg–Funktionen liefert. Von besonderem Interesse sind dabei Funktionen mit identischem Wertebereich (d.h. $D_i \stackrel{\text{def}}{=} D_i^0 = D_i^1$), wenn die Zufallsvariable $D(\sigma, i)$ gleichverteilt über D_i ist und die Funktionen f_i^0 und f_i^1 Permutationen über D_i sind. Derartige Kollektionen werden *Kollektionen von Claw-Free Permutationen von Paaren* genannt.

Den Abschnitt der Einweg–Funktionen soll die Definition der *Hashfunktionen* und insbesondere der kollisions-resistenten Hashfunktionen abschließen. Diese Funktionen werden ebenfalls in einem Großteil der modernen kryptographischen Algorithmen verwendet.

Definition 2.25 (Hashfunktionen)
Seien h eine Funktion, $l_h \in \mathbb{N}$ ein beliebiger aber fester Wert und x aus dem Definitionsbereich von h. Dann heißt h ***Hashfunktion****, wenn die folgenden drei Bedingungen erfüllt sind:*

1. *Komprimierung: $\forall x, \exists l_h : |h(x)| = l_h$*
2. *Einfach zu berechnen: Bei Eingabe von x ist $h(x)$ einfach zu berechnen.*
3. *Schwer zu invertieren: Sei $y = h(x)$. Dann ist es ohne Kenntnis von x berechenbar unmöglich, ein x' zu finden, so dass $h(x') = y$ gilt.*

Hashfunktionen haben dabei zwei wesentliche Eigenschaften:

- Sei M eine Nachricht und h eine Hashfunktion. Dann steht $y = h(M)$ in keinem erkennbaren Zusammenhang mehr mit M.
- Der erzeugte Hashwert ist im Allgemeinen wesentlich kürzer als die beliebig lange Nachricht $|y| < |M|$.

Von besonderem Interesse sind dabei häufig die kollisions-resistenten Hashfunktionen.

Definition 2.26 (Kollisions-resistente Hashfunktion)
Sei h eine Hashfunktionen gemäß Definition 2.25 und stammt x aus dem Definitionsbereich von h. h heißt ***kollisions-resistente Hashfunktion****, wenn es berechendbar unmöglich ist, ein $x' \neq x$ zu finden, so dass $h(x) = h(x')$ gilt.*

2.8 Zahlentheorie

Die nachfolgenden Ausführungen sollen nur einen kleinen Überblick über die notwendigen zahlentheoretischen Grundlagen liefern. Für das tiefere Verständnis der kryptographischen Verfahren ist das Wissen darum unabdingbar. Als Einstieg wird in Bezug auf kryptographische Besonderheiten z.B. auf die Ausführungen von Goldreich [Gol01], A. Pfitzmann [Pfi00], B. Pfitzmann [Pfi98] und von Menezes, van Oorschot und Vanstone [MvV01] verwiesen.

Im Weiteren bezeichne $n \in \mathbb{N}$ eine positive ganze Zahl und $\mathrm{ggT}(\cdot)$ den größten gemeinsamen Teiler.

Primzahl

Eine ganze Zahl $p \in \mathbb{N}$ ist eine *Primzahl* bzw. hat die Eigenschaft prim, wenn gilt $\forall i \in \mathbb{N}, i < p : \mathrm{ggT}(i, p) = 1$.

Relativ prim

$a, b \in \mathbb{N}$ heißen *relativ prim*, wenn gilt $\mathrm{ggT}(a, b) = 1$.

Euler'sche Phi Funktion $\phi(n)$

Für $n \geq 1$ bezeichnet $\phi(n)$ die Anzahl der ganzen Zahlen im Intervall $[1, n]$, die relativ prim zu n sind.

2.8.1 Integer modulo n

Modulo

Seien $a, b \in \mathbb{Z}$, Dann bedeutet $a \equiv b \pmod{n}$, dass a **kongruent** b ist, wenn n $(a - b)$ teilt, geschrieben $n|(a - b)$.

$$\exists m \in \mathbb{Z}; a - b = m \cdot n$$

Dazu gelten folgende Eigenschaften. Seien $a, a_1, b, b_1, c \in \mathbb{Z}$. Dann gilt

(i) $a \equiv b \pmod{n}$ dann und nur dann, wenn a und b das gleiche Restglied haben.

(ii) (Reflexivität) $a \equiv a \pmod{n}$

(iii) (Symmetrie) $a \equiv b \pmod{n} \iff b \equiv a \pmod{n}$.

(iv) (Transitivität) Aus $a \equiv b \pmod{n}$ und $b \equiv c \pmod{n}$ folgt $a \equiv c \pmod{n}$.

(v) Wenn $a \equiv a_1 \pmod{n}$ und $b \equiv b_1 \pmod{n}$ gilt, dann gilt auch $a + b \equiv a_1 + b_1 \pmod{n}$ und $ab \equiv a_1 b_1 \pmod{n}$.

Die Äquivalenzklasse einer ganzen Zahl $a \in \mathbb{Z}$ ist die Menge aller ganzen Zahlen, die kongruent zu a mod n sind. Aus den Eigenschaften (ii), (iii) und (iv) folgt, das für ein festes n die Relation der Kongruenz modulo n die Menge $\mathbb{Z}$ in Äquivalenzklassen teilt. Sei $a = qn+r$ mit $0 \leq r < n$, dann gilt $a \equiv r \pmod{n}$. Jede ganze Zahl a, die kongruent modulo n zu einer eindeutigen ganzen Zahl x mit $0 \leq x \leq n-1$ ist, wird als *kleinster Rest* eines Modulo n bezeichnet. Somit sind a und r in der selben Äquivalenzklasse und r kann als Repräsentant dieser Äquivalenzklasse dienen.

Restklassenring von n: $\mathbb{Z}_n$

Die Menge aller ganzen Zahlen modulo n werden als $\mathbb{Z}_n$ bezeichnet und stellen die Menge der (äquivalenten Klassen der) ganzen Zahlen von $\{0, 1, \ldots, n-1\}$ dar. Addition, Subtraktion und Multiplikation in $\mathbb{Z}_n$ werden jeweils modulo n ausgeführt.

Multiplikatives inverses Element

Seien $a, b, x \in \mathbb{Z}_n$. Das multiplikative Inverse von a mod n ist eine ganze Zahl x, so dass $ax \equiv 1 \pmod{n}$ gilt. Wenn so ein inverses Element existiert, dann ist es einmalig und a wird invertierbar genannt. Das inverse Element von a wird mit a^{-1} bezeichnet.

Eine Division von a durch b mod n ist ein Produkt von a und b^{-1} mod n und ist nur dann definiert, wenn b invertierbar modulo n ist.

Beispiel 1

Die invertierbaren Elemente von $\mathbb{Z}_9$ sind 1, 2, 4, 5, 7 und 8, wie am Beispiel der 4 zu sehen ist. $4^{-1} = 7$ da $4 \cdot 7 \equiv 1 \pmod{9}$.

Sei $d = \text{ggT}(a, n)$. Die Kongruenzrelation $ax \equiv b \pmod{n}$ hat eine Lösung x dann und nur dann, wenn $d|b$ gilt. In diesem Fall existieren exakt d Lösungen zwischen $0 \leq n-1$. Alle Lösungen sind kongruent modulo $\frac{n}{d}$.

2.8.2 Multiplikative Gruppe $\mathbb{Z}_n^*$

Die mulitplikative Gruppe von $\mathbb{Z}_n$ wird mit $\mathbb{Z}_n^* = \{a \in Z_n \mid \text{ggT}(a, n) = 1\}$ bezeichnet. Insbesondere gilt, falls n eine Primzahl ist, $\mathbb{Z}_n^* = \{a \mid 1 \leq a \leq n-1\}$

Beispiel 2
Sei $n = 21 \neq$ prim. Dann ist $\mathbb{Z}_{21}^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$. Sei $n = 13 =$ prim. Dann ist $\mathbb{Z}_{13}^* = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$.

Sei p ein Primzahl. Dann gilt

(i) (kleiner Fermat'schen Satz) Wenn $\text{ggT}(a, p) = 1$, dann gilt $a^{p-1} \equiv 1 \pmod{p}$.

(ii) Aus $r \equiv s \pmod{p-1}$ folgt $a^r \equiv a^s \pmod{p}$ für alle ganzen Zahlen a. Mit anderen Worten, wenn modulo einer Primzahl p gearbeitet wird, können die Exponenten mit modulo $(p-1)$ reduziert werden.

(iii) Insbesondere gilt $a^p \equiv a \pmod{p}$ für alle $a \in \mathbb{Z}$.

Grad von a

Sei $a \in \mathbb{Z}_n^*$. Der Grad von a, bezeichnet mit $grad(a)$, ist die kleinste Zahl $t \in \mathbb{Z}$, für die gilt: $a^t \equiv 1 \pmod{n}$.

Beispiel 3
Sei $n = 21$ mit $\mathbb{Z}_{21}^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$. Dann ist $\phi(21) = \phi(7) \cdot \phi(3) = 12 = |\mathbb{Z}_{21}^*|$. Die Grade der Elemente von $\mathbb{Z}_{21}^*$ sind in der Tabelle 1 aufgelistet.

$a \in \mathbb{Z}_{21}^*$	1	2	4	5	8	10	11	13	16	17	19	20
Grade von a	1	6	3	6	2	6	6	2	3	6	6	2

Tabelle 1: Grad der Elemente von $\mathbb{Z}_{21}^*$

Erzeugendes Element von $\mathbb{Z}_n^*$

Sei $\alpha \in \mathbb{Z}_n^*$. Ist $grad(\alpha) = \phi(n)$, dann heißt α das *erzeugende Element* (engl. generator) oder auch *primitive Elemenet* von $\mathbb{Z}_n^*$. $\mathbb{Z}_n^*$ heißt *zyklisch*, wenn es ein erzeugendes Element α hat.

Eigenschaften des erzeugenden Elements

(i) $\mathbb{Z}_n^*$ hat dann und nur dann ein erzeugendes Element, wenn $n = 2, 4, p^k, 2p^k$ gilt, wobei p eine ungerade Primzahl und $k \geq 1$ sind. Insbesondere gilt, wenn p prim ist, dass $\mathbb{Z}_n^*$ ein erzeugendes Element hat.

(ii) Wenn α ein erzeugendes Element von $\mathbb{Z}_n^*$ ist, dann ist $\mathbb{Z}_n^* = \{\alpha^i \bmod n \mid 0 \leq i \leq \phi(n) - 1\}$.

(iii) Vorausgesetzt, α ist ein erzeugendes Element von $\mathbb{Z}_n^*$. $b = \alpha^i \bmod n$ ist ebenfalls ein erzeugendes Element von $\mathbb{Z}_n^*$ dann und nur dann, wenn $\text{ggT}(i, \phi(nI) = 1)$. Falls $\mathbb{Z}_n^*$ zyklisch ist, folgt daraus, dass die Anzahl der erzeugenden Elemente $\phi(\phi(n))$ ist.

(iv) $\alpha \in \mathbb{Z}_n^*$ ist ein erzeugendes Element von $\mathbb{Z}_n^*$ dann und nur dann, wenn für jeden Primzahlteiler p von $\phi(n)$ gilt $\alpha^{\frac{\phi(n)}{p}} \not\equiv 1 \pmod{n}$.

Beispiel 4
$\mathbb{Z}_{21}^*$ ist nicht zyklisch, weil $\phi(21) = 12 \notin \mathbb{Z}_{21}^*$ (siehe Tabelle 1). Beachte, dass $\alpha = 21$ nicht die Eigenschaft (i) für erzeugende Elemente erfüllt. Dagegen ist $\mathbb{Z}_{25}^*$ zyklisch und hat $\alpha = 2$ als erzeugendes Element.

2.8.3 Quadratischer Rest

$a \in \mathbb{Z}_n^*$ wird als *quadratischer Rest modulo n* bezeichnet, wenn ein $x \in \mathbb{Z}_n^*$ exisiert, so dass $x^2 \equiv a \pmod{n}$ gilt. Existiert kein derartiges x, dann wird a als quadratischer *Nichtrest* modulo n bezeichnet.

Die Menge aller quadratischen Reste modulo n wird mit Q_n, die der quadratischen Nichtreste modulo n mit $\overline{Q}_n$ bezeichnet. Dabei gilt per Definition $0 \notin \mathbb{Z}_n^*$, wobei $0 \notin Q_n$ und $0 \notin \overline{Q}_n$ gilt.

Sei p eine ungerade Primzahl und a ein erzeugendes Element von $\mathbb{Z}_n^*$. $x \in \mathbb{Z}_n^*$ ist ein quadratischer Rest modulo p dann und nur dann, wenn $x = a^i \pmod{p}$ für eine gerade Zahl $i \in \mathbb{Z}$. Es folgt, dass $|Q| = \frac{p-1}{2}$ und $|\overline{Q}| = \frac{p-1}{2}$. Das heißt, dass die Hälfte der Elemente von $\mathbb{Z}_n^*$ quadratische Reste und die andere Hälfte quadratische Nichtreste sind.

Beispiel 5
$a = 6$ ist erzeugendes Element von $\mathbb{Z}_{13}^*$. Die Potenzen von a ergeben sich aus folgender Tabelle:

i	1	2	3	4	5	6	7	8	9	10	11	12
$a^i \bmod 13$	1	6	10	8	9	2	12	7	3	5	4	11

Daraus folgt $Q_{13} = \{1, 3, 4, 9, 10, 12\}$ sowie $\overline{Q}_{13} = \{2, 5, 6, 7, 8, 11\}$.

2.8.4 Blum-Zahl

Eine Zahl $n \in \mathbb{Z}$ wird *Blum-Zahl* genannt, wenn es zwei Primzahlen $p \neq q$ mit $n = pq$ gibt, die kongruent zu 3 modulo 4 sind

Sei $n = pq$ eine Blum-Zahl und $a \in Q_n$. Dann hat a genau vier Quadratwurzeln modulo n, von denen genau eine ebenfall in Q_n liegt. Diese einzige in Q_n liegende Zahl wird *Hauptquadratwurzel* genannt.

Beispiel 6
Sei $n = 21$ eine Blum-Zahl. Die vier Wurzeln für eine Zahl $a = 4$ lauten dann 2, 5, 16 und 19. Die einzige in Q_n liegende Hauptquadratwurzel ist dabei 16.

Ist $n = pq$ eine Blum-Zahl, dann ist die Funktion $f : Q_n \to Q_n$ definiert durch $f(x) = x^2 \pmod{n}$ eine Permutation. Das Inverse der Funktion f ist

$$f^{-1}(x) = x^{\frac{(p-1)(q-1)+4}{8}} \bmod n.$$

2.8.5 Diskreter Logarithmus

Die Sicherheit vieler kryptographischer Techniken basiert auf der Schwierigkeit des diskreten logarithmischen Problems (DLP).

Für diesen Abschnitt bezeichnet G eine endliche zyklische Gruppe vom Grad n mit dem erzeugenden Element α. G kann sich vorgestellt werden als eine multiplikative Gruppe $\mathbb{Z}_n^*$ vom Grad $p-1$, beim dem die Gruppenoperation eine Multiplikation modulo p ist.

Diskreter Logarithmus

Seien G und α wie beschrieben und sei $\beta \in G$. Der *diskrete Logarithmus* von β zur Basis α, bezeichnet mit $\log_\alpha \beta$, ist die eindeutige ganze Zahl x mit $0 \leq x \leq n-1$, so dass $\beta = \alpha^x$ gilt.

Beispiel 7
Sei $p = 97$. Dann hat die zyklische Gruppe $\mathbb{Z}_{97}^*$ den Grad $n = 96$. Ein erzeugendes Element von $\mathbb{Z}_{97}^*$ ist $\alpha = 5$. Da $5^{32} \equiv 35 \pmod{97}$ gilt, liegt $\log_5 35 = 32$ in $\mathbb{Z}_{97}^*$.

Nachfolgend ein paar elementare Regeln über Logarithmen:

Sei G eine zyklische Gruppe vom Grad n mit einem erzeugenden Elemente α. Seien weiterhin $\beta, \gamma \in G$ und s eine beliebige ganze Zahl. Dann gilt:

(i) $\log_\alpha(\beta\gamma) = (\log_\alpha \beta + \log_\alpha \gamma) \bmod n$

(ii) $\log_\alpha \beta^s = s \log_\alpha \beta \bmod n$

Problem des diskreten Logarithmus (DLP)

Das *diskrete Logarithmus Problem* besagt folgendes: Seien p eine Primzahl, α ein erzeugendes Element von $\mathbb{Z}_p^*$ und $\beta \in \mathbb{Z}_p^*$. Dann wird die ganze Zahl x mit $0 \leq x \leq p-2$ gesucht, für die $\alpha^x \equiv \beta \pmod{p}$ gilt.

Dazu existiert die allgemeine Form des

Problem des allgemeinen diskreten Logarithmus (GDLP)[8]

Sei G eine endliche zyklische Gruppe vom Grad n mit dem erzeugenden Element α und einem weiteren Element $\beta \in G$. Gesucht wird die ganze Zahl x mit $0 \leq x \leq n-1$, so dass $\alpha^x = \beta$ gilt.

Anmerkung: Die Schwiergikeit des GDLP ist unabhängig vom gewählten erzeugenden Element. Seien α und γ zwei erzeugende Elemente der zyklischen Gruppe G vom Grad n und sei $\beta \in G$. Seien $x = \log_\alpha \beta$, $y = \log_\gamma \beta$ und $z = \log_\alpha \gamma$. Dann gilt $\alpha^x = \beta = \gamma^y = (\alpha^z)^y$. Daraus folgt $x = zy \bmod n$ und

$$\log_\gamma \beta = (\log_\alpha \beta)(\log_\alpha \gamma)^{-1} \bmod n$$

Das bedeutet, dass jeder Algorithmus, der den Logarithmus zur Basis α berechnet, benutzt werden kann, um den Logarithmus zu jeder beliebigen Basis γ zu berechnen, wenn γ ein erzeugendes Element von G ist.

Die einfachste Lösung für das GDLP besteht darin, nacheinander alle $\alpha^0, \alpha^1, \alpha^2, \ldots$ zu berechnen, bis β gefunden wurde. Diese Methode benötigt $O(n)$ Versuche, wobei n der Grad von α, β ist, und ist daher für große n ineffizient (was im Falle der Kryptographie von besonderem Interesse ist). Eine Form von Annäherungsverfahren kann nicht verwendet werden, denn im Gegensatz zur reellen Logarithmusfunktion kann man über die diskrete nicht sagen, sie sei stetig oder monoton. Beispielsweise folgt aus $x < y$ nicht $\log_\alpha(x) < \log_\alpha(y)$.

Die schnellsten bislang bekannten Verfahren zur Berechnung des diskreten Logarithmus benötigen in Abhängigkeit der Gruppeneigenschaft und des Grades (z.B. ob der Grad eine Primzahl oder sogar eine Blum-Zahl ist) subexponenzielle Zeit, werden hier jedoch nicht weiter aufgeführt.

2.8.6 Binärer Körper – Galois-Feld $GF(2)$

Die meisten der in dieser Arbeit vorgestellten Protokolle arbeiten in den Basis-Algorithmen lediglich auf einzelnen Bits. Aus diesem Grund sind einige explizit auf dem binären Körper definiert bzw. nutzen diesen.

[8] engl: generalized discrete logarithm problem

Im Gegensatz zu den unbeschränkten Körpern wie z.B. $\mathbb{R}$ oder $\mathbb{C}$, die unendlich viele Elemente beinhalten, werden in der modularen Arithmetik nur Körper mit endlich vielen Elementen betrachtet. Endliche Körper werden auch *Galois-Felder* genannt und mit GF bezeichnet, wobei mit einer Zahl p der Grad des Körpers $GF(p)$ angegeben wird. Dieser Körper enthält genau p unterschiedliche Elemente. Die Zahl p darf nicht beliebig gewählt werden, um die Axiome der endlichen Körper zu erfüllen. Sie muss eine Primzahl oder eine geradzahlige Potenz einer Primzahl sein.

Das Rechnen modulo p von ganzen Zahlen aus $\mathbb{Z}$ genügt allen Axiomen eines Ringes. Dabei bildet die Menge aller Zahlen unter den beiden Rechenoperationen Addition und Multiplikation jeweils modulo p den Restklassenring $GF(p) = (\mathbb{Z}_p, \oplus, \odot)$.

Hier soll keine vertiefende Einführung in die endlichen Körper vorgenommen werden. Dazu wird auf weiterführende Literatur verwiesen, die hauptsächlich der Nachrichtentechnik und den Codierungsverfahren entstammt. Dort werden Galois-Felder unterschiedlicher Grade verwendet. Als ein Einstieg zu Galois-Feldern kann u.a. [Sta02] empfohlen werden.

Für die moderne Kryptographie ist besonders der zweielementige oder binäre Körper $GF(2) = (\{0, 1\}, +, *)$ von Interesse. Aufgrund der Modulo-Rechnung gelten folgende Rechenregeln, die in Form einer Additions- und Multiplikationstabelle dargestellt werden:

+	0	1
0	0	1
1	1	0

*	0	1
0	0	0
1	0	1

Tabelle 2: Additions- und Multiplikationstabelle im $GF(2)$

Der Umgang mit dem Körper $GF(2)$ ist im Hinblick auf die moderne Kryptographie und die zuvor beschriebenen Schaltkreise besonders gut geeignet, da eine Addition in $GF(2)$ einem XOR-Gatter und eine Multiplikation einem AND-Gatter entspricht (siehe auch Beschreibung von Schaltkreisen in Abschnitt 2.6 ab Seite 17).

3 Zero–Knowledge Proof

3.1 Definition des Zero–Knowledge Proofs

Aufgrund der besonderen Bedeutung der Zero–Knowledge Proofs sowohl in der theoretischen Kryptographie als auch in der theoretischen Informatik wurden in der Folgezeit unzählige Arbeiten über dieses Thema veröffentlicht. Neben den Hauptarbeiten von Goldwasser, Micali und Rackoff [GMR85] sowie Goldreich, Micali und Widgerson [GMW86] sowie ähnlich gelagerten Arbeiten wie die Arthur–Merlin Games von Babai [Bab85] oder eine praxisnähere Abwandlung der Arbeiten aus [GMR85] durch Tompa und Woll [TW87] ist als allgemeine Einführung insbesondere das Grundlagenwerk von Goldreich [Gol01] zu erwähnen.

In dieser Arbeit soll nur ein einführender Überblick über Zero–Knowledge Proofs gegeben werden, um dann in Kapitel 4 ab Seite 36 darauf aufbauend die Zero–Knowledge Arguments zu definieren.

Kurz gesagt bedeutet Zero–Knowledge Proof ein interaktives Beweissystem (P, V), wenn dem Verifier V in der Interaktion mit dem Prover P für eine Eingabe $x \in L$ keine Information vermittelt wird, die V nicht selber aus x ohne Interaktion mit P berechnen kann. Das gilt auch für einen eventuell täuschenden Verifier V^*. Zero–Knowledge ist eine Eigenschaft des Prover P. Sie besagt, wie sicher P gegen Angriffe ist, die auf die Herausgabe von Informationen gerichtet sind.

Zur Prüfung der Zero–Knowledge Eigenschaft wird daher ein formales Kriterium benötigt, mit dem die Ausgabe von P während der gesamten Kommunikationsphase mit beliebigen Verifiern V^* überprüft werden kann. Zur Prüfung wird für jede Maschine V^* ein polynomiell–beschränkter *Simulator* konstruiert, so dass sich die Ausgabe dieses Simulators (berechenbar) nicht von dem für V^* sichtbaren Protokoll unterscheidet.

Definition 3.1 (Zero–Knowledge Proof)
Sei (P, V^*) ein interaktives Beweissystem für eine Sprache L mit $x \in L$. Dann hat (P, V^*) die Zero–Knowledge *Eigenschaft bzw. ist ein Zero–Knowledge Proof, wenn für jede Maschine V^* eine probabilistische polynomiell–beschränkte Maschine M^* existiert, so dass die beiden Familien von Wahrscheinlichkeitsverteilungen $\{\mathsf{view}_{V^*}^{P}(x)\}_{x\in L}$ und $\{M^*(x)\}_{x\in L}$ (berechenbar) ununterscheidbar sind.*

Dabei ist zu beachten, dass *berechenbar* ununterscheidbar eine Schwächung der Sicherheit des Provers bedeutet, da in der Praxis der Verifier nach der Kommunikation erheblich mehr Zeit und gegebenenfalls eine höhere Berechnungskomplexität zur Verfügung haben kann, um aus dem Kommunikationsprotokoll eventuell doch noch weitere Informationen zu berechnen.

3.2 Perfekt Zero–Knowledge versus Berechenbar Zero–Knowledge

Entsprechend der Einteilung in Definition 2.14 auf Seite 14 wird daher Zero–Knowledge in *Perfekt Zero–Knowledge* und *Berechenbar Zero–Knowledge* unterschieden.

Der entscheidende *Unterschied* zwischen *berechenbarem* und *perfektem* Zero–Knowledge liegt darin, dass im ersten Fall der Verifier möglicherweise Informationen über das Geheimnis des Provers erlangt, die von dem Verifier nicht in polynomieller Zeit entschlüsselt werden können (außer die kryptographischen Annahmen versagen), während im zweiten Fall der Verifier überhaupt keine Informationen über das Geheimnis des Provers erhält (über die Existenz und dem Umstand, dass der Prover eines hat, hinaus).

Die von einer Maschine V^* erlangten Informationen könnten trotz einer entsprechenden polynomiellen Beschränkung V^*'s im Nachhinein für einen beliebigen Zeitraum und unter Zuhilfenahme einer beliebige Anzahl weiterer Maschinen „untersucht“ werden. In der Praxis existieren Verfahren, wo der Zeitraum unter Beachtung der vorhandenen Rechnerkapazitäten überschaubar ist, in dem die Informationen des Provers geheim bleiben müssen. In diesen Fällen kann mittels des Sicherheitsgrenzwerts n_k bei der μ-Funktion (siehe dazu Definition 2.13 auf Seite 13) die gewünschte Höhe der Sicherheit gewählt werden.

In anderen Fällen der praktischen Kryptographie ist es dagegen erforderlich, dass die dem Prover bekannten Informationen auch im Nachhinein nicht berechenbar sind, so dass ein perfektes Zero–Knowledge Verfahren benötigt wird.

Es ist somit von der relativen Wichtigkeit bezüglich der Geheimhaltung, der Rechengeschwindigkeit der Maschinen in Verbindung mit der Komplexität der verwendeten Algorithmen sowie der Täuschungssicherheit gegen den Prover abhängig, welche Verfahren eingesetzt werden. In der Praxis kann davon ausgegangen werden, dass die Ehrlichkeit des Provers und damit dessen Täuschungspotenzial nur für die Dauer der Ausführung eine mitentscheidende Rolle spielt, weshalb die Entscheidung für perfekte bzw. berechenbare Zero–Knowledge Verfahren überweigend nur noch von der Komplexität der Algorithmen beeinflusst werden.

3.3 Komplexität des Zero–Knowledge Proofs

Die Menge der Sprachen L, für die ein Zero–Knowledge Proof existiert, wird wie folgt beschrieben.

Definition 3.2 (Klasse der Zero–Knowledge Proofs)
Die Klasse $\mathcal{ZK}$ enthält alle Sprachen, für die ein Zero–Knowledge Proof existiert. Mit $\mathcal{PZK}$ rsp. $\mathcal{CZK}$ werden die Klassen der Sprachen bezeichnet, die einen perfekten Zero–Knowledge Proof rsp. einen berechenbaren (engl. Computational) Zero–Knowledge Proof haben.

Von Goldreich, Micali und Widgerson wurde in [GMW86] gezeigt, dass jede Sprache in $\mathcal{NP}$ einen Zero–Knowledge Proof besitzt. Das wurde mittels der $\mathcal{NP}$-vollständigen 3-färbbaren Graphen nachgewiesen, die zur Klasse $3\mathcal{C}$ gehören, für die die Existenz von Zero–Knowledge Proofs bewiesen wurde. Somit gilt $3\mathcal{C} \in \mathcal{ZK}$.

Der dazugehörige Zero–Knowledge Proof wurde wie folgt durchgeführt.

Definition 3.3 (Zero–Knowledge Proof für Dreifärbbare Graphen)
Sei $G(V, E)$ ein Graph. Die Knotenfärbung sei eine Funktion $\phi : V \to \{1, 2, 3\}$. Seien $n = |V|$, $m = |E|$ und der Einfachheit halber $V = \{1, 2, \ldots, n\}$.

Die folgenden vier Schritte werden m^2 Mal ausgeführt, wobei in jedem Schritt unabhängige Münzwürfe benötigt werden.

Unter der Voraussetzung, dass ein sicheres und eindeutiges Verschlüsselungsverfahren mit $f(x, r) = f(y, r) \Rightarrow x = y$ mit r als zufälligem Münzwurf existiert, gilt:

1. *P wählt eine Permutation $\pi \in_R Sym(\{1, 2, 3\})$ mit zufälligen r_v's der Drei-Färbung, verschlüsselt diese, indem für jedes $v \in V$ die $R_v = f(\pi(\phi(v)), r_v)$ berechnet werden, und sendet diesen verschlüsselten Graphen als Folge $R_1, \ldots, R_n$ an V.*

2. *V wählt zufällig eine Kante $e \in_R E$ und sendet diese an P. Damit fragt V nach der Färbung der Endpunkte von $e \in E$.*

3. *Wenn $e = (u, v) \in E$ gilt, offenbart P die Färbung von u und v, indem er die Verschlüsselung dieser Färbungen als $(\pi(\phi(u)), r_u)$ und $(\pi(\phi(v)), r_v)$ an V sendet. Ist $e \notin E$ stoppt P.*

4. *V prüft den aus Schritt (3) übermittelten „Beweis“. Insbesondere prüft V ob*

   - $R_u = f(\pi(\phi(u)), r_u)$,
   - $R_v = f(\pi(\phi(v)), r_v)$,
   - $\pi(\phi(u)) \neq \pi(\phi(v))$ *und*
   - $\pi(\phi(u)), \pi(\phi(v)) \in \{1, 2, 3\}$.

   Ist eine der Bedingungen verletzt, dann lehnt V ab und stoppt. Andernfalls fährt V mit der nächsten Iteration fort.

Die Drei-Färbung wird akzeptiert, wenn V alle Iterationen erfolgreich durchlaufen hat.

Der Nachweis über die Korrektheit ist etwas umfangreicher und kann in [GMW86] nachgelesen werden. Darüber hinaus wird dort gezeigt, dass nicht nur jede $\mathcal{NP}$-Sprache sondern jede Sprache in $\mathcal{IP}$ einen Zero–Knowledge Proof hat. Damit sind auch diejenigen

Sprachen umfasst, die zwar nicht in $\mathcal{NP}$ liegen, für die aber trotzdem ein effizientes Beweisverfahren existiert.

In [Gol01] ist nachzulesen, dass folgende Beziehungen gelten:

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

Es wird allgemein angenommen, dass die Beziehungen zwischen $\mathcal{BPP}$, $\mathcal{PZK}$ und $\mathcal{CZK}$ sogar strikt sind. Unter der Annahme der Existenz von Einweg–Funktionen gilt darüber hinaus, dass $\mathcal{CZK}$ und $\mathcal{IP}$ identisch sind. Somit dürften folgende Beziehungen zwischen den Klassen bestehen:

$$\mathcal{BPP} \subset \mathcal{PZK} \subset \mathcal{CZK} = \mathcal{IP}$$

3.4 Arthur–Merlin Beweissysteme

1985 wurden von Babai in [Bab85] die sogenannten *Arthur–Merlin Games* eingeführt. Dabei wurde versucht, die kleinste Klasse von Sprachen zu charakterisieren, für die interaktive Beweissyssteme existieren. Der grundsätzliche Unterschied zu den bis hierher definierten interaktiven Beweissystemen ist der Umstand, dass bei den Arthur–Merlin Games der Prover und der Verifier (dort als Merlin und Arthur bezeichnet) eine gemeinsame Eingabe bezüglich des Zufallsbandes haben. Aus diesem Grund werden heute Zero–Knowledge Proof und Zero–Knowledge Argument Systeme, die über eine öffentliche, gemeinsame Eingabe verfügen, auch als Arthur–Merlin Systeme bezeichnet. Eine formale Definition der Arthur–Merlin Games, wie sie von Babai erfolgte, wird hier unterlassen, da diese Definition nicht weiter benötigt wird. Von Goldwasser und Sipser wurde jedoch in [GS86] gezeigt, dass Arthur–Merlin bzw. öffentliche Münzwurf Systeme equivalent zu allgemeinen interaktiven Beweissystemen sind.

3.5 Witness indistinguishing - Witness hiding

Die Konstruktion von Zeugnis–ununterscheidbaren und Zeugnis–verbergenden Systemen (engl. *Witness Indistinguishable and Witness Hiding Protocols*) wurde von Feige und Shamir in [FS90a] vorgestellt. Diese Technik ermöglicht innerhalb der Zero–Knowledge Proofs neue kryptographische Algorithmen mit Eigenschaften, die mit der bisherigen Konstruktion der Zero–Knowledge Proofs nicht möglich sind.

Grob gesagt ist ein interaktives Protokoll (A, B) *Zeugnis–ununterscheidbar*, wenn A eines von vielen geheimen Zeugnissen für eine $\mathcal{NP}$ Behauptung nutzt, ohne dass B mit

einiger Wahrscheinlichkeit sagen kann, welches Zeugnis A verwendet. Das Protokoll ist *Zeugnis–verbergend*, wenn am Ende der Kommunikation B kein neues Zeugnis berechnen kann, welches er nicht schon zu Beginn der Interaktion kannte.

In [FS90a] werden dabei zwei zentrale Resultate bewiesen:

1. Im Gegensatz zu Zero–Knowledge Protokollen bleibt die Zeugnis–ununterscheidbarkeit auch unter beliebiger Komposition, insbesondere also auch unter paralleler Ausführung erhalten.

2. Wenn eine Aussage mindestens zwei unabhängige Zeugnisse hat, dann ist jedes Zeugnis–ununterscheidbare Protokoll für diese Aussage auch Zeugnis–verbergend.

Der entscheidende Unterschied zu Zero–Knowledge Protokollen ist dabei, dass in dem Ausgabeprotokoll des Provers eines Zero–Knowledge Protokolls überhaupt keine Information vorhanden ist, die der Verifier nicht auch selber berechnen könnte. Bei einem Zeugnis–verbergenden Protokoll ist lediglich gefordert, dass in einem Ausgabeprotokoll des Provers keine Information über das Geheimnis bzw. das Zeugnis enthalten ist.

Definition 3.4 (Zeugnis–ununterscheidbar)
Seien R eine binäre Relation, $L = L(R)$ eine Sprache, x ein genügend großes Wort aus L und $R(x)$ die Menge der Zeugnisse für x. Ein Beweissystem (P, V) ist ***Zeugnis–ununterscheidbar*** *über R, wenn für jeden Verifier V^*, alle $z_1, z_2 \in R(x)$ und jede Hilfsfunktion $f(\cdot)$ für V^* die Familien $\{view_{V^*}(P(x, z_1), V^*(x, f(x)))\}_{x \in L}$ und $\{view_{V^*}(P(x, z_2), V^*(x, f(x)))\}_{x \in L}$ ununterscheidbar sind.*

Im Unterschied zur Zero–Knowledge Definition wird dabei kein Simulator benötigt.

In [FS90a] wird bewiesen, dass unter der Voraussetzung der Existenz von Einweg–Funktionen jede $\mathcal{NP}$ Sprache ein Zeugnis–ununterscheidbares Beweissystem mit konstanter Rundenzahl hat.

Das Konzept des *Zeugnis–verbergends* ist eine mögliche Alternative zu Zero–Knowledge. Es stellt geringere Anforderungen als Zero–Knowledge, erfüllt aber in vielen Fällen die Sicherheitsanforderungen in kryptographischen Protokollen.

Im Rahmen dieser Arbeit ist die Eigenschaft eines Protokolls, Zeugnis–verbergend zu sein, nicht weiter relevant, so dass auf weitergehende Erläuterungen verzichtet wird.

3.6 Statistisch Korrekt versus Berechenbar Korrekt

Eine weitere Unterscheidungsmöglichkeit bei bekannten Zero–Knowledge Protokollen ist die aus der Sicht des Verifiers, d.h. dass nach dem Grad der Korrektheit des Provers unterschieden wird.[9]

[9] Siehe dazu Definition 4.1, *Korrektheitsbedingung*, auf Seite 37

Während oft beim perfekten versus berechenbaren Zero–Knowledge die Geheimhaltung über die Kommunikation hinaus gewährleistet sein muss, braucht in den meisten Fällen die Korrektheit des Provers nur während der Kommunikationsphase von einigen Sekunden bzw. Minuten gesichert sein. Das interaktive Beweissystem muss in der Praxis keinem täuschenden Prover mit unbegrenzter Rechenleistung oder Rechenzeit standhalten. In Abhängigkeit bekannter Computerleistungen könnte z.B. der Verifier nach einer wählbaren Zeitschranke die Kommunikation abbrechen bzw. bräuchte dem Prover beim Überschreiten der Zeitschranke nicht mehr vertrauen.

Aus diesem Grund hat sich ausgehend von Arbeiten von Brassard, Chaum und Crépeau [BC86, Cha87, BCC88] der Begriff *Zero–Knowledge Argument* oder auch *Computational Sound Proofs* gebildet.

In Anlehnung an Bellare, Jakobsson und Yung [BJY97] werden Zero–Knowledge-*Proofs* und Zero–Knowledge-*Arguments* nach ihrem Grad der Korrektheit abgegrenzt. Es wird dabei unterschieden, ob und mit welcher Mächtigkeit ein täuschender Prover einem Verifier eine unwahre Tatsache als wahr vermitteln kann. Bei einem Zero–Knowledge Proof gelingt dieses selbst unbeschränkten Provern nur mit einer vernachlässigbar kleinen Fehlerwahrscheinlichkeit. Bei einem Zero–Knowledge Argument gilt dieser kleine Fehlerquotient nur noch für Prover, die eine polynomiell–beschränkte Rechenzeit besitzen.

Falls $x \notin L$ gilt, stellt sich die Frage, wie klein die Fehlerwahrscheinlichkeit $\varepsilon(\cdot)$ ist, mit der ein täuschender Prover P einen Verifier V davon überzeugen kann, dass trotzdem $x \in L$ gilt.

Definition 3.5 (Grad der Korrektheit)
Sei (P, V) in interaktives Beweissystem für eine Sprache L, $x \notin L$, $k \in \mathbb{N}$ eine Konstante und $\varepsilon : \mathbb{N} \to \mathbb{R}$ eine Funktion.

Statistisch Korrekt[10] *Nur ein nicht polynomiell–beschränkter Prover P wird V davon überzeugen können, dass $x \in L$ gilt, wobei die erwartete Fehlerwahrscheinlichkeit $\varepsilon(k)$ ist.*

Berechenbar Korrekt[11] *Einem polynomiell–beschränkten Prover P ist es mit Ausnahme einer Fehlerwahrscheinlichkeit von $\varepsilon(k)$ unmöglich, V davon zu überzeugen, dass $x \in L$ gilt. Einem mächtigeren P könnte dieses jedoch mit großer Wahrscheinlichkeit gelingen.*

[10] Das ist die Notation aus [GMR85].
[11] Diese ursprüngliche Notation stammt von Brassard, Chaum und Crépeau [BC86, Cha87, BCC88]

4 Zero–Knowledge Arguments

Wie schon bei der Diskussion des Zero–Knowledge Proofs im Abschnitt 3.2 auf Seite 31 erwähnt wurde, muss bei einem in der Praxis eingesetzten Beweissystem in der Regel nur für die Dauer der Ausführung gewährleistet sein, dass der Prover den Verifier nicht täuschen kann. Kurz nach Veröffentlichung der Zero–Knowledge Proofs versuchte die Forschung daher, für praktische Anwendungsfälle geringere aber ausreichende Anforderungen an interaktive Beweissysteme stellen zu müssen. Brassard, Chaum und Crépeau [BCC88] veröffentlichten 1988 die erste Arbeit, die sich mit berechenbar eindeutigen im Gegensatz zu perfekt eindeutigen Beweissystemen befasst. Diese werden als *computationally sound proof systems* oder auch *argument systems* bezeichnet.

Der Hauptteil dieser Arbeit beschäftigt sich damit, typische Vertreter dieser Zero–Knowledge Arguments vorzustellen. Aus dem Abschnitt 3.6 ergibt sich die Unterscheidung von Zero–Knowledge Arguments und Zero–Knowledge Proofs dadurch, dass Zero–Knowledge Arguments ausschließlich solche Zero–Knowledge Verfahren sind, die nur für die Dauer der Kommunikation für (in der Realität existierende) polynomiell–beschränkte Prover[12] die Korrektheit der Ausgaben des Provers P garantieren.

Somit erfüllen alle im weiteren Verlauf aufgeführten Protokolle die strengeren Anforderungen eines Zero–Knowledge Proofs bezüglich der Korrektheit nicht. Da eine statistische Korrektheit selbstverständlich eine berechenbare Korrektheit umfasst, erfüllen alle Zero–Knowledge Proof Protokolle die Anforderungen an Zero–Knowledge Arguments. Hier werden jedoch nur diejenigen Protokolle dargestellt und die Besonderheiten hervorgehoben, die sich aus der Beschränkung auf die algorithmische Korrektheit ergeben. Insofern fallen Protokolle aus Arbeiten wie z.B. von Goldreich und Kahan [GK96a], Dwork und Stockmeyer [DS02] und anderen heraus, die zwar für die Grundlagenforschung im Bereich der Zero–Knowledge Systeme wichtig sind, aber die ausschließlich auf perfekte bzw. statistische Korrektheit aufbauen.

Zero–Knowledge Arguments unterscheiden sich von den Zero–Knowledge Proofs, in dem die Bedingung der Eindeutigkeit des zugrunde liegenden interaktiven Systems nur für polynomiell–beschränkte Prover gewährleistet wird. Die formale Definition für interaktive Argumentsysteme lautet:

Definition 4.1 (Interaktives Argumentsystem)
Sei $L \subseteq \{0,1\}^*$ eine Sprache, P und V interaktive probabilistische polynomiell–beschränkte Turing Maschinen und sei (P, V) ein interaktives System, welches beim Akzeptieren einer Eingabe eine „1“ auf die Ausgabe schreibt. Dann ist (P, V) ein **interaktives Argumentsystem für eine Sprache** *L, wenn gilt:*

[12] Im Gegensatz zu den informationstheoretisch existierenden unbeschränkten Turing Maschinen.

- *Vollständigkeitsbedingung:*

$$\forall k \in \mathbb{N}, \exists N \in \mathbb{N} : \forall x \in L, |x| \geq N : \Pr[(P, V)(x) = 1] \geq 1 - \frac{1}{|x|^k}$$

- *Korrektheits- oder Eindeutigkeitsbedingung:*

  Für jede (möglicherweise täuschende) interaktive probabilistische polynomiell–beschränkte Turing Maschine P^* gilt

$$\forall P^* : \forall k \in \mathbb{N}, \exists N \in \mathbb{N} : \forall x \notin L, |x| \geq N : \Pr[(P^*, V)(x) = 1] < \frac{1}{|x|^k}$$

Somit sind die interaktiven Beweissysteme mit Zusatzeingabe gemäß Definition 2.12 gleichzeitig auch *interaktive Argumentsysteme mit Zusatzeingabe*.

Damit kann die Definition der Zero–Knowledge Argument Systeme analog der Definition 3.1 für Zero–Knowledge Proofs erfolgen:

Definition 4.2 (Zero–Knowledge Argument)
*Sei (P, V^*) ein interaktives Argumentsystem für eine Sprache L mit $x \in L$. Dann ist (P, V^*) ein **Zero–Knowledge Argument**, wenn für jede Maschine V^* eine probabilistische polynomiell–beschränkte Maschine M^* existiert, so dass die beiden Familien von Wahrscheinlichkeitsverteilungen $\{\mathsf{view}_{V^*}^{P}(x)\}_{x \in L}$ und $\{M^*(x)\}_{x \in L}$ (berechenbar) ununterscheidbar sind.*

Ein wichtiges Untersuchungsfeld der Forschung in Bezug auf Zero–Knowledge Beweissysteme ist die Minimierung der Komplexität. Im Zusammenhang mit Zero–Knowledge Systemen bezieht sich der Begriff Komplexität auf

- die erforderlichen algebraischen bzw. informationstheoretischen Voraussetzungen und Annahmen, die für das entsprechende Zero–Knowledge Argument erforderlich sind, und
- die Anzahl der Informationen, die während der Kommunikation zwischen Prover und Verifier ausgetauscht werden müssen.

Im ersten Fall ist das Ziel, möglichst keine Annahmen vorauszusetzen, die noch nicht bewiesen sind (wie z.B. die Fragen, ob $\mathcal{P} \subset \mathcal{NP}$ und $\mathcal{NP} \not\subseteq \mathcal{P}$ oder ob $\mathcal{NP} \subseteq \mathcal{BPP}$) sowie möglichst niedrige Anforderungen an die eingesetzten algebraischen bzw. informationstheoretischen Verschlüsselungsverfahren zu stellen.

Im zweiten Fall wird versucht, Protokolle zu entwickeln, die die Anzahl der Nachrichten minimiert auf z.B. $O(\log |x|)$ oder eine konstante Rundenzahl.

Um die unterschiedlichen Zero–Knowledge Arguments, die in der Literatur bisher veröffentlicht wurden, unterteilen zu können, wird in Anlehnung an Kapitel 3.2 entsprechend den algebraischen bzw. informationstheoretischen Voraussetzungen und Annahmen bezüglich der Verschlüsselung nach der Sicherheit für den Prover, keine ungewollten Informationen zu veröffentlichen, unterschieden. Die nachfolgend dargestellten Protokolle werden daher untergliedert nach

- berechenbaren Zero–Knowledge Verfahren sowie
- perfekten Zero–Knowledge Verfahren.

Zur Frage, welches Protokoll die bessere Wahl ist, hat Fortnow in [For87] folgendes bewiesen: Eine statistische oder perfekte Sicherheit des Provers verlangt zwangsläufig einen beliebig mächtigen Prover (wobei ein exponentiell mächtiger ausreichend ist), der den Verifier täuschen kann. Daher ist es unwahrscheinlich, dass ein perfekter Zero–Knowledge Proof für ein $\mathcal{NP}$-vollständiges Problem existiert.

Aus naheliegenden Gründen wird daher in beide Richtungen geforscht, wobei der Schwerpunkt der informationstheoretischen, wissenschaftlichen Forschung bei berechenbaren Zero–Knowledge Verfahren liegt, da mit diesen $\mathcal{NP}$-vollständige Berechenbarkeits- und Komplexitätsuntersuchungen mit neuen „Werkzeugen“ möglich sind. Diese Verfahren können selbstverständlich auch in der Praxis eingesetzt werden, verlangen dann aber eine Abschätzung über die Wahl des Sicherheitsparameters, um das Geheimnis des Provers für eine überschaubare Zeit sicherzustellen. Für die eher praxisrelevante Forschung, die eine (zeitlich) unbegrenzte Sicherheit des Provers beabsichtigt, stehen dagegen die perfekten Zero–Knowledge Verfahren im Mittelpunkt, auch wenn damit möglicherweise keine $\mathcal{NP}$-vollständigen Verfahren definiert werden können.

Eine weitere Einteilung der unterschiedlichen Zero–Knowledge Argument Verfahren wird in der Literatur nach der Anzahl der benötigten Runden gemäß Definition 2.7 vorgenommen. Seien (P, V) ein Zero–Knowledge Argument und f eine positive Funktion gemäß Definition 2.7.

- Wenn $f \in O(1)$ gilt, dann hat (P, V) eine *konstante Rundenzahl*.
- Sei p ein positives, wachsendes Polynom und gilt $f \in O(p)$, dann hat (P, V) eine *polynomielle Rundenzahl*.

Aus naheliegenden Gründen ist eine überpolynomielle Komplexität in Bezug auf die Anzahl der Runden nicht von Interesse der Forschung. Andererseites fallen alle Verfahren, die nicht in konstanter (also auch sub-polynomieller) Rundenzahl arbeiten, in die Klasse der polynomiellen Verfahren.

Zu dem Komplex der berechenbaren Zero–Knowledge Arguments mit polynomieller Rundenzahl sind keine wissenschaftlichen Arbeiten vorhanden. Daher werden im nachfolgenden Text aufgrund der Einteilung in

1. perfekte Zero–Knowledge Arguments mit polynomieller Rundenzahl,
2. perfekte Zero–Knowledge Arguments mit konstanter Rundenzahl und
3. berechenbare Zero–Knowledge Arguments mit konstanter Rundenzahl

hauptsächlich die Arbeiten von

- Naor, Ostrovsky, Venkatesan und Yung [NOVY92] in Abschnitt 5 ab Seite 41 als Vertreter von Punkt 1,
- Brassard, Crépeau und Yung [BCY91] in Abschnitt 6 ab Seite 53 als Vertreter von Punkt 2 und
- Barak [Bar03] in Abschnitt 7 ab Seite 72 als Vertreter für den letzten Punkt

ausführlich vorgestellt.

Aus dem Vorhandensein von Protokollen mit konstanter Rundenzahl, die vordergründig besser sind, als diejenigen mit polynomieller Rundenzahl, kann aber nicht der Schluss gezogen werden, dass Protokolle mit polynomieller Komplexität überflüssig sind. Der Vorteil der konstanten Rundenzahl ist mit höheren Anforderungen an die Leistungsfähigkeit der Teilnehmer des interaktiven Systems verbunden, die in einer geplanten Umgebung eventuell nicht realisiert werden kann. In diesen Fällen wird die erfordeliche Sicherheit über polynomielle Rundenzahlen, zu der auch z.B. logarithmische gehören, erzielt.

Neben den Eigenschaften perfekt Zero–Knowledge in polynomieller Rundenzahl zu sein ist die Arbeit [NOVY92] typisch für eine in der modernen Forschung angewandten Technik. Grob gesagt geht es in dieser Veröffentlichung darum, dass kein vollständiges Protokoll eines Zero–Knowledge Arguments vorgestellt sondern der erstmals in [GMW86] bewiesene Umstand ausgenutzt wird, dass es ausreichend ist, ein sicheres Verschlüsselungsverfahren zu konstruieren.

Die Besonderheit bei [BCY91] besteht darin, dass es die letzte vollständige Konstruktion eines in sich geschlossenen perfekten Zero–Knowledge Arguments mit konstanter Rundenzahl ist. Neuere Arbeiten verfolgen diesbezüglich den Weg, der auch in [NOVY92] gegangen wurde, indem lediglich ein entsprechendes Verschlüsselungsverfahren meistens in Form von Bit–Commitment–Schemes erzeugt wird. Bei [BCY91] ergeben sich dagegen die Konstruktion und die einzelnen Beweise noch in einem Zusammenhang.

Wie schon an den Veröffentlichungsterminen zu sehen ist, beschäftigt sich die informationstheoretische Forschung im Bereich der Zero–Knowledge Arguments in den letzten Jahren zum weitaus überwiegenden Teil mit berechenbaren Zero–Knowledge Verfahren. Zu diesem Themenkomplex existieren eine Vielzahl von Veröffentlichungen. Zur Darstellung der aktuellen Forschung wurde als Vertreter die Arbeit von Barak ausgewählt. Dieses erfolgt unter anderem aufgrund der Tatsachen, dass diese Arbeit sehr aktuell ist, einen völlig anderen und für Änderungen offenen Weg der Konstruktion und des Beweises aufweist, sich mit der *nicht Black-Box* Simulation des Verifieres auseinandersetzt und seit dem ersten Erscheinungstermin 2001 als Grundlage weiterer (auch Zero–Knowledge Proof) Verfahren gedient hat.

Für die drei Arbeiten gilt, dass alle einen völlig unterschiedlichen Weg der Konstruktion und der anschließenden Beweisführung gegangen sind, so dass über die sehr detaillierte Darstellung ein Einblick in diese sehr verschiedenen Techniken erfolgt.

5 Perfekte Zero–Knowledge Arguments mit polynomieller Rundenzahl

Als Hauptvertreter dieser Verfahren wird das 1992 von Naor, Ostrovsky, Venkatesan und Yung in [NOVY92] vorgestellte **Perfect Zero–Knowledge Arguments for $\mathcal{NP}$** dargestellt. Die Ausführungen dazu sind zum größten Teil dieser Arbeit entnommen.

Es wird kein vollständiges Zero–Knowledge Argument Verfahren von den Autoren vorgestellt. Vielmehr beschränken sie sich darauf, ein Bit–Commitment–Scheme auf beliebigen Einweg–Permutationen als kryptographisches Primitiv zu erzeugen. Es wird gezeigt, dass entsprechend sichere Verschlüsselungsfunktionen existieren,[13] die dann gemäß [GMW86, IY87, BCC88] auf Zero–Knowledge Arguments reduzierbar sind. So kann das in [NOVY92] vorgestelte Bit–Commitment–Scheme in das Verfahren, welches in Abschnitt 3.3 ab Seite 31 vorgestellt wurde, als nachgewiesene Einweg–Permutation eingesetzt werden.

Das vorliegende Schema ist in dem Sinne effizient, dass die Teilnehmer des interaktiven Beweises diesen in polynomieller Zeit wärend des Protokolls ausführen können. Dabei wird folgende Sicherheit erreicht: Um zu täuschen oder ein falsches Theorem zu beweisen muss der Prover *on-line* während der Kommunikation kryptographische Voraussetzungen brechen, wohingegen der Verifier niemals und uneingeschränkt im informationstheoretischen Sinn irgendeine Information erlangen kann.

Bis zur Veröffentlichung 1992 waren die Komplexitätsvoraussetzungen, die für ein perfektes Zero–Knowledge Argument benötigt wurden, sehr hoch – sie setzten spezifische algebraische Verfahren voraus. Das stand im Gegensatz zum Zero–Knowledge Proof, der zu dem Zeitpunkt lediglich auf beliebigen Einweg–Funktionen basieren konnte. In der Veröffentlichung wurde folgendes Resultat gezeigt:

Satz 5.1 (Hauptergebnis)
Wenn Einweg–Permutationen existieren, dann ist es mit polynomiell–beschränkten Maschinen möglich, perfekte Zero–Knowledge Arguments für alle Sprachen in $\mathcal{NP}$ in polynomieller Rundenzahl auszuführen.

In dem Beweis wird ein informationstheoretisch sicheres Bit–Commitment–Scheme konstruiert. Gemäß Anmerkung nach Definition 2.16 handelt es sich um ein Schema mit perfekter Geheimhaltung und lediglich berechenbarer Eindeutigkeit. In einer praktischen Anwendung könnte dieses sichere Zero–Knowledge Argument Verfahren z.B. mit Nachfolgern des bekannten DES-Verfahrens[14] implementiert werden.

[13] Diese Existenz ist selbstverständlich nur unter der Annahme gewährleistet, dass $\mathcal{N} \neq \mathcal{NP}$ gilt.

[14] Bei dem DES-Verfahren (**D**ata **E**ncryption **S**tandard) handelt es sich um ein Blockchiffrierungsverfahren für 64 Bit große Blöcke, welches von IBM entwickelt und 1977 erstmals vom National Bureau of Standards (US) veröffentlicht wurde. Der DES wurde und wird immer noch hautpsächlich in

5.1 Hintergrund

Das Protokoll hat zwei Stufen. Zuerst wird gezeigt, wie ein informationstheoretisch sicheres Bit–Commitment–Scheme zwischen zwei polynomiell beschränkten Maschinen entworfen wird, welches auf jeder Einweg–Permutation aufbaut. Die Grundidee dazu wurde, jedoch unter anderen Voraussetzungen, von Ostrovsky, Venkatesan und Yung in [OVY91] für unbeschränkte Maschinen veröffentlicht. Über dieses Verfahren hinaus ist das jetzt vorgestellte Protokoll so beschaffen, dass es in einer erwarteten polynomiellen Zeit simulierbar ist. Danach wird eine Reduktion des „perfekt-sicheren simulierbaren Bit–Commitment–Schemes“ auf „perfekte Zero–Knowledge Argument“ angewendet. Das grundsätzliche Schema, um verschiedene Commitments mit verschiedenen Zero–Knowledge Systemen zu verbinden, wurde u.a. [IY87] entnommen.

In dem vorgestellten Protokoll werden ausschließlich polynomiell–beschränkte Teilnehmer vorausgesetzt. Es wurde ausdrücklich als Modell für die kryptographische *Anwendung* entwickelt. Ferner werden keine Trapdoor Eigenschaften benutzt, da Bit–Commitments und sichere interaktive Beweise keine Entschlüsselung benötigen sondern lediglich die Urbilder der Commitments beweisen sollen.

5.2 Modell und Definitionen

Die zugrunde liegende Arbeit [NOVY92] beschreibt kein „vollständiges“ Zero–Knowledge Argument, sondern lediglich den dafür wichtigsten Teil der Bit–Commitments für eine Reduktion auf ein Zero–Knowledge Argument. Die Teilnehmer des Bit–Commitment-Protokolls werden als Sender S und Empfänger E bezeichnet. In einer tatsächlichen Implementation eines Zero–Knowledge Arguments mit diesem Bit–Commitment–Scheme kann daher der Prover P als Sender oder Empfänger der Commitments vorkommen. Dieses ist abhängig davon, wie und für welchen Zweck das vorgestellte Bit–Commitment–Scheme in einem Zero–Knowledge Argument eingesetzt wird.

Zugrunde liegt ein interaktives Argumentsystem mit Zusatzeingabe gemäß Abschnitt 12, Seite 37, so dass beide Maschinen polynomiell–beschränkt sind. Es wird unterstrichen, dass S zwar nur polynomielle Zeit benötigt, um das Protokoll zu erfüllen. Die Sicherheit bleibt aber auch erhalten, wenn S unbegrenzte Rechenkapazität hätte.

Die Zusatzeingabe für den Sender S liegt in Form eines „Zeugnisses“ für die Behauptung vor. Der Empfänger E erhält keine Zusatzeingabe.

O.b.d.A wird hier angenommen, dass der Beweis eine Lösung für ein beliebiges SAT-Problem darstellt (andernfalls kann jedes $\mathcal{NP}$-Problem zusammen mit dem Zeugnis als Zusatzeingabe erst auf SAT reduziert werden). Am Ende des Protokolls akzeptiert E den Beweis oder weist ihn zurück.

finanziellen Transaktionsprotokollen eingesetzt, obwohl er nicht mehr als sicher gilt und Ende der 90er Jahre des letzten Jarhunderts bereits innerhalb weniger Tagen berechnet werden konnte.

Die Zero–Knowledge Eigenschaften des Protokolls entsprechen der Definition 3.1 mit folgenden Änderungen:

- Die Maschine M^* hat lediglich eine *erwartete* polynomielle Laufzeitbeschränkung.
- $\{\mathsf{view}_{E^*}^{S}(x)\}_{x\in L}$ und $\{M^*(x)\}_{x\in L}$ sind identisch.

Das verwendete Bit–Commitment–Scheme entspricht dem Grundprinzip aus Definition 2.16. Es wird nachfolgend konkretisiert:

Definition 5.1 (Perfekt sicheres Commitment–Scheme)
Sei (S, E) ein interaktives Beweissystem, $\sigma \in \{0,1\}$, p ein beliebiges Polynom und n ein genügend großer Sicherheitsparameter. Ein Protokoll heißt ***perfekt sicheres Commitment–Scheme****, wenn gilt:*

- *Geheimhaltung: Wenn S dem Protokoll folgt, kann E nach der Festlegungsphase $\{\mathsf{view}_{E}^{S(\sigma)}(n)\}_{n\in\mathbb{N}}$ mit keiner größeren Wahrscheinlichkeit als $\frac{1}{2} + \frac{1}{p(n)}$ schätzen.*
- *Festlegung: Wenn E dem Protokoll folgt, kann S nach der Festlegungsphase mit einer Wahrscheinlichkeit von mindestens $1 - \frac{1}{p(n)}$ nur einen möglichen Wert aufdecken.*

Anmerkung: Man beachte, dass für die Definition des perfekt sicheren Commitment–Schemes der Empfänger E *nicht* polynomiell beschränkt sein muss. Es wird betont, dass die Geheimhaltung nicht darauf aufbaut, dass E polynomiell beschränkt ist. Reicht eine polynomielle Rechenzeit für E jedoch aus, so wird das Verfahren „effizient“ genannt, worauf sich das vorliegende Protokoll konzentriert.

Bei der Definition der Eigenschaften, die das Bit–Commitment haben muss, wurde ein Szenario vorausgesetzt, in dem $\{\mathsf{view}_{E}^{S}(n)\}_{n\in\mathbb{N}}$ von E nicht mit einer höheren Wahrscheinlichkeit als $\frac{1}{2}$ geschätzt werden kann. Im Allgemeinen[15] bekommt E während der Kommunikation aber möglicherweise Informationen übermittelt, die es E erlauben könnten, mit einer Wahrscheinlichkeit größer als $\frac{1}{2}$ zu schätzen. Die Definition für diesen Fall sieht vor, dass der Vorteil, den E als Ergebnis der Festlegungsphase gewinnt, kleiner als $\frac{1}{p(n)}$ sein muss. Alle Resultate des vorgestellten Protokolls umfassen diesen allgemeinen Fall.

Definition 5.2 (Polynomzeit simulierbar)
Ein Commitment–Scheme wird als in Polynomzeit simulierbar *(im Hinblick auf den Empfänger) bezeichnet, wenn für einen Empfänger E^* die Wahrscheinlichkeitsverteilung $\{\mathsf{view}_{E^*}^{S}(x)\}_{x\in L}$ als Teil von $\{M^*(x)\}_{x\in L}$ in erwarteter Polynomialzeit gemäß Zero–Knowledge Definition simuliert werden kann.*

[15] Da das vorliegende Protokoll für alle Einweg–Permutationen gültig ist, sind diesbezüglich keine Einschränkungen möglich.

5.3 Perfekt–sicheres simulierbares Bit–Commitment

Das perfekt–sichere Schema wird nachfolgend aufgeführt und dessen Sicherheit bewiesen. Grob skizziert funktioniert es wie folgt. Der polynomielle Sender S generiert eine Bit-Verschlüsselung, die von zwei möglichen Verteilungen stammt. Der Sender wird die Verschlüsselung aber nur als Element einer der beiden Verteilungen öffnen können, auch wenn die Verteilungen identisch sind.

Das Schema baut auf beliebige Einweg–Permutationen auf.

Protokoll 5.1 (Perfekt–sicheres simulierbares Bit–Commitment)
Sei (S, E) ein interaktives Beweissystem mit Zusatzeingabe, f eine starke Einweg–Permutation auf $\{0,1\}^n$, $\sigma \in \{0,1\}$ das nur S bekannte Bit und $B(x,y)$ das Skalarprodukt mod 2 von x und y.

Festlegungsphase:

1. Der Sender S wählt ein zufälliges $x \in_R \{0,1\}^n$ und berechnet $y = f(x)$. S hält sowohl x als auch y vor E geheim.
2. Der Empfänger E wählt $h_1, \ldots, h_{n-1} \in \{0,1\}^n$ so, dass jedes h_i einen zufälligen Vektor über $GF(2)$ in der Form $0^{i-1}1\{0,1\}^{n-i}$ darstellt. Zufällig in diesem Sinn bedeutet, dass nach $i-1$ Nullen eine 1 folgt, an die sich eine beliebige Wahl für die letzten $n-i$ Positionen anschließt. Die $h_1, \ldots, h_{n-1}$ sind linear unabhängig über $GF(2)$.
3. Für j von 1 bis $n-1$ gilt
   - E sendet h_j an S.
   - S berechnet $\eta_j = B(h_j, y)$ und sendet η_j an E.
4. Zu diesem Zeitpunkt existieren genau zwei Vektoren $y_0, y_1 \in \{0,1\}^n$, die $\eta_j = B(h_j, y_i)$ erfüllen[16] mit $i \in \{0,1\}$ und $1 \leq j \leq n-1$. Dabei wird y_0 als der lexikographisch kleinere der beiden Vektoren definiert. S und E berechnen beide y_0 und y_1.
   Sei
   $$\eta = \begin{cases} 0 & \text{wenn } y = y_\sigma \\ 1 & \text{wenn } y = y_{1-\sigma} \end{cases}$$
5. S berechnet η und sendet es an E.

Öffnungsphase:

1. S sendet σ und x an E.

[16] Dass zwei Vektoren $\eta_j = B(h_j, y_i)$ erfüllen, liegt an der Eigenschaft von $B(x,y)$ und daran, dass die $h_1, \ldots, h_{n-1}$ linear unabhängig über GF(2) sind.

2. E berechnet $y = f(x)$ und überprüft,
   - ob die $B(h_j, y), 1 \leq j \leq n-1$, den von S in der Festlegungshase Pkt. 3 übersandten η_j entsprechen und
   - ob $y = y_\sigma$, falls $\eta = 0$ bzw. ob $y = y_{1-\sigma}$, falls $\eta = 1$ ist .

Es ist klar, dass das Protokoll in Polynomialzeit von beiden Parteien ausgeführt werden kann. Nachfolgend wird gezeigt, dass es tatsächlich ein perfekt sicheres Bit–Commitment-Protokoll ist.

5.4 Beweis der Sicherheit

Als erstes wird die grundlegende Eigenschaft als Bit–Commitment–Scheme bewiesen.

Satz 5.2
Falls eine Einweg–Permutation f existiert, dann ist das Schema aus Protokoll 5.1 ein perfekt–sicheres berechenbar–eindeutiges Bit–Commitment–Scheme.

Der Beweis folgt aus den beiden folgenden Sätzen, dem *Sicherheits-Satz* und dem *Eindeutigkeits-Satz*.

Satz 5.3 (Sicherheits-Satz)
Nach der Festlegungsphase ist für jeden Empfänger E^* das Bit σ informationstheoretisch sicher verborgen.

Beweis:
Induktiv kann über j geprüft werden, dass für jede Wahl der $h_1, \ldots, h_j$ die bedingte Wahrscheinlichkeit von y über $h_1, \ldots, h_j$ und $\eta_1, \ldots, \eta_j$ in dem von $h_1, \ldots, h_j$ und $\eta_1, \ldots, \eta_j$ definierten Unterraum gleichverteilt ist. Somit ist die Wahrscheinlichkeit in der Festlegungsphase bei Pkt. 4, dass $y = y_0$ ist, exakt $\frac{1}{2}$. Daher wird mit η keine Information über σ verbreitet. ■

Satz 5.4 (Eindeutigkeits-Satz)
Angenommen, es existiert eine probabilistische polynomiell–beschränkte Maschine $S^*(n)$, die dem Protokoll in der Festlegungsphase folgt, und einem ehrlichen Empfänger für ein σ zwei unterschiedliche Werte mit nichtvernachlässigbarer Wahrscheinlichkeit (bezüglich der Münzwürfe) $\varepsilon = \varepsilon(n)$ offenbarte. Dann existierte auch eine probabilistische polynomiell–beschränkte Maschine A, die für eine nichtvernachlässigbare Anzahl von $y \in \{0,1\}^n$ f invertieren kann.

Beweis:
Es wird eine Maschine A zum Invertieren von f unter Anwendung eines derartigen S^* konstruiert. A hat eine feste Polynomialzeit-Grenze und bricht die Berechnung ab,

wenn die Grenze überschrittten wird. Unter der Annahme, dass eine Menge Ω von $\varepsilon(n)$ Bruchteilen von Zeichenketten existiert, und unter der Voraussetzung, dass S^* mit $\omega \in \Omega$ initialisiert wird, kann S^* zwei unterschiedliche Werte von σ nach der Festlegungsphase von $n-1$ Runden aufdecken. Mit der Festlegung eines beliebigen aber festen ω wird S^* als deterministische Maschine betrachtet. Das ist möglich, weil A mit dem Zufallsband von S^* wiederholt ausgeführt werden kann und initialisiert wird mit $\omega_i, i := 1, ..., m = \frac{1}{\varepsilon^2}$ und der Wahrscheinlichkeit der $\omega_i \in \Omega$ von $1 - e^{-\sqrt{m}}$. Daher wird S^* nachfolgend als determinstischer Algorithmus betrachtet.

Die Antworten η_i von S^* auf die Fragen h_i von E definieren einen Baum T mit Wurzel, dessen Kanten aus $\{0,1\}$ markiert werden. Ein Pfad von der Wurzel zu einem Blatt ist durch die Zuweisung von $h_1, \ldots, h_{n-1}$ definiert und wird mit den $\eta_1, \ldots, \eta_{n-1}$ gekennzeichnet. Ein Knoten U in Stufe i entspricht einem Zustand von S^* nach $i-1$ Schritten und wird definiert durch $h_1, \ldots, h_{i-1}$ und $\eta_1, \ldots, \eta_{i-1}$. Die abgehenden Kanten von U entsprechen den 2^{n-i} möglichen Fragen von E. Diese Kanten werden mit den Antworten von S^* markiert. Zu beachten ist, dass die Antwort aufgrund einer möglichen Täuschung seitens S^* nicht konsistent sein muss und auf dieselbe Frage unterschiedliche Antworten in Abhängigkeit der vorhergehenden Fragen gegeben werden könnten.

Bezeichne u ein Blatt, dann ist $\{y_o(u), y_1(u)\}$ die Menge, die aus den Antworten von S^* folgt. u wird als *gut* bezeichnet, wenn gilt: Sei u durch die Fragen von E festgelegt. Dann kann S^* das Bit–Commitment auf zwei unterschiedliche Wege öffnen; S^* invertiert sowohl $y_0(u)$ als auch $y_1(u)$.

Protokoll 5.2 (Beschreibung von A)
A erhält als Eingabe eine zufällige Zeichenkette $y \in \{0,1\}^n$ und versucht y zu invertieren. Um $f^{-1}(y)$ zu berechnen, versucht A, ein gutes Blatt u zu finden, so dass $y \in \{y_0(u), y_1(u)\}$ gilt. An der Wurzel beginnend entwickelt A Knoten für Knoten einen Pfad, der zu y konsistent ist. Sei j fest gewählt mit $n - 8\left(\frac{\log}{\varepsilon} + 1\right)$. Für j Runden führt A folgendes aus: Für $1 \leq i < j$ sind in Runde i die $h_1, \ldots, h_{i-1}$ definiert, so dass die Marken $\eta_1, \ldots, \eta_{i-1}$ mit $\eta_i = B(h_i, y)$ bestehen. Dann wird ein zufälliges h aus $0^{i-1}1\{0,1\}^{n-i}$ gewählt.

Anmerkung: h ist linear unabhängig von h_k für $k < i$. Wenn die Kante h die Markierung $B(h, y)$ hat, wird $h_i = h$ und der Pfad wird durch den neuen Knoten erweitert. Andernfalls wird S^* auf den Zustand vor seiner Antwort zurückgesetzt und ein neuer Kandidat für h_i wird ausgewählt. Das wird solange wiederholt, bis entweder ein erfolgreiches h_i gefunden wurde oder kein Kandidat mehr übrig ist. Im diesem Fall bricht A die weitere Berechnung ab. Wenn A die j-te Stufe erreicht, schätzt A die restlichen $n-j$ Fragen $h_j, h_{j+1}, \ldots, h_{n-1}$ und prüft, ob der Pfad zu dem Blatt zu $B(y, h_i)$ konsistent markiert ist. Wenn das der Fall und das erreichte Blatt gut ist, konnte A y invertieren.

Der Rest dieses Beweises widmet sich dem Nachweis, dass die so definierte Maschine A mit einer Wahrscheinlichkeit von mindestens $\frac{\varepsilon^{10}}{8e^3n^8}$ y invertiert. Dabei ist zu beachten, dass A nicht unbedingt nach einer polynomiellen Anzahl von Schritten hält. Trotzdem wird am Ende des Beweises gezeigt, dass die Anzahl der vergeblichen Versuche, ein konsistentes h zu finden, auf $8n$ beschränkt werden kann, ohne signifikant die Wahrscheinlichkeit zu verringern, dass A y invertieren kann.

Zuerst erfolgen ein paar Notationen. Da verschiedene Typen von Vektoren der Länge n über $GF(2)$ benutzt werden, werden diese wie folgt unterschieden: Von E gesendete Vektoren werden als *Fragen* und die von S gesendeten als *Bilder* bezeichnet. Sei U ein Knoten an der i-ten Stufe des Baums, definiert durch $h_1, \ldots, h_{i-1}$ und $\eta_1, \ldots, \eta_{i-1}$. Dann wird $y \in \{0,1\}^n$ ein Bild von U genannt, wenn $B(h_k, y) = \eta_k$ für alle $1 \leq k < i$. Die Menge aller Bilder von U wird mit $\mathcal{I}(U)$ bezeichnet.[17] Es gilt $|\mathcal{I}(U)| = 2^{n-i+1}$. $h \in \{0,1\}^n$ wird als Frage von U bezeichnet, wenn h die Form $0^{i-1}1\{0,1\}^{n-i}$ hat.

Sei $A(U,y) = \big|\{h \mid h \text{ ist Frage von } U \text{ und } B(h,y) \text{ entspricht Markierung } h \text{ von } U\}\big|$. Ein Bild y heißt *ausgeglichen* in U_i, einem Knoten der i-ten Stufe, wenn gilt

$$1 - \frac{1}{n} \leq \frac{A(U_i, y)}{2^{n-i-1}} \leq 1 + \frac{1}{n}$$

Ein Bild y heißt *voll ausgeglichen* in U, einem Knoten der j-ten Stufe, wenn es in allen Vorfahren von U ausgeglichen ist. Weiterhin sei $\mathcal{F}(U) = \{y \mid y \in \mathcal{I}(U) \text{ und } y \text{ ist voll ausgeglichen in } U\}$.[18] Für eine Menge H von Fragen am Knoten U und einem Bild y von U ist die *Abweichung* von y über H die absolute Differenz zwischen $\frac{|H|}{2}$ und der Anzahl von Fragen in H, die mit y übereinstimmen. Es sei wiederholt, dass $j = n - 8\left(\frac{\log n}{\varepsilon} + 1\right)$ ist.

Lemma 5.1
Für jeden Knoten U der Stufe j haben mindestens $2^{n-j}(1-\beta)$ mit $\beta = 2^{\frac{-3}{4(n-j)}}$ der Bilder von U die Eigenschaft $2^{n-j} - 2^{\frac{7}{8(n-j)}} \leq A(U,y) \leq 2^{n-j} + 2^{\frac{7}{8(n-j)}}$.

Beweis:
Einleitend sei bemerkt, dass jedes Paar Fragen h', h'' von U im Sinne von h'' und $h', h_1, \ldots, h_{j-1}$ linear unabhängig ist. Sei ein Bild y von U zufällig gewählt und gelte $a_h = 1$, wenn $B(h,y)$ und U's Antwort auf h identisch sind. Dann gilt für alle h, $\Pr[a_h = 1] = \frac{1}{2}$ und für jedes Paar (h', h''), $a_{h'}$ und $a_{h''}$ sind paarweise unabhängig. Nun interessiert vor allem

$$\Pr\left(\left|\sum_{h \text{ Frage von } U} a_h - E\left[\sum_{h \text{ Frage von } U} a_h\right]\right| \geq 2^{\frac{7}{8(n-j)}}\right) \tag{1}$$

[17] $\mathcal{I}$ als Menge der Bilder, engl. **I**mages.
[18] $\mathcal{F}$ als Menge der voll ausgeglichenen Bilder, engl. **F**ully balanced

Mit der Tschebyschev'schen Ungleichung ergibt sich

$$\Pr\left(\left|\sum_{h \text{ Frage von } U} a_h - E\left[\sum_{h \text{ Frage von } U} a_h\right]\right| \geq \lambda\sqrt{Var\left[\sum a_h\right]}\right) \leq \frac{1}{\lambda^2}$$

$Var[\sum_h a_h]$ ist dabei 2^{n-j} und daher ist (1) höchstens $2^{\frac{-3}{4(n-j)}}$. □

Lemma 5.2
Für jeden Knoten U der Stufe j und jedes zufällige Bild y von U gilt: Die Wahrscheinlichkeit, dass y voll ausgeglichen ist, beträgt mindestens $1-\gamma$ für $\gamma = n2^{\frac{-5}{8(n-j)}}$.

Beweis:
Seien $U_1, \ldots, U_j = U$ die Knoten auf dem Pfad nach U. Für jedes $1 \leq i \leq j$ werden die 2^{n-i} Fragen von U_i in 2^{j-i} Teilmengen $H_1, \ldots, H_{2^{j-i}}$ mit der Größe 2^{n-j} unterteilt, so dass für jedes $1 \leq \ell \leq 2^{j-i}$ und $h', h'' \in H_\ell$ gilt, dass die h' linear unabhängig von den $h_{i+1}, \ldots, h_j, h''$ sind. Entsprechend Lemma 5.1 gilt daher $\Pr[|\sum_{h\in H_\ell} - E[\sum_{h\in H_\ell} a_h]| > 2^{\frac{7}{8(n-j)}} \leq 2^{\frac{-3}{4(n-j)}}$, so dass nach der Markov Ungleichung die Wahrscheinlichkeit, dass mehr als $2^{\frac{-1}{8(n-j)}}$ Anteile der H_ℓ's eine Abweichung größer als $2^{\frac{7}{8(n-j)}}$ haben, höchstens $2^{\frac{-5}{8(n-j)}}$ beträgt. Damit beträgt mit einer Wahrscheinlichkeit von mindestens $1-2^{\frac{-5}{8(n-j)}}$ die totale Abweichung am Knoten U_i höchstens

$$2^{\frac{-1}{8(n-j)}}2^{n-j}2^{j-i} + (1-2^{\frac{-1}{8(n-j)}})2^{\frac{7}{8(n-j)}}2^{j-i} \leq 2\cdot 2^{\frac{7}{8n}+\frac{1}{8j}-i} \tag{2}$$

und schließlich mit einer Wahrscheinlichkeit von mindestens $1-2^{\frac{-5}{8(n-j)}}$

$$1-\frac{1}{n} \leq 1-2^{\frac{-1}{8(n-j)+1}} \leq \frac{A(U_i,y)}{2^{n-i-1}} \leq 1+2^{\frac{-1}{8(n-j)+1}} \leq 1+\frac{1}{n}\ .$$

Die Wahrscheinlichkeit, dass y in allen Stufen ausgeglichen ist, beträgt daher mindestens $1-n2^{\frac{-5}{8(n-j)}} = 1\gamma$ □

Lemma 5.3
Die Wahrscheinlichkeit, dass ein Knoten U auf Stufe j von einer Programmausführung durch A erreicht wird, beträgt $\frac{1-\gamma}{e}$ der Wahrscheinlichkeit, dass U von einer Programmausführung durch S^* erreicht wird.

Beweis:
Seien $U_1, \ldots, U_j = U$ die Knoten auf dem Pfad von der Wurzel zu U. Die Wahrscheinlichkeit für jeden Knoten U_i, dass dieser in S^* erreicht wird, beträgt $\prod_{i=1}^{j-1}\frac{1}{2^{n-i}}$. Es gilt

$$\begin{aligned}
\Pr[U \text{ wird erreicht von } \mathsf{A}] &= \sum_{y \in \mathcal{I}(U)} \Pr[y \text{ gewählt, } U \text{ wird erreicht}] \\
&\geq \sum_{y \in \mathcal{F}(U)} \Pr[y \text{ gewählt, } U \text{ wird erreicht}] \\
&\geq \sum_{y \in \mathcal{F}(U)} \frac{1}{2^n} \prod_{i=1}^{j-1} \frac{1}{A(U_i, y)} \\
&\geq \sum_{y \in \mathcal{F}(y,U)} \frac{1}{2^n} \prod_{i=1}^{j-1} \frac{1}{(1+\frac{1}{n})2^{n-i-1}} \\
&\geq \sum_{y \in \mathcal{F}(U)} \frac{1}{2^n} \prod_{i=1}^{j-1} \frac{1}{(1+\frac{1}{n})2^{n-i-1}} \\
&\geq \frac{2^{n-j+1}(1-\gamma)}{2^n} \cdot \frac{1}{(1+\frac{1}{n})^n} \prod_{i=1}^{j-1} \frac{1}{2^{n-i-1}} \\
&\geq \frac{(1-\gamma)}{e} \prod_{i=1}^{j-1} \frac{1}{2^{n-i}}
\end{aligned}$$

□

Lemma 5.4
Die Wahrscheinlichkeit, dass das Bild, welches A zu invertieren versucht, auf der Stufe j voll ausgeglichen ist, beträgt mindestens $\frac{(1-\gamma)^2}{e}$.

Beweis:
Für jeden Knoten auf Stufe j und jedes voll ausgeglichenes Bild y von U beträgt die Wahrscheinlichkeit $\Pr[y \text{ ist gewählt und } U \text{ wird erreicht}] \geq \frac{(1-\gamma)}{2^n e} \prod_{i=1}^{j-1} \frac{1}{2^{n-i-1}}$. Daher gilt

$$\begin{aligned}
\Pr[U \text{ wird erreicht mit einem voll ausgeglichenen } y] &\geq 2^{n-j}(1-\gamma) \cdot \frac{1-\gamma}{2^n e} \prod_{i=1}^{j-1} \frac{1}{2^{n-i-1}} \\
&= \frac{(1-\gamma)^2}{e} \prod_{i=1}^{j-1} \frac{1}{2^{n-i}}
\end{aligned}$$

Die Anzahl der Knoten auf der j-ten Stufe beträgt $\prod_{i=1}^{j-1} 2^{n-i}$ und daher ist die Wahrscheinlichkeit, dass das gewählte Bild auf Stufe j voll ausgeglichen ist, $\frac{(1-\gamma)^2}{e}$.

Ein Knoten wird als *gut* bezeichnet, wenn mindestens ε der Blätter des Teilbaums, der von U als neuer Wurzel aufgespannt wird, die Eigenschaft hat, dass S^* erfolgreich täuschen kann, d.h. beide Bilder invertieren kann. Gemäß Annahme ist der Anteil der guten Knoten U mindestens ε. Daher beträgt gemäß Lemma 5.3 die Wahrscheinlichkeit, dass A einen gutes U auf Stufe j erreicht, mindestens $\frac{(1-\gamma)^2}{e}\varepsilon$. □

Lemma 5.5
In jedem guten U auf Stufe j beträgt der Anteil der guten Blätter, die wenigstens ein Bild aus $\mathcal{F}(U)$ haben, mindestens $\frac{\varepsilon}{2}$.

Beweis:
Jedes Paar von Bildern $y_1 \neq y_2$ in $\mathcal{I}(U)$ kann in höchstens $\frac{1}{2^{n-j}}$ der Blätter zusammen vorkommen: In jedem Knoten U' auf dem Weg von U zu den Blätter. Darüber hinaus gilt für zufällige Fragen von U' $\Pr[B(h, y_1) = B(h, y_2)] = \frac{1}{2}$. Weil höchstens $\gamma 2^{n-j+1}$ Bilder existieren, die in U nicht voll ausgeglichen sind, haben höchstens

$$\frac{\binom{\gamma 2^{n-j+1}}{2}}{2^{n-j-1}} \leq 2\gamma^2 2^{n-1} \leq n^2 2^{\frac{-1}{4(n-j)}+1} = n^2 2^{-2\frac{\log n}{\varepsilon+1}+1} \leq \frac{\varepsilon^2}{2}$$

der Blätter ihre beiden Bilder von den unausgeglichenen. Daher sind mindestens $\varepsilon - \frac{\varepsilon^2}{2} \geq \frac{\varepsilon}{2}$ der Blätter beide gut und haben mindestens ein in U voll ausgeglichenes Bild. □

Lemma 5.6
Für jeden guten Knoten U der Stufe j, der mit einem voll ausgeglichenem y erreicht wird, und für ein $z \in \mathcal{F}(U)$, ist die Wahrscheinlchkeit, dass $y = z$ gilt, mindestens $\frac{1}{e^2 2^{n-j+1}}$.

Beweis:
Für die untere Schranke gilt

$$\frac{\Pr[z \text{ ist gewählt und } U \text{ wird erreicht}]}{\Pr[U \text{ wird erreicht und das Bild ist voll ausgeglichen}]} \tag{3}$$

Weiterhin gilt

$$\begin{aligned}
\Pr[U \text{ wird erreicht und das Bild ist voll ausgeglichen}] &= \\
\sum_{z\in\mathcal{F}(U)} \Pr[z \text{ gewählt, } U \text{ wird erreicht}] &= \sum_{y\in\mathcal{F}(U)} \frac{1}{2^n} \prod_{i=1}^{j-1} \frac{1}{A(U_i,y)} \\
&\leq \sum_{y\in\mathcal{F}(U)} \frac{1}{2^n} \prod_{i=1}^{j-1} \frac{1}{(1-\frac{1}{n})2^{n-i-1}} \\
&\leq \sum_{y\in\mathcal{F}(U)} \frac{e}{2^n} \prod_{i=1}^{j-1} \frac{1}{2^{n-i-1}} \\
&\leq \frac{2^{n-j}}{2^n} \cdot e \prod_{i=1}^{j-1} \frac{1}{2^{n-i-1}} \\
&\leq e \prod_{i=1}^{j-1} \frac{1}{2^{n-i}}
\end{aligned}$$

Wie schon im Beweis zum Lemma 5.3 gezeigt wurde, gilt für alle $z \in \mathcal{F}(U)$

$$\Pr[z \text{ ist gewählt und } U \text{ wird erreicht}] \geq \frac{1}{e2^{n-j+1}} \prod_{i=1}^{j-1} \frac{1}{2^{n-i}}$$

Daher ist (3) mindestens $\frac{1}{e^2 2^{n-j+1}}$. □

Lemma 5.7
Die Wahrscheinlichkeit, das A erfolgreich ist, beträgt mindestens $\frac{\varepsilon^{10}}{4\varepsilon^3 n^8}$.

Beweis:
Angenommen, (i) dass A einen guten Knoten U der Stufe j erreicht und dass y voll ausgeglichen ist, und (ii), dass $h_j, h_{j+1}, ..., h_{n-1}$ einen Pfad zu einem guten Blatt definiert, das mindestens ein Bild in $\mathcal{F}(U)$ hat. Nach Lemma 5.6 ist bekannt, dass die Wahrscheinlichkeit für $y = z$ mindestens $\frac{1}{e^2 2^{n-j}}$ beträgt. Die Wahrscheinlichkeit, dass (i) eintritt, ist nach Lemma 5.4 mindestens $\varepsilon\frac{(1-\gamma)^2}{e}$ und dass (ii) bei gegebenem (i) eintritt, nach Lemma 5.5 mindestens $\frac{\varepsilon}{2}$. Daher beträgt die Erfolgswahrscheinlichkeit mindestens $\varepsilon^2\frac{(1-\gamma)^2}{2^{n-j}e^3} \geq \frac{\varepsilon^{10}}{4e^3n^8}$.

Anzumerken ist, dass A's Erfolg nur für den Fall betrachtet wurde, dass y auf Stufe j voll ausgeglichen ist. Dennoch ist die Wahrscheinlichkeit unter der Voraussetzung, dass y auf Stufe j voll ausgeglichen ist, für A, viele nicht erfolgreiche Kandidaten bis

zum Erreichen der j-ten Stufe verarbeiten zu müssen, sehr gering: Gemäß gegebener Voraussetzung ist y in den U_i für alle $1 \leq i < j$ ausgeglichen, so dass $\frac{A(U,y)}{2^{n-i}} > \frac{1}{4}$ gilt. Somit ist die Wahrscheinlichkeit, dass A mehr als $8n$ Kandidaten für die h_i's bis zur j-ten Stufe testen muss, in n exponentiell klein. Selbst wenn die Laufzeit von A auf $8n^2$ beschränkt ist, beträgt die Erfolgswahrscheinlichkeit weiterhin mindestens $\frac{\varepsilon^{10}}{8e^3n^8}$. Wenn ε nicht vernachlässigbar klein ist, dann ist auch diese nicht vernachlässigbar klein. □

Dieses schließt den Beweis zu Satz 5.4. ■

Für die Anwendung des Bit–Commitment–Schemes in einem Zero–Knowledge Argument ist es notwendig, dass das Commitment–Scheme simulierbar ist.

Satz 5.5
Es gibt ein simulierbares perfekt–sicheres Commitment–Scheme.

Beweisskizze:
Da alle Prozesse von S in polynomieller Zeit ablaufen, ist eine Simulierbakeit gegeben, die dieselbe Verteilung in polynomieller Zeit erzeugt. □

Nach dem Beweis eines perfekt–sicheren simulierbaren Commitment–Schemes erfolgt mit dem nächsten Satz die Anwendung des „Reduktions-Theorems“ aus den Arbeiten über perfekte Zero–Knowledge Arguments von [GMW86, IY87, BCC88].

Satz 5.6
Wenn ein perfekt–sicheres Commitment–Scheme mit einem polynomiell–beschränkten Empfänger existiert, welches von einer probabilistischen Maschine in erwarteter Polynomialzeit simulierbar ist, dann existiert ein perfektes Zero–Knowledge Argument für jede Behauptung in $\mathcal{NP}$.

Das simulierbare perfekt–sichere Bit–Commitment Protokoll kann gemäß dieses „Reduktions-Theorems“ genutzt werden, so dass unter der Voraussetzung der Existenz von Einweg–Permutationen perfekte Zero–Knowledge Arguments für alle Sprachen in $\mathcal{NP}$ existieren. Zusammengefasst ergibt sich damit der Beweis des Satzes 5.1.

6 Perfekte Zero–Knowledge Arguments mit konstanter Rundenzahl

Die einzigen Arbeiten, die perfekte Zero–Knowledge Argument in konstanter Rundenzahl vorstellen, stammen von Brassard, Crépeau und Yung [BCY89, BCY91] und bauen auf den Arbeiten von Brassard, Crépeau sowie Chaum [BC86, Cha87, BCC88] auf.

Das hier vorgestellte Protokoll entspricht der Veröffentlichung [BCY91]: **Constant-round perfect zero-knowledge computationally convincing protocols** und ist diesem zum größten Teil entnommen.

Bei diesem Verfahren wird die Sicherheit des Provers bedingungslos garantiert, auch wenn mächtige Organisationen mit unbekannter Berechnungsmöglichkeit und algorithmischen Kenntnissen versuchen, das Geheimnis des Provers zu erlangen, siehe dazu auch *Unterschied zwischen berechenbarem und perfektem Zero–Knowledge* in Abschnitt 3.2 auf Seite 31.

Die Arbeit [BCY91] besagt folgendes:

Satz 6.1 (Hauptergebnis)
Unter der Voraussetzung, dass es unmöglich ist, den diskreten Logarithmus einer natürlichen Zahl in polynomieller Zeit zu berechnen, existiert ein perfektes Zero–Knowledge Argument Protokoll mit nur drei Runden, um alle Sprachen in $\mathcal{NP}$ zu entscheiden.

Das Protokoll basiert auf der Verwendung eines Bit–Commitment–Schemes. Um nicht nur ein einzelnes Bit sondern ganze Zeichenketten gesichert übertragen zu können, bei dem die Geheimhaltung auch bei paralleler Anwendung erhalten bleibt (siehe dazu Abschnitt 2.5 ab Seite 15), wird in diesem Protokoll eine „Verzahnung“ zweier Bit–Commitment–Schemes vorgenommen.

In einem Standard Bit–Commitment–Scheme, wie es z.B. in Abschnitt 2.5 vorgestellt wurde, wird von dem Prover ein Bit verschlüsselt und dem Verifier mitgeteilt. Zu einem späteren Zeitpunkt wird das verschlüsselte Bit vom Prover aufgedeckt. Beim Einsatz in einem Zero–Knowledge Argument hängt die Öffnung eines Commitments u.a. von der *Wahl* des Verifiers ab. In dieser Wahl durch den Verifier, im weiteren Verlauf auch *Öffnungswahl* oder *Öffnungsaufforderung* (in englischen Originaltexten oft *challenge*: Herausforderung) genannt, liegt das Problem der parallelen Ausführung. In bestimmten Konstellationen kann der Verifier, nachdem er die Commitments erhalten hat, seine Wahl zur Öffnung der Commitments so treffen, dass der Verifier nach der Öffnung das Geheimnis des Provers berechnen kann.

Das Funktionsprinzip des hier vorgestellten Protokolls besteht darin, dass der Verifier dem Prover schon *vor* der Festlegungsphase mitteilt, welche Commitments später geöffnet werden sollen. Damit nun aber der Prover auch keine Täuschungsmöglichkeit hat, wird diese Wahl vom Verifier ebenfalls in einem Bit–Commitment–Scheme übermittelt.

Dieses erste Bit–Commitment–Scheme, mit dem der Verifier seine Wahl an den Prover übermittelt, muss dabei in Bezug auf die Geheimhaltung nur für die Dauer der Kommunikation sicher sein. Eine berechenbare Sicherheit reicht dabei aus, wenn aufgrund der gewählten Sicherheitsparameter gewährleistet ist, dass der Prover in den wenigen Sekunden oder Minuten bis zum Aufdecken der gewählten Bits den kryptographischen Algorithmus nicht brechen kann.

6.1 Grobkonzept des Algorithmus

Somit besteht der grundlegende Algorithmus des Protokolls von Brassard, Crépeau und Yung aus folgenden Schritten. Dabei handelt es sich nur um einen groben Überblick. Die Einzelheiten, insbesondere welche Parameter ausgetauscht werden und welche Inhalte bzw. Bedeutungen die x_k, y_k und z_k haben, werden im weiteren Verlauf dargestellt.

Protokoll 6.1 (Grobkonzept des Algorithmus)
Gemeinsame Eingabe:

- k: Sicherheitsparameter
- p und α, die nachfolgend in den zahlentheoretischen Voraussetzungen erläutert werden.

Schritt V1: Der Verifier wählt ein $s \in \mathbb{Z}_p^*$ und sendet es. Er wählt weiterhin Bits $y_1, \ldots, y_k$ und sendet sie in einem Commitment.

Schritt P2: Der Prover sendet seinen Beweis $x_1, \ldots, x_k$ in einem Commitment.

Schritt V3: Der Verifier entschlüsselt seine Öffnungsaufforderungen $y_1, \ldots, y_k$ für den Prover.

Schritt P4: Der Prover entspricht den Forderungen und sendet die gewünschten $z_1, \ldots, z_k$.

6.2 Zahlentheoretische Voraussetzungen

Das Verfahren beruht auf der Schwierigkeit, den diskreten Logarithmus einer ganzen Zahl zu berechnen, siehe Abschnitt 2.8.5 ab Seite 27. Neben der multiplikativen Gruppe $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$ wird hier zusätzlich die additive Gruppe $\mathbb{Z}_{p-1} = \{0, \ldots, p-2\}$ benötigt, in der modulo $p-1$ addiert wird. Beide Gruppen enthalten dieselbe Anzahl an Elementen. Für alle $a, b, \nu \in \mathbb{Z}$ mit der Eigenschaft $a \not\equiv 0 \pmod{p}$ und $b \equiv \nu \pmod{p-1}$ gilt nach einem Satz von Fermat $a^b \equiv a^\nu \pmod{p}$. In diesem Sinne existiert ein x^i für ein $x \in \mathbb{Z}_p^*$ und ein $i \in \mathbb{Z}_{p-1}$. Sei α ein erzeugendes Element von $\mathbb{Z}_p^*$. Dann ist die Funktion $\exp_\alpha : \mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$, die definiert wird als $\exp_\alpha(i) = \alpha^i$, eine Permutation.

Zur Vorbereitung auf das Bit–Commitment–Scheme einigen sich der Prover und Verifier auf ein große Primzahl p, für die beide die Faktorisierung von $p-1$ kennen. Derartige Primzahlen können mittels effizienter Verfahren gefunden werden. Darüber hinaus einigen sich Prover und Verifier über ein erzeugendes Element $\alpha \in \mathbb{Z}_p^*$. Aufgrund der polynomiellen Berechenbarkeit sind beide Eigenschaften von beiden Teilnehmern leicht zu prüfen.

Diese beiden Parameter α und p müssen nicht bei jeder Kommunikation geändert werden. Zusammen mit der Faktorisierung von $p-1$ können sie öffentlich bekannt sein und von einem Trust-Center verwaltet werden, wenn sie einmal überprüft wurden. Ist eine Zahl $i \in \mathbb{Z}_{p-1}$ gegeben, kann sehr leicht α^i berechnet werden. Es ist aber kein effizienter Algorithmus bekannt, diesen Prozess zu invertieren, wenn die Faktoren von $p-1$ groß genug gewählt wurden.

Um ein Bit $\nu \in \{0,1\}$ in einem Commitment zu verschlüsseln, wählt zuerst der *Empfänger* ein beliebiges $s \in \mathbb{Z}_p^*$ und sendet es dem Sender. Dieser wählt ein zufälliges $r \in_R \mathbb{Z}_{p-1}$ und berechnet $x = \alpha^r s^\nu$. Diese x ist das Commitment und wird dem Empfänger gesendet. Das r wird dabei vom Sender geheim gehalten. Soll das Commitment geöffnet werden, wird das r vom Sender an den Empfänger gesendet, der einfach $x = \alpha^r s^\nu$ überprüfen kann.

Da die $\exp_\alpha$ Funktion eine Permutation ist, kann jedes Element aus $\mathbb{Z}_p^*$ dafür genutzt werden, sowohl eine 0 als auch eine 1 in einem Commitment zu verschlüsseln, welche über $\mathbb{Z}_p^*$ gleichverteilt sind. Daher ist es für den Empfänger informationstheoretisch unmöglich, ein 0–Commitment von einem 1–Commitment zu unterscheiden.

6.3 Täuschungsmöglichkeit des Provers

Unter bestimmten Voraussetzungen kann mit dem bisherigen Verfahren der Prover immer noch betrügen. Dazu ein ein kleines Beispiel:

Beispiel 8
In Schritt 2 des Protokolls 6.1 verschlüsselt der Verifier seine Wahl, dass das Bit y später in Schritt 4 vom Prover geöffnet werden soll, mit einem zufällig gewählten $r \in_R \mathbb{Z}_{p-1}$ und berechnet $e = \alpha^r s^y$. Dieses Commitment wird an den Prover gesendet, der seinerseits ein Bit $x \neq y$ in einem Commitment mit $x = \alpha^q e$ mit einem zufällig gewählten $q \in_R \mathbb{Z}_{p-1}$ erstellt. Bis jetzt kann der Prover dieses Commitment natürlich nicht wechselseitig öffnen. Nichtsdestotrotz wird der Verifier sein r in Schritt 4 öffnen und an den Prover senden. Jetzt hat der Prover die Möglichkeit, sein Commitment von x als Bit y mittels eines „Zeugnisses“ $q+r$ zu öffnen, da $x = \alpha^{q+r} s^y$ gilt (q und r sind Elemente von $\mathbb{Z}_{p-1}$ und werden modulo $p-1$ addiert).

Dieses Problem enspringt nicht dem Protokoll 6.1 sondern wohnt der unzureichenden Definition des Bit–Commitment–Schemes inne. Es reicht nicht aus, per Definition zu

verhindern, dass ein Sender sein Commitment nicht zweideutig öffnen kann. Zusätzlich ist erforderlich, dass der Sender zum Zeitpunkt der Erstellung des Commitments weiß, wie er das Commitment öffnen kann. Mit anderen Worten darf es nicht möglich sein, ein Bit zu verschlüsseln, welches noch nicht bekannt ist.

Entsprechende Commitments werden von Brassard, Crépau und Yung als „EPR-blobs“ bzw. als EPR–Commitments bezeichnet.[19] Die Gefahr dieser EPR-Commitments besteht darin, dass diese nachträglich entweder 0–Commitments oder 1–Commitments werden können, wenn der Prover weitere Informationen vom Verifier erhält.

Es wird daher ein Unterprozess zur *Commitment Zertifizierung* eingeführt. Dieses erlaubt dem Prover den Verifier davon zu überzeugen, dass ein gegebenes Commitment nicht EPR ist, ohne irgendwelche Informationen darüber zu veröffentlichen, ob es sich um ein 0–Commitment oder ein 1–Commitment handelt.

6.4 Commitment Zertifizierung

Sei x ein nicht–EPR Commitment das *aktuelle* Commitment und $\nu \in \{0,1\}$ die Bezeichnung für x als ν–Commitment. Sei ferner z das Zeugnis des Provers, dass dieser das Commitment öffnen kann, d.h. $x = \alpha^z s^\nu$. Um den Verifier davon überzeugen zu können, dass er das Commitment x öffnen kann, erzeugt der Prover k zusätzliche Kontroll–Commitments $u_1, \ldots, u_k$, indem er k zufällige Bits $b_1, \ldots, b_k$ zusammen mit $v_1, \ldots, v_k$ in $\mathbb{Z}_{p-1}$ wählt und $u_i = \alpha^{v_i} s^{b_i}$ mit $1 \leq i \leq k$ berechnet. Diese Kontroll Commitments werden dem Verifier übergeben. An diesem Punkt wählt der Verifier zufällige Öffnungsaufforderungen $h_1, \ldots, h_k \in \{0,1\}$ und sendet diese dem Prover. Für jedes $h_i = 0$ öffnet der Prover die Kontroll Commitments u_i, indem er b_i und v_i übersendet. Für jedes $h_i = 1$ zeigt der Prover, dass er das aktuelle Commitment x dann und nur dann öffnen kann, wenn er das Kontroll Commitment u_i öffnen kann. Dies geschieht folgendermaßen:

- Falls $\nu = b_i$ ist, berechnet der Prover $w_i = v_i - z$ und sendet es dem Verifier, der $u_i = \alpha^{w_i} x$ prüft;
- Falls $\nu \neq b_i$ ist, berechnet der Prover $w_i = v_i + z$ und sendet es dem Verifier, der $u_i x = \alpha^{w_i} s$ prüft.

Beweis:
Wenn x kein EPR–Commitment ist, besteht der einzige Weg für den Prover zu täuschen darin, die $h_1, \ldots, h_k$ vorher exakt zu raten. In diesem Fall kann der Prover die Kontroll Commitments u_i vorbereiten, falls $h_i = 0$ ist oder andernfalls mit einem zufällig gewähltem $w_i \in \mathbb{Z}_{p-1}$ die u_i zu berechnen mit $u_i = \alpha^{w_i}(x \cdot (1 - b_i) + s x^{-1} \cdot b_i)$. Sollte irgendeines

[19] Diese Bezeichnung wurde in Anlehnung an ein von **E**instein-**P**odolsky-**R**osen veröffentlichtes Paradoxon der Quantenphysik gewählt.

der h_i falsch geraten worden sein, wird der Prover dabei erwischt, entweder das Kontroll Commitment nicht öffnen zu können (falls tatsächlich $h_i = 0$) oder das x nicht auf das h_i beziehen zu können (falls $h_i = 1$). Die Wahrscheinlichkeit für ein erfolgreiches Raten liegt bei 2^{-k}. Auf der anderen Seite ist offensichtlich, dass der Verifier nichts über ν durch den Wert b_i erfährt, falls $h_i = 0$ ist, bzw. ob $\nu = b_i$ ist, falls $h_i = 1$ ist. □

Es ist zu beachten, dass dieser Unterprozess parallel ausgeführt werden kann, ohne dabei seine Eigenschaften zu verlieren.

Somit kann das endgültige Protokoll beschrieben werden. Dieses entspricht dem Protokoll 6.1 mit der Ausnahme, dass die Commitment Zertifizierung als Unterprozess zwischen Schritt 2 und 3 für jedes Commitment benutzt wird, welches vom Prover ausgegeben wird. Dadurch wird es für den Prover mit einer Fehlerwahrscheinlichkeit von 2^{-k} unmöglich, mit Hilfe der Informationen des Verifiers aus Schritt 3 zu täuschen.

6.5 Vorbemerkungen zum Protokoll

Wie eingangs erwähnt wird bezüglich der Mächtigkeit des Verifiers keine Einschränkung vorgenommen. Nichtsdestotrotz muss zur Ausführung der ehrliche Verifier lediglich effizient sein, da andernfalls kein praktischer Nutzen für das Verfahren gegeben wäre. Von dem Prover wird ebenfalls keine informationstheoretisch geprägte Beschränkung wie z.B. polynomiell beschränkte Zeit verlangt, da diese Schranke lediglich asymptotisch sei. Es wird vielmehr vorausgesetzt, dass ein spezieller Teil des Protokolls existiert, den der Prover während der Laufzeit des Protokolls nicht ausführen kann.[20] Es soll damit betont werden, dass die dem Prover auferlegten Grenzen tatsächlicher und nicht nur informationstheoretischer Natur seien. Selbst unter der Voraussetzung, dass $\mathcal{P} = \mathcal{NP}$ ist, wird es für einen Prover mit einer vernachlässigbaren Fehlerwahrscheinlichkeit unmöglich sein, einen genügend großen diskreten Logarithmus während der Kommunikation mit dem Verifier zu berechnen.

Desweiteren wird im Unterschied zu vielen anderen Veröffentlichungen der Sicherheitsparameter k *nicht* in Abhängigkeit zur Behauptung des Provers als Schranke einer Funktion genutzt. Nach Ansicht von Brassard, Crépeau und Yung besteht keine direkte Verbindung zwischen der Wichtigkeit für den Verifier, nicht betrogen zu werden, und der zur Diskussion stehenden Behauptung.

Der in diesem Verfahren eingesetzte Simulator zur Überprüfung der Zero–Knowledge Eigenschaft ersetzt nicht den Verifier. Vielmehr ruft der Simulator den Verifier als Unterprozess auf und verhält sich ihm gegenüber so, als kommuniziere der Verifier direkt mit dem Prover. Zur nachvollziehbaren Steuerung hat der Simulator jedoch die volle

[20] Was nach Auffassung des Autors im Gegensatz zur Meinung von Brassard, Crépeau und Yung gerade von einer polynomiellen Schranke erfasst wird, da diese eine maximale Anzahl von Rechenschritten enthalten kann und damit der in [BCY91] nur verbal formulierten Zeitobergrenze entspricht.

Kontrolle über die Münzwürfe, die der Verifier während seiner Aktion durchführen muss. Weiterhin kann der Simulator zu jedem Zeitpunkt einen Speicherauszug des Verifiers durchführen oder ihn zu einem früheren Status zurückversetzen.[21] Somit ist gewährleistet, dass der Verifier nichts vom Prover erfährt, da dessen Geheimnis dem Simulator unbekannt ist.

Der Simulator muss zusätzlich effizient sein. Das heißt, dass die erwartete Zeit bis zur Ausgabe seiner Sicht sich nur durch einen (kleinen) konstanten Faktor von der erwarteten Zeit für die Ausgabe durch den Prover bei der Kommunikation mit dem Verifier bezüglich der betreffenden Behauptung unterscheidet. Der Simulator kann zwar den Verifier beliebig zurücksetzen und einige wenige Male mit leicht veränderten Sichten weiter laufen lassen. Die erwartete Zahl dieser Wiederholungen muss jedoch klein bleiben, so dass der Simulator effizient arbeitet.

Das Protokoll aus [BCY91] basiert wie anfangs erwähnt auf der Arbeit von [BCC88], dessen Protokoll hier in einer Kurzform vorgestellt werden soll, um die zugrundeliegende Technik und Idee besser verstehen zu können. Das Basis–Protokoll wird anschließend zu dem endgüligen Protokoll des perfekten Zero–Knowledge Argument in konstanter Rundenzahl erweitert.

6.6 Kurzbeschreibung des Basis–Protokolls und Definitionen

Mit der Bezeichnung $f : X \to Y$ wird nur eine Funktion bezeichnet, die sich effizient berechnen lässt. Darüber hinaus bezeichnet aus Vereinfachungsgründen f einen effizienten Algorithmus zur Berechnung dieser Funktion. Mit $f : X \xrightarrow{R} Y$ wird ein effizienter probabilistischer Algorithmus bezeichnet, der jedem $x \in X$ eine Wahrscheinlichkeitsverteilung $f(x)$ über Y zuordnet. Wenn keine Verwechslungsgefahr besteht, wird mit $f(x)$ das wahrscheinliche Ergebnis des verwendeten Algorithmus f bei der Eingabe von x bezeichnet.

Sei $\Sigma = \{0, 1\}$ und $\Gamma = \{0, 1, \square\}$. Bei einer gegebenen Menge X wird mit $X^\star$ die endliche Folge von Elementen aus X bezeichnet. (Das Symbol $*$ aus $\mathbb{Z}_p^*$ ist nicht zu verwechseln mit dem $\star$ von $X^\star$.) Für eine Zeichenkette $\vec{x} \in X^\star$ bezeichnet $x[i]$ das i-te Element. Für $\vec{\nu} \in \Sigma^\star$ und $\vec{\epsilon} \in \Gamma^\star$ sind $\vec{\nu}$ und $\vec{\epsilon}$ miteinander *verträglich*, bezeichnet durch $\vec{\nu} \asymp \vec{\epsilon}$,[22] wenn $|\vec{\nu}| = |\vec{\epsilon}|$ und für alle $i, 1 \leq i \leq |\vec{\nu}|$, $\vec{\nu}[i] = \vec{\epsilon}[i]$ gilt, falls $\vec{\epsilon}[i] \neq \square$. Dabei soll das Symbol $\square$ ein ungeöffnetes Bit–Commitment repräsentieren. Dazu wird eine Überführungsfunktion $\diamondsuit : \Gamma \to \Sigma$ mit $\diamondsuit(0) = 0$, $\diamondsuit(1) = 1$ und $\diamondsuit(\square) = 0$ definiert. Diese Funktion wird in entsprechender Weise auf $\diamondsuit : \Gamma^\star \to \Sigma^\star$ erweitert.

[21] Das heißt, dass der Verifier als Turing Maschine betrachtet wieder auf eine schon mal abgearbeitete Konfiguration gesetzt wird.

[22] Das Symbol $\asymp$ soll dabei an das $=$ Symbol erinnern, wenngleich auch zwei Elemente unterschiedlicher Ausgangsmengen „verglichen“ werden.

Sei $n \in \mathbb{Z}$, dann bezeichnet $\mathcal{B}_n$ die Menge aller Booleschen Ausdrücke mit n Variablen und $\mathcal{B} = \bigcup_{n\in\mathbb{Z}} \mathcal{B}_n$. Sei ferner $\Psi \in \mathcal{B}$ die Bezeichnung für einen Booleschen Ausdruck. Dann wird mit $\mathcal{W}_n = \{\vec{a} \mid \vec{a} \in \{\texttt{wahr, falsch}\}^n,\ \vec{a} \text{ erfüllt } \Psi\}$ sowie $\mathcal{W} = \bigcup_{n\in\mathbb{Z}} \mathcal{W}_n$ die Menge aller Belegungen bezeichnet, die die Booleschen Ausdrücke erfüllen.

Weiterhin ist jedem $\Psi \in \mathcal{B}$ eine endliche Menge $\mathcal{V}_\Psi \subseteq \Sigma^\star$ der „verschlüsselten Schaltkreise“ sowie $\mathcal{E}_\Psi \subseteq \Gamma^\star$ der teilweise „entschlüsselten Schaltkreise“ zugeordnet. Für jeden Ausdruck $\Psi \in \mathcal{B}$ und jede Zeichenkette $\vec{\nu} \in \Sigma^\star$ (rsp. $\vec{\epsilon} \in \Gamma^\star$) ist effizient entscheidbar, ob $\vec{\nu} \in \mathcal{V}_\Psi$ (rsp. $\vec{\epsilon} \in \mathcal{E}_\Psi$) gilt. Überdies gilt für jeden $\Psi \in \mathcal{B}$, dass alle Elemente aus $\mathcal{V}_\Psi$ und $\mathcal{E}_\Psi$ dieselbe Länge $\ell(\Psi)$ haben.

Die Schlüsseleigenschaft des Basis–Protokolls besagt nun folgendes: Sei $\Psi \in \mathcal{B}$ ein beliebiger Ausdruck. Dann gilt

$$\exists\vec{\nu} \in \mathcal{V}_\Psi, \exists\vec{\epsilon} \in \mathcal{E}_\Psi : \vec{\nu} \asymp \vec{\epsilon} \iff \Psi \text{ ist erfüllbar.}$$

Dazu existieren vier effiziente Algorithmen:

1. $\zeta : \mathcal{B} \xrightarrow{R} \mathcal{V}$
2. $\rho : \mathcal{B} \xrightarrow{R} \mathcal{E}$
3. $\gamma : \mathcal{B} \times \mathcal{V} \times \mathcal{W} \to \mathcal{E} \cup \{\perp\}$ und
4. $\delta : \mathcal{B} \times \mathcal{V} \times \mathcal{E} \to \mathcal{W} \cup \{\perp\}$

so dass folgendes gilt:

1. $\forall\Psi \in \mathcal{B} : \zeta(\Psi)$ ist gleichverteilt über $\mathcal{V}_\Psi$.
2. $\forall\Psi \in \mathcal{B} : \rho(\Psi)$ ist gleichverteilt über $\mathcal{E}_\Psi$.
3. Sei $\Psi \in \mathcal{B}_n$, $\vec{a} \in \mathcal{W}_n$ erfüllt Ψ und $\vec{\nu} \in \mathcal{V}_\Psi$. Dann ist $\gamma(\Psi, \vec{\nu}, \vec{a}) \in \mathcal{E}_\Psi$ und verträglich mit $\vec{\nu}$. (Andernfalls gilt $\gamma(\Psi, \vec{\nu}, \vec{a}) \in \perp$.) Außerdem gilt $\forall\Psi \in \mathcal{B}, \forall\vec{a} \in \mathcal{W}$ erfüllt Ψ : $\gamma(\Psi, \zeta(\Psi), \vec{a})$ ist gleichverteilt über $\mathcal{E}_\Psi$.
4. Seien $\Psi \in \mathcal{B}$, $\vec{\nu} \in \mathcal{V}_\Psi$ und $\vec{\epsilon} \in \mathcal{E}_\Psi$ mit $\vec{\nu} \asymp \vec{\epsilon}$. Dann ist $\delta(\Psi, \vec{\nu}, \vec{\epsilon})$ eine erfüllende Belegung zu Ψ. Andernfalls ist $\delta(\Psi, \vec{\nu}, \vec{\epsilon}) = \perp$.

Angenommen, der Prover möchte den Verifier davon überzeugen, dass er eine erfüllende Belegung für einige $\Psi \in \mathcal{B}$ kennt. Das Basis–Protokoll besteht aus k unabhängigen Runden, die nacheinandern ausgeführt werden. Dabei sieht eine Runde i wie folgt aus:

Protokoll 6.2 (Runde i des Basis–Protokolls)
Schritt P1: Der Prover berechnet $\vec{\nu}_i = \zeta(\Psi)$, verschlüsselt jedes Bit $\nu_i[j]$ in einem Commitment $x_i[j]$ und sendet es.

Schritt V2: Der Verifier sendet $y_i \in \{0, 1\}$ als seine Öffnungswahl der Commitments.

Schritt P3: War die Wahl eine „0“, öffnet der Prover die $x_i[j]$, sendet diese für jedes j und zeigt damit $\vec{\nu}_i$. Der Verifier prüft $\vec{\nu}_i \in \mathcal{V}_\Psi$.

War die Wahl eine „1“, nutzt der Prover sein geheimes Wissen über eine erfüllende Belegung $\vec{a}$ und berechnet $\vec{\epsilon}_i = \gamma(\Psi, \vec{\nu}_i, \vec{a})$. Er sendet lediglich die $x_i[j]$'s, die $\epsilon_i[j] \neq \square$ entsprechen. (Zu beachten ist, dass in diesem Fall $\epsilon_i[j] = \nu_i[j]$ gilt, da $\vec{\nu}_i \asymp \vec{\epsilon}_i$.) Der Verifier prüft $\vec{\epsilon}_i \in \mathcal{E}_\Psi$.

In beiden Fällen stoppt der Verifier und verwirft die Behauptung, wenn die Prüfung fehlerhaft war.

Lemma 6.1
Das Protokoll 6.2 ist berechenbar eindeutig, da der Prover nicht im Voraus weiß, welche Aufforderung in jeder Runde zu erwarten ist.

Beweis:
Seien $\vec{\nu}_i$ und $\vec{\epsilon}_i$ die Antworten auf die Öffnungswahlen in Runde i. Unter der Voraussetzung, dass der Prover die Commitments nicht zweideutig öffnen kann, müssen $\vec{\nu}_i$ und $\vec{\epsilon}_i$ verträglich sein. Falls $\vec{\nu}_i \in \mathcal{V}_\Psi$ und $\vec{\epsilon}_i \in \mathcal{E}_\Psi$ gilt, dann ist $\delta(\Psi, \vec{\nu}_i, \vec{\epsilon}_i)$ eine erfüllende Belegung zu Ψ. Infolgedessen ist Ψ erfüllbar und der Prover kennt eine effizient berechenbare, erfüllbare Belegung dafür. Andernfalls, wenn $\vec{\nu}_i \notin \mathcal{V}_\Psi$ und $\vec{\epsilon}_i \notin \mathcal{E}_\Psi$ gilt, beträgt die Wahrscheinlichkeit mindestens 50%, dass der Verifier den Prover beim Täuschen in dieser Runde erwischt. Die Wahrscheinlichkeit für unerkannte falsche Behauptung des Provers beträgt somit 2^{-k} für das vollständige Protokoll, immer vorausgesetzt, dass der Prover das Bit–Commitment–Scheme nicht überlisten kann. ■

Lemma 6.2
Das Protokoll 6.2 ist perfekt Zero–Knowledge, vorausgesetzt, dass ein 0–Commitment informationstheoretisch nicht von einem 1–Commitment zu unterscheiden ist.

Beweis:
Um eine Runde des Protokolls zu simulieren, wirft der Simulator eine Münze.

Ist der Kopf oben, generiert der Simulator $\vec{\nu} = \zeta(\Psi)$, verschlüsselt jedes Bit in $\vec{\nu}$ in einem Commitment und fragt den Verifier nach seiner Wahl. Beträgt diese „0“, öffnet der Simulator alle Commitments. Andernfalls missachtet der Simulator diese „unglückliche“ Wahl und startet einen neuen Versuch.

Ist nach dem Münzwurf die Zahl oben, erzeugt der Simulator $\vec{\epsilon} = \rho(\Psi)$, setzt $\vec{\nu} = \diamondsuit(\vec{\epsilon})$, verschlüsselt jedes Bit in $\vec{\nu}$ in Commitments und fragt den Verifier nach seiner Wahl. Beträgt diese „1“, öffnet der Simulator die Commitments, die $\epsilon[j] \neq \square$ entsprechen. Andernfalls missachtet der Simulator diese „unglückliche“ Wahl und startet einen neuen Versuch.

Da ein 0–Commitment und ein 1–Commitment ununterscheidbar sind, ist die Wahl des Verifiers unabhängig von dem Münzwurf des Simulators. Daher ist die erwartete Anzahl der Versuche für eine erfolgreiche Simulation einer Runde 2.

Somit ist die Simulation vollständig und alle k Runden können eine nach der anderen in ungefähr doppelt soviel Zeit simuliert werden. ■

6.7 Perfektes Zero–Knowledge Argument in konstanter Rundenzahl

Nun kann die formale Beschreibung des endgültigen Protokolls für das perfekte Zero–Knowledge Argument in konstanter Rundenzahl für das SAT-Problem erfolgen. Da das SAT–Problem $\mathcal{NP}$-vollständig ist, folgt daraus, dass dieses Protokoll für alle Behauptungen, die in $\mathcal{NP}$ liegen, geeignet ist.

Zusammenfassend werden hier nochmal alle Bezeichnungen aufgeführt, die in dem Protokoll verwendet werden.

- p, α und s entsprechen dem Protokoll 6.1, $\mathbb{Z}_p^*$ einer multiplikativen Gruppe und $\mathbb{Z}_{p-1}$ einer additiven Gruppe gemäß Seite 54.
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, \square\}$
- ν das zu verschlüsselnde Bit
- $z \in_R \mathbb{Z}_{p-1}$
- $x = \alpha^z s^\nu$
- $\beta : \Sigma \xrightarrow{R} \mathbb{Z}_p^* \times \mathbb{Z}_{p-1}$ wird definiert mit $\beta(\nu) = (x, z)$.

  Das Tupel (x, z) entspricht dem Commitment des Bits ν, wobei x das eigentliche Commitment und z das Zeugnis darstellt.

  Diese Notation wird in natürlicher Weise erweitert auf $\beta(\vec{\nu})$ für $\vec{\nu} \in \Sigma^\star$. Sie wird ebenfalls erweitert auf $\beta(\vec{\epsilon})$ für $\vec{\epsilon} \in \Gamma^\star$, wobei hier $\beta(\vec{\epsilon}) = \beta(\diamondsuit(\vec{\epsilon}))$ bedeutet. O.b.d.A. wird für diesen Fall willkürlich $\epsilon[i] = \square$ als $\epsilon[i] = 0$ angenommen.

- $\varphi : \mathbb{Z}_p^* \times \mathbb{Z}_{p-1} \times \Gamma \to \{\texttt{wahr, falsch}\}$ wird definiert mit $\varphi(x, z, \nu) = \texttt{wahr}$, wenn entweder $\nu = \square$ oder $x = \alpha^z s^\nu$ gilt, d.h. dann und nur dann, wenn entweder das Commitment x ungeöffnet bleibt oder z ein korrektes Zeugnis dafür ist, dass x ein ν–Commitment ist.

  Diese Notation wird ebenfalls entsprechend erweitert auf $\varphi : (\mathbb{Z}_p^*)^\ell \times (\mathbb{Z}_{p-1})^\ell \times \Gamma^\ell \to \{\texttt{wahr, falsch}\}$ für jede ganze Zahl ℓ. In diesem Fall gilt $\varphi(\vec{x}, \vec{z}, \vec{\nu}) = \texttt{wahr}$ dann und nur dann, wenn $x[i] = \alpha^{z[i]} s^{\nu[i]}$ für alle i mit $\nu[i] = \square, 1 \leq i \leq \ell$.

- Ψ ist ein erfüllbarer Boolescher Ausdruck. Es wird angenommen, dass der Prover eine erfüllende Belegung $\vec{a}$ für Ψ kennt.
- Alle weiteren Notationen folgen den Definitionen des Basis–Protokolls ab Seite 58.

Um das nachfolgende Protokoll nicht mit trivialen Details zu überfrachten, werden offensichtliche aber notwendige Tests nicht explizit aufgeführt. Zum Beispiel muss der Prover sicherstellen, dass im Schritt P1 $s \in \mathbb{Z}_p^*$ gilt, weil der Verifier sonst umgehend das Geheimnis erhält, wenn der Prover die Kommunikation bei einem $s = 0$ fortführt.

Protokoll 6.3 (Zero–Knowledge Argument in konstanter Rundenzahl)
Gemeinsame Eingabe (Initialisierung)

- p: Eine große Primzahl, von dessen Vorgänger $p-1$ die Faktorisierung bekannt ist.
- $\alpha \in \mathbb{Z}_p^*$: Ein erzeugendes Element.
- $k \in \mathbb{N}$: Sicherheitsparameter

Dieser Schritt zählt nicht bei der Rundenzahl, da diese Einigung schon im Vorwege erfolgen kann. Desweiteren können p und α öffentlich bekannt und frei verfügbar sein.

Schritt V1: (Der Verifier bereitet die Bit–Commitment–Scheme Paramter vor und verschlüsselt seine Öffnungsaufforderung.) Der Verifier wählt zufällige $s \in_R \mathbb{Z}_p^*$ und $\vec{y} \in_R \Sigma^k$. Er berechnet $(\vec{e}, \vec{r}) = \beta(\vec{y})$ und sendet s und $\vec{e}$ an den Prover.

Schritt P1: (Der Prover verschlüsselt mit einem verschlüsselten Schaltkreis und sendet die Kontroll Commitments zusammen mit jedem aktuellen Commitment.) Für jedes i mit $1 \leq i \leq k$ berechnet der Prover $\vec{\nu}_i = \sigma(\Psi)$ und $(\vec{x}_i, \vec{z}_i) = \beta(\vec{\nu}_i)$. Für jedes Commitment $x_i[j]$ wählt der Prover ein zufälliges $\vec{b}_{ij} \in_R \{0,1\}^k$ und berechnet $(\vec{u}_{ij}, \vec{v}_{ij}) = \beta(\vec{b}_{ij})$. Er sendet alle $\vec{x}_i$'s und $\vec{u}_{ij}$'s an den Verifier. (Das $\vec{x}_i[j]$ repräsentiert dabei das akuelle Commitment für das j-te Bit des i-ten verschlüsselten Schaltkreises und $\vec{u}_{ij}$ stellt eine Familie von Kontroll Commitments für $x_i[j]$ dar.)

Schritt V2: (Der Verifier wählt für das aktuelle Commitment eine Öffnungsaufforderung um sicherzustellen, dass es kein EPR–Commitment ist.) Für jedes i und j wählt der Verifier zufällige $\vec{h}_{ij} \in_R \{0,1\}^k$ und sendet diese $\vec{h}_{ij}$ an den Prover.

Schritt P2: (Der Prover zeigt, dass seine aktuellen Commitments nicht EPR sind.) Für jedes i, j und m berechnet der Prover folgendes:

Wenn $h_{ij}[m] = 0$	dann setzt er	$w_{ij}[m] = v_{ij}[m]$
	und	$t_{ij}[m] = b_{ij}[m]$
Wenn $h_{ij}[m] = 1$ und $\nu_i[j] = b_{ij}[m]$	dann setzt er	$w_{ij}[m] = v_{ij}[m] - z_i[j]$
	und	$t_{ij}[m] = 0$
Wenn $h_{ij}[m] = 1$ und $\nu_i[j] \neq b_{ij}[m]$	dann setzt er	$w_{ij}[m] = v_{ij}[m] + z_i[j]$
	und	$t_{ij}[m] = 1$

Der Prover sendet alle $\vec{w}_{ij}$'s und $\vec{t}_{ij}$'s an den Verifier.

Schritt V3: (Der Verifier prüft, dass die aktuellen Commitments nicht EPR sind. In diesem Fall öffnet er seine Öffnungsaufforderung.) Für jedes i, j und m prüft der Verifier:

Wenn $h_{ij}[m] = 0$	prüft er	$\varphi(u_{ij}[m], w_{ij}[m], t_{ij}[m])$
Wenn $h_{ij}[m] = 1$ und $t_{ij}[m] = 0$	prüft er	$u_{ij}[m] = \alpha^{w_{ij}[m]} x_i[j]$
Wenn $h_{ij}[m] = 1$ und $t_{ij}[m] = 1$	prüft er	$u_{ij}[m] x_i[j] = \alpha^{w_{ij}[m]}$

Wenn einer dieser Kontrollen fehl schlägt, dann stoppt der Verifier und verwirft die Behauptung. Andernfalls sendet der Verifier $\vec{r}$ und $\vec{y}$ an den Prover.

Schritt P3: (Der Prover kontrolliert, dass die Öffnungsaufforderung korrekt ist und folgt dieser.) Der Prover prüft $\varphi(\vec{e}, \vec{r}, \vec{y})$. Falls die Prüfung fehl schlägt, beendet der Prover die Kommunikation. Andernfalls sendet der Prover folgendes für alle i an den Verifier:

Wenn $y[i] = 0$	sendet er	$\vec{z}_i$ und $\vec{\nu}_i$.
Wenn $y[i] = 1$	berechnet er	$\vec{\epsilon}_i = \gamma(\Psi, \vec{\nu}_i, \vec{a})$,
	setzt	$z_i[j] = 0$ falls $\epsilon_i[j] = \square$
	und sendet	$\vec{z}_i$ und $\vec{\epsilon}_i$.

Schritt V4: (Der Verifier kontrolliert, ob alles in Ordnung ist.) Für alle i prüft der Verifier:

Wenn $y[i] = 0$ prüft er $\vec{\nu}_i \in \mathcal{V}_\Psi$ und $\varphi(\vec{x}_i, \vec{z}_i, \vec{\nu}_i)$.
Wenn $y[i] = 1$ prüft er $\vec{\epsilon}_i \in \mathcal{E}_\Psi$ und $\varphi(\vec{x}_i, \vec{z}_i, \vec{\epsilon}_i)$.

Wenn einer dieser Kontrollen fehl schlägt, dann stoppt der Verifier und verwirft die Behauptung. Andernfalls stoppt der Verifier und akzeptiert.

Es ist offensichtlich, dass das Protokoll 6.3 perfekt vollständig ist. Als nächstes wird gezeigt, dass das Protokoll berechenbar eindeutig ist.

Lemma 6.3 (Protokoll 6.3 ist berechenbar eindeutig)
Angenommen, der Prover versucht seinen Beweis auf eine Boolesche Formel Ψ zu begründen, die nicht erfüllbar ist. Weiterhin angenommen, dass der Prover nicht in der Lage ist, während der Laufzeit der Kommunikation den diskreten Logarithmus von dem s auszurechnen, welches er in Schritt V1 von dem Verifier erhalten hat. Dann beträgt die Wahrscheinlichkeit, dass der Täuschungsversuch unentdeckt bleibt, im besten Fall 2^{-k}, wenn der Prover gemäß Protokoll 6.3 mit einem ehrlichen Verifier interagiert.

Als Vorbemerkung zum Beweis sei auf folgendes hingewiesen. Das Verhalten des Provers kann nicht von dem Vektor $\vec{y}$, mit dem der Verifiers seine Öffnungswahl festlegt, abhängig sein, da der Vektor der Commitments $\vec{e}$ informationstheoretisch nicht mit der Öffnungswahl korreliert.

Beweis:
In Schritt P1 kann der Prover entweder mindestens ein EPR–Commitment $x_i[j]$ ausgeben oder alle der $\vec{x}_i$'s sind vollständig aus nicht–EPR–Commitments zusammengesetzt. Im ersten Fall beträgt die Wahrscheinlichkeit, dass ein solches Commitment vom Verifier in Schritt V3 nicht erkannt wird, entsprechend der *Commitment Zertifizierung* auf Seite 56 maximal 2^{-k}. Im zweiten Fall kann es höchstens einen Öffnungsaufforderungsvektor $\vec{y}$ geben, für den der Prover den Verifier in Schritt V4 zufrieden stellen kann. Dieses gilt unter folgender Voraussetzung. Es gibt ein i derart, dass der Prover in beiden Fällen, sowohl bei $y_i = 0$ (bei gegebenem $\vec{\nu}_i$) als auch bei $y_i = 1$ (bei gegebenem $\vec{\epsilon}_i$), die Aufforderung des Verifiers zum Öffnen des Commitments zufriedenstellen kann. In diesem Fall müssen $\vec{\nu}_i$ und $\vec{\epsilon}_i$ verträglich sein, da gemäß Voraussetzung der Prover die Commitments nicht ändern kann und diese nicht EPR sind. Aber dann muss $\delta(\Psi, \vec{\nu}, \vec{\epsilon})$ eine erfüllende Zuweisung für Ψ sein, was ein Widerspruch zur Voraussetzung ist, dass Ψ nicht erfüllbar ist. Daher ist die Wahrscheinlichkeit, dass der Verifier in Schritt V3 gerade den Öffnungsaufforderungsvektor an den Prover sendet, den dieser erwartet und mit dem er täuschen kann, höchstens 2^{-k}. ■

Anschließend wird der für das Protokoll 6.3 wichtigste Teil des Satzes 6.1 bewiesen, die perfekte Zero–Knowledge Eigenschaft. Zu diesem Zweck muss ein Simulator gezeigt werden, dessen Laufzeiteffizienz zu beweisen ist und dessen Ausgabe des Empfangsbandes dieselbe Verteilung aufweist, wie das aufgeführte Protokoll.

Um die Kommunikation des realen Provers mit jedem beliebigen Verifier perfekt und effizient simulieren zu können, müssen unterschiedliche Verhaltensmöglichkeiten der Verifier beachtet werden. Unter anderem muss der Simulator folgende Situationen berücksichtigen:

1. Der Verifier wählt in Schritt V1 ein s derart, dass dessen diskreter Logarithmus bekannt ist. Dies erlaubt dem Verifier, in Schritt V1 ein simuliertes Commitment zu erzeugen und die echte Wahl der Öffnungsaufforderung nach Schritt V3 zu verschieben.

2. Der Verifier wählt in Schritt V1 ein s, ohne dessen diskreten Logarithmus zu kennen.

3. Der Verifier wählt das s so, dass er dessen diskreten Logarithmus in Schritt V1 nicht kennt, diesen aber in einigen Fällen vor dem Schritt V3 ausrechnen kann, zum Beispiel durch Informationen, die vom Prover in den Schritten P1 und P2 veröffentlicht wurden. Natürlich helfen solche Informationen dem Verifier nur durch

reines Glück, da der Prover den diskreten Logarithmus von s nicht kennt. Nichtsdestotrotz ist derartiges Glück möglich und muss berücksichtigt werden.

4. Der Verifier weicht zu bestimmten Zeitpunkten von dem Protokoll ab, das heißt dass er die „vernünftige“ Zusammenarbeit verweigert und dadurch den Prover dazu bringt, die Interaktion vorzeitig zu beenden. Dass geschieht zum Beispiel in Schritt V1, wenn der Verifier ein $s = 0$ liefert, oder in Schritt V3, wenn er behauptet, der Prover habe in Schritt P2 bei den Kontroll Commitments getäuscht, ohne dass das tatsächlich der Fall ist, oder wenn er eine „2“ als eine der Öffungsaufforderungen y_i in V3 liefert.

Im ersten Fall könnte die Öffnungswahl des Verifiers von der Ausgabe des Provers in den Schritten P1 und P2 abhängig sein. Wenn dies eintritt, wird der Simulator den Verifier zurücksetzen und dadurch den diskreten Logarithmus von s erhalten. Wenn der Simulator diese Information erst einmal hat, kann er in Schritt P3 „mogeln“, da er damit den Schlüssel zum Invertieren des Bit–Commitment–Schemes hat.

Im zweiten Fall lässt der Simulator den Verifier einmal laufen, um die Öffnungswahl in Schritt V3 zu erhalten. Anschließend wird der Verifier auf eine vorherige Konfiguration zurückgesetzt, und der Schritt P1 wird erneut simuliert. Diesmal jedoch mit Commitments, die der Öffnungswahl des Verifiers entsprechen.

Die Fälle drei und vier sind schwieriger zu handhaben und werden bei der nachfolgenden formalen Beschreibung des Simulators abgehandelt.

Um den Simulator zu beschreiben, ist es einfacher sich vorzustellen, dass der Verifier eine deterministische Maschine ist, die Zugriff auf ein Zufallsband hat. Dieses Zufallsband unterliegt der vollständigen Kontrolle des Simulators. Anfangs ist das Band leer. Jedesmal, wenn der Verifier Zugriff auf ein neues Bit des Zufallsbandes hat, führt der Simulator einen Münzwurf aus und schreibt das Resultat für den Verifier auf das Zufallsband. Zu jeder Zeit kann der Simulator einen Speicherauszug vom Verifier anfertigen bzw. sich als Turing Maschine die Konfiguration merken und später den Verifier exakt an dieser Stelle weiterarbeiten lassen. Wenn der Verifier zu einer vorhergehenden Konfiguration zurückgesetzt wird, dann wird der Lesekopf des Zufallbandes ebenfalls auf die entsprechende Stelle zurückgesetzt, mit dem Unterschied, dass die bereits auf das Zufallsband geschriebenen Zufallsbits, die hinter dieser Stelle liegen, erhalten bleiben, so dass der Verifier nach dem Rücksetzen exakt dieselben Bits wieder verarbeitet, die er beim vorigen Durchlauf schon einmal gelesen hat. Der Vorteil dieses Ansatzes gegenüber dem einmaligen Erzeugen des Zufallbandes durch den Simulator für alle denkbaren Zugriffe des Verifiers liegt darin, dass damit Verifier simuliert werden können, die keine von vornherein bekannte Zeitschranke haben. Noch wichtiger ist jedoch, dass selbst dann, wenn nur Verifier mit einer *erwarteten* polynomiellen Laufzeit betrachtet werden, im Einzelfall signifikant mehr Zeit benötigt wird, wenngleich auch nur mit geringer Wahrscheinlichkeit.

Werden in dem nachfolgendem Protokoll Schritte V_i oder P_i erwähnt, beziehen sich diese auf die entsprechenden Schritte aus Protokoll 6.3. Dabei ist zu beachten, dass der Simulator bis einschließlich Schritt S6 exakt dem Protokoll 6.3 folgt (in der Rolle des Provers).

Protokoll 6.4 (Simulator des Protokolls 6.3)

Schritt S0: Der Simulator führt die Initialisierung der gemeinsamen Eingabe exakt so wie der Prover durch. Falls der Verifier von dem Protokoll 6.3 abweicht, gibt der Simulator den bisherigen Inhalt des Empfangs- und Zufallbandes des Verifiers aus und stoppt.

Schritt S1: Der Simulator wartet entsprechend Schritt V1 darauf, dass der Verifier die s und $\vec{e}$ ausgibt. Falls der Verifier von dem Protokoll 6.3 abweicht, gibt der Simulator den bisherigen Inhalt des Empfangs- und Zufallbandes des Verifiers aus und stoppt. Andernfalls fertigt der Simulator einen Speicherauszug vom Verifier an.

Schritt S2: Der Simulator verfährt entsprechend Schritt P1. Insbesondere sendet er die $\vec{x}_i$'s und $\vec{u}_{ij}$'s an den Verifier.

Schritt S3: Der Simulator wartet darauf, dass der Verifier wie in Schritt V2 die $\vec{h}_{ij}$'s liefert. Falls der Verifier von dem Protokoll 6.3 abweicht, gibt der Simulator den bisherigen Inhalt des Empfangs- und Zufallbandes des Verifiers aus und stoppt.

Schritt S4: Der Simulator verfährt entsprechend Schritt P2. Insbesondere sendet er die $\vec{w}_{ij}$'s und $\vec{t}_{ij}$'s an den Verifier.

Schritt S5: Der Simulator wartet darauf, dass der Verifier wie in Schritt V3 die $\vec{r}$'s und $\vec{y}$'s liefert. Falls der Verifier von dem Protokoll 6.3 abweicht, gibt der Simulator den bisherigen Inhalt des Empfangs- und Zufallbandes des Verifiers aus und stoppt.

Schritt S6: Der Simulator verifiziert $\varphi(\vec{e}, \vec{r}, \vec{y})$. Misslingt diese Prüfung, gibt der Simulator den bisherigen Inhalt des Empfangs- und Zufallbandes des Verifiers aus und stoppt.

Schritt S7: (Das ist der Wendepunkt in dieser Simulation.) Der Simulator setzt den Verifier auf diejenige Konfiguration zurück, die sich der Simulator am Ende von Schritt S1 gemerkt hat.

Schritt S8: (Der Simulator sendet solche Commitments, die er nur dann öffnen kann, wenn die Öffnungsaufforderungen $\hat{y}$ erneut gestellt werden.) Für jedes i mit $\hat{y}_i = 0$ berechnet der Prover $\vec{\nu}_i = \sigma(\Psi)$ und $(\vec{x}_i, \vec{z}_i) = \beta(\vec{\nu}_i)$. Für jedes i mit $\hat{y}_i = 1$ berechnet der Prover $\vec{\epsilon}_i = \rho(\Psi)$ und $(\vec{x}_i, \vec{z}_i) = \beta(\vec{\epsilon}_i)$. Danach verfährt der Simulator entsprechend Schritt P1. Für jedes Commitment $x_i[j]$ wählt der Prover ein zufälliges $\vec{b}_{ij} \in_R \{0,1\}^k$ und berechnet $(\vec{u}_{ij}, \vec{v}_{ij}) = \beta(\vec{b}_{ij})$. Er sendet alle $\vec{x}_i$'s und $\vec{u}_{ij}$'s an den Verifier.

Schritt S9: Der Simulator wartet darauf, dass der Verifier wie in Schritt V2 die $\vec{h}_{ij}$'s liefert. Falls der Verifier von dem Protokoll 6.3 abweicht, *geht der Simulator zurück zu Schritt S7.*

Schritt S10: Der Simulator verfährt entsprechend Schritt P2 mit der Ausnahme, dass dann, wenn $\hat{y}_i = 1$ ist, $\nu_i[j]$ durch $\diamondsuit(\epsilon_i[j])$ ersetzt wird. Insbesondere sendet er die $\vec{w}_{ij}$'s und $\vec{t}_{ij}$'s an den Verifier.

Schritt S11: Der Simulator wartet darauf, dass der Verifier wie in Schritt V3 die $\vec{r}$'s und $\vec{y}$'s liefert. Es ist möglich, dass die Antwort des Verifiers sich von derjenigen unterscheidet, die er vorher in Schritt S5 gegeben hat. Falls der Verifier von dem Protokoll 6.3 abweicht, *geht der Simulator zurück zu Schritt S7.*

Schritt S12: Der Simulator verifiziert $\varphi(\vec{e}, \vec{r}, \vec{y})$. Misslingt diese Prüfung, *geht der Simulator zurück zu Schritt S7.*

Schritt S13: An diesem Punkt gibt es zwei Möglichkeiten: Entweder es gilt $\vec{y} = \hat{y}$ oder $\vec{y} \neq \hat{y}$. Im ersten Fall hat der Simulator seine Commitments so vorbereitet, dass er der jeweiligen Öffnungsaufforderung nachkommen kann. Im zweiten Fall bedeutet dies, dass der Verifier seine Öffnungswahl geändert hat, was dem Simulator ermöglicht, den diskreten Logarithmus von s zu berechnen. Wenn $\vec{y} = \hat{y}$ gilt, fährt das Protokoll mit Schritt S15 fort, sonst mit Schritt S14.

Schritt S14: (Der Verifier hat seine Öffnungswahl zwischen Schritt S5 und S11 geändert.) Es werden alle i mit $1 \leq i \leq k$ betrachtet, für die $y_i \neq \hat{y}_i$ gilt. Der Simulator berechnet

$$a = \begin{cases} r_i - \hat{r}_i & \text{falls } y_i = 0 \\ \hat{r}_i - r_i & \text{sonst} \end{cases}$$

(Dabei ist zu bedenken, dass diese Subtraktion in $\mathbb{Z}_{p-1}$ und damit modulo $p-1$ erfolgt.) Da $\alpha^{r_i} s^{y_i} = e_i = \alpha^{\hat{r}_i} s^{\hat{y}_i}$ gilt, folgt damit $\alpha^a = s$. Somit hat der Simulator den diskreten Logarithmus von s.

Für jedes i mit $y_i = 0$ und $\hat{y}_i = 1$ berechnet der Simulator $\vec{\nu}_i = \sigma(\Psi)$ und für jedes i mit $y_i = 1$ und $\hat{y}_i = 0$ berechnet er $\vec{\epsilon}_i = \rho(\Psi)$. Für jedes i, für das $y_i \neq \hat{y}_i$ gilt, und für jedes j, für das $\nu_i[j] \neq \diamondsuit(\epsilon_i[j])$ gilt, ändert der Simulator die $z_i[j]$'s aus Schritt S8 wenn nötig wie folgt:

$$z_i[j] = \begin{cases} z_i[j] - a & \text{falls } \hat{y}_i = \nu_i[j] \\ z_i[j] + a & \text{sonst} \end{cases}$$

Dies erlaubt dem Simulator, die „Commitments“ aus den x_i's in einer Weise zu öffnen, die die Öffnungsaufforderungen y_i erfüllen, auch wenn $\vec{x}_i$ ursprünglich dafür vorbereitet war, die Öffnungswahl $\hat{y}_i$ zu erfüllen.

Schritt S15: (Der Simulator ist jetzt fertig, um die Sicht des Verifiers auszugeben.) Der Simulator gibt folgende Speicherinhalte aus:

- Das Zufallsband des Verifiers.
- Die $\vec{x}_i$'s und $\vec{u}_{ij}$'s aus Schritt S8.
- Die $\vec{w}_{ij}$'s und $\vec{t}_{ij}$'s aus Schritt S10.
- Für jedes i mit $y[i] = 0$ die $\vec{z}_i$ und $\vec{\nu}_i$.
- Für jedes i mit $y[i] = 1$ setzt der Simulator $z_i[j] = 0$, falls $\epsilon_i[j] = \square$ ist, und gibt die $\vec{z}_i$ und $\vec{\epsilon}_i$ aus.

Der Simulator hält.

Es muss nachgewiesen werden, dass die von dem Simulator im Protokoll 6.4 erzeugte Ausgabe mit derselben Wahrscheinlichkeitsverteilung erfolgt, wie die Sicht eines beliebigen Verifiers bei einer Kommunikation mit dem ehrlichen Prover im Protokoll 6.3 und dass der Simulator effizient ausgeführt werden kann.

Lemma 6.4
Protokoll 6.3 ist perfekt Zero–Knowledge.

Beweis:
Seien Ψ ein erfüllbarer Boolescher Ausdruck und $\vec{a}$ eine dem ehrlichen Prover bekannte erfüllbare Belegung für Ψ. Mit $\ell = \ell(\Psi)$ wird die Länge jedes Elements von $\mathcal{V}_\Psi$ und $\mathcal{E}_\Psi$ bezeichnet. R sei eine unendlich lange Reihe von zufälligen Bits. Nachfolgend werden $\Psi, \vec{a}, \ell, R$ und der Verifier als beliebig aber fest angenommen. Daher kann gesagt werden, dass „A eindeutig von B determiniert“ ist, was bedeutet „von B und diesen festen Parametern“.

Nachfolgend wird die Sicht eines Verifiers von Protokoll 6.3 mit dem Zufallsband R bei einer Kommunikation mit einem ehrlichen Prover verglichen mit derjenigen Sicht, die von dem Simulator erzeugt wird, wenn dieser den entsprechenden Teil desselben Zufallbands R bei der Initialisierung erhält. Der Einfachheit halber wird davon ausgegangen, dass der Verifier nicht schon vor dem Schritt P1 vom Protokoll abweicht. In diesem Fall ist es offensichtlich, dass der Simulator perfekt ist. Ebenso offensichtlich ist, dass Schritt 0 in Protokoll 6.3 von Schritt S0 in Protokoll 6.4 exakt simuliert wird.

Bedingt durch die festgelegten Parameter insbesondere der Zufallbits sind die s und $\vec{e}$ des Verifiers in Schritt V1 determiniert. Somit können diese ebenfalls als fest betrachtet werden. Weiterhin sei darauf hingewiesen, dass der *einzige* Schritt in Protokoll 6.3, in dem der Prover eine Zufallsauswahl durchführt, P1 ist. Daher ist alles in Protokoll 6.3 vollständig determiniert durch die vom Prover gewählten $\vec{\nu}_i$'s, $\vec{z}_i$'s, $\vec{b}_{ij}$'s und $\vec{v}_{ij}$'s. (Die $\vec{x}_i$'s und $\vec{u}_{ij}$'s sind determiniert von diesen zufälligen Auswahlen.)

Die Sicht des Verifiers in Protokoll 6.3 ist vor dem Schritt P3 vollständig von den $\vec{x}_i$'s, $\vec{u}_{ij}$'s und $\vec{t}_{ij}$'s determiniert, die von dem Prover in den Schritten P1 und P2 gesendet

wurden (außer wenn Schritt P2 nicht stattfand und nur die $\vec{x}_i$'s und $\vec{u}_{ij}$'s allein die Sicht des Verifiers deterministisch bestimmen). Das beruht darauf, dass die vom ehrlichen Prover in Schritt P2 gesendeten $\vec{w}_{ij}$'s (vorausgesetzt, dieser Schritt findet statt) eindeutig durch die $\vec{x}_i$'s, $\vec{u}_{ij}$'s, $\vec{t}_{ij}$'s und $\vec{h}_{ij}$'s determiniert werden, die vom Verifier in Schritt V2 gesendet werden, wobei die $\vec{h}_{ij}$'s wiederum eindeutig durch die $\vec{x}_i$'s und $\vec{u}_{ij}$'s determiniert werden. Um das zu verstehen, sei darauf hingewiesen, dass die Tests in Schritt V3 bezüglich der $\vec{w}_{ij}$'s, die den Verifier nicht zum Zurückweisen veranlassen, nicht der Beliebigkeit unterliegen, da der diskrete Logarithmus eindeutig definiert ist.

Eine entscheidende Beobachtung ist weiterhin, dass alle vom Prover stammenden $\vec{x}_i$'s, $\vec{u}_{ij}$'s und $\vec{t}_{ij}$'s zufällig und unabhängig gleichverteilt über $\mathbb{Z}_p^*$, $\mathbb{Z}_p^*$ und $\{0,1\}$ erzeugt werden (mit der Ausnahme, dass die $\vec{t}_{ij}$'s nicht erzeugt werden, wenn Schritt P2 nicht durchgeführt wird). Bezeichne Λ den Raum der möglichen Werte für die Familien der $\vec{x}_i$'s, $\vec{u}_{ij}$'s und $\vec{t}_{ij}$'s mit $\Lambda = (\mathbb{Z}_p^* \times (\mathbb{Z}_p^* \times \{0,1\})^k)^{\ell k}$. Dann wird jedes $\lambda \in \Lambda$ als *gut* bezeichnet, wenn das Protokoll 6.3 Schritt V4 erreicht und die dem Verifier gelieferte Sicht auf λ im Einklang mit den Schritten P1 und P2 steht. Andernfalls ist λ *schlecht*. („Im Einklang stehen" bedeutet dabei, dass die $\vec{w}_{ij}$'s diejenigen eindeutigen Werte annehmen, die in Schritt V3 den Verifier nicht zum Zurückweisen veranlassen.) Schließlich bezeichne $\wp$ die Anzahl der guten Elemente von Λ im Verhältnis zu der Gesamtzahl der Elemente von Λ. Einfach ausgedrückt bezeichnet $\wp$ die Wahrscheinlichkeit, dass der Verifier sich im Protokoll 6.3 an den Protokollalgorithmus hält. Die Wahrscheinlichkeitsverteilung erfolgt dabei über die Zufallsauswahl des Provers (ob der Verifier in Schritt V4 die Behauptung verwirft, obwohl er sie gemäß Protokoll akzeptieren müsste, wird hierbei nicht betrachtet, da diese Entscheidung nicht Gegenstand dieser Betrachtung ist). Natürlich kann $\wp$ eine Funktion mit Bezug auf das Zufallsband R sein. Das betrifft diese Betrachtung jedoch nicht, da R hier fest ist.

Die Interaktion des Verifiers mit dem ehrlichen Prover in Protokoll 6.3 ist wie folgt charakterisiert. Der Prover wählt die $\vec{\nu}_i$'s, $\vec{z}_i$'s, $\vec{b}_{ij}$'s und $\vec{v}_{ij}$'s in Schritt P1 zufällig. Damit sind die $\vec{x}_i$'s und $\vec{u}_{ij}$'s determiniert. Falls der Verifier in Schritt V2 von dem Algorithmus abweicht, müssen zufällige Binärwerte für die $\vec{t}_{ij}$'s in Betracht gezogen werden, andernfalls legen die $\vec{x}_i$'s und $\vec{u}_{ij}$'s die $\vec{h}_{ij}$'s des Verifiers fest, welche wiederum die $\vec{t}_{ij}$'s des Provers in Schritt P2 deterministisch bestimmen. Stehe λ für die Familie der so erzeugten $\vec{x}_i$'s, $\vec{w}_{ij}$'s und $\vec{t}_{ij}$'s.

Wenn λ gut ist, was mit einer Wahrscheinlichkeit von $\wp$ eintritt, bestimmt es die Wahl $\vec{y}$ des Verifiers in Schritt V3. In Schritt P3 zeigt der Prover dem Verifier $\vec{\nu}_i$'s aus $\mathcal{V}_\Psi$, wenn $y_i = 0$ und $\vec{\epsilon}_i$'s aus $\mathcal{E}_\Psi$, wenn $y_i = 1$ ist, jeweils mit den dazugehörigen $\vec{z}_i$'s. Obwohl die $\vec{\nu}_i$'s und $\vec{\epsilon}_i$'s schon in Schritt P1 deterministisch festgelegt werden, sind sie vollständig unabhängig von λ, da Commitments nicht mit den Bits korrelieren, die sie verbergen. Außerdem sind die $\vec{\nu}_i$'s gleichverteilt über $\mathcal{V}_\Psi$ (weil sie durch einen Aufruf von $\sigma(\Psi)$ in Schritt P1 erzeugt werden), und die $\vec{\epsilon}_i$'s sind gleichverteilt über $\mathcal{E}_\Psi$ (weil sie durch $\gamma(\Psi, \sigma(\Psi), \vec{a})$ in den Schritten P1 und P3 erzeugt werden). Ganz im Gegenteil

dazu werden die $\vec{z}_i$'s vollständig von λ, den $\vec{\nu}_i$'s (bei $h_i = 0$) und den $\vec{\epsilon}_i$'s (bei $h_i = 1$) determiniert.

Als Schluss folgt daraus, dass die *Sicht des Verifiers* in Protokoll 6.3 eindeutig deterministisch ist und bestimmt wird von einem über Λ gleichverteilten, zufälligen λ und, falls λ gut ist, von einer Familie von zufälligen $\vec{\nu}_i$'s und $\vec{\epsilon}_i$'s, die über $\mathcal{V}_\Psi$ respektive $\mathcal{E}_\Psi$ gleichverteilt sind.

Nun wird die vom Simulator im Protokoll 6.4 erzeugte Sicht betrachtet, wenn dieser den Verifier mit dem entsprechenden Anfangssegment desselben Zufallbands R aufruft. Bekannt ist, dass das Protokoll 6.4 das Protokoll 6.3 exakt simuliert, bis (und falls) es Schritt S7 erreicht. Somit wird der Simulator Schritt S7 mit einer Wahrscheinlichkeit von $1-\wp$ niemals erreichen und erzeugt die Ausgabe eines Verifiers, der von dem Algorithmus abweicht. Diese Sicht wird mit der gleichen Wahrscheinlichkeitsverteilung erzeugt, als wenn der Verifier mit dem Prover interagiert hätte. Darüber hinaus steht diese Sicht im Einklang mit einem $\lambda \in \Lambda$, welches nicht gut ist, und dieses λ wird entsprechend der Gleichverteilung der schlechten Elemente von Λ erzielt (vorausgesetzt, dass $\wp \neq 1$).

Andererseits erreicht der Simulator Schritt S7 mit einer Wahrscheinlichkeit von $\wp$. Dieses erfolgt dann und nur dann, wenn die Sicht des Verifiers bis hierher im Einklang mit einem guten $\lambda \in \Lambda$ steht, wobei dieses λ gleichverteilt ist über die guten Elemente in Λ. Jeder Versuch weiter von Schritt S7 nach Schritt S13 zu gehen, kann als eine Stichprobe innerhalb von Λ angesehen werden, mit der Hoffnung, erneut ein gutes Element zu treffen. Es ist einleuchtend, dass ein solches gutes Element entsprechend der Gleichverteilung über den guten Elementen in Λ erlangt wird.

Von hier an erzeugt Schritt S15 $\vec{\nu}_i$'s aus $\mathcal{V}_\Psi$, wenn die Wahl des Verifiers $y_i = 0$ ist, und $\vec{\epsilon}_i$'s aus $\mathcal{E}_\Psi$, wenn $y_i = 1$ gilt. Zusammen mit den eindeutigen $\vec{z}_i$'s stehen diese im Einklang mit den $\vec{x}_i$'s aus λ. Unabhängig davon, ob diese $\vec{\nu}_i$'s und $\vec{\epsilon}_i$'s in Schritt S8 oder S14 gewählt wurden, sind sie gleichverteilt über $\mathcal{V}_\Psi$(mit einem Aufruf von $\sigma(\Psi)$) und $\mathcal{E}_\Psi$(mit einem Aufruf von $\rho(\Psi)$) und korrelieren nicht mit λ.

Zusammenfassend ist festzustellen, dass die Sicht, die von dem Simulator erzeugt wird, im Einklang mit einem zufälligen λ steht, welches mit einer Wahrscheinlichkeit von $1 - \wp$ gleichverteilt über die schlechten Elemente von Λ oder mit einer Wahrscheinlichkeit von $\wp$ über die guten Elemente von Λ und somit über alle Elemente von Λ ist. Ist darüber hinaus λ gut, dann enthält diese Sicht zusammen mit den dazugehörigen $\vec{z}_i$'s die $\vec{\nu}_i$'s und $\vec{\epsilon}_i$'s (wie sie von den mit λ im Einklang stehenden $\vec{h}_i$'s benötigt werden), die gleichverteilt über $\mathcal{V}_\Psi$ und $\mathcal{E}_\Psi$ sind. Mit anderen Worten, die von dem Simulator produzierte Sicht ist exakt genauso verteilt, wie die Sicht eines Verifiers bei einer Kommunikation mit einem ehrlichen Prover. Die beiden entscheidenden Eigenschaften des Protokolls 6.3 sind dabei zum einen die Funktionen $\rho(\Psi)$ und $\gamma(\Psi, \sigma(\Psi), \vec{a})$, die Elemente von $\mathcal{E}_\Psi$ mit derselben Wahrscheinlichkeitsverteilung erzeugen, und zum zweiten, dass die Commitments informatiomstheoretisch nicht mit den darin verborgenen Bits korrelieren. Daraus folgt der Nachweis, dass der Simulator perfekt ist. □

Lemma 6.5
Protokoll 6.3 ist effizient auszuführen.

Beweis:
Seien $\wp$, Λ sowie die festen Parameter wie im Beweis zum Lemma 6.4. Offensichtlich ist $\wp$ exakt die Wahrscheinlichkeit, mit der der Simulator in Protokoll 6.4 den Schritt S7 erreicht, da bis dahin der Simulator dem Protokoll 6.3 ohne Abweichung entspricht. Interessanter ist, dass $\wp$ ebenfalls exakt der Wahrscheinlichkeit entspricht, dass der Simulator, wenn er erst einmal Schritt S7 erreicht hat, ebenfalls Schritt 13 erreichen kann (und damit erfolgreich mit Schritt 15 die Simulation beenden kann), ohne dabei wenigstens einmal zurück zu Schritt S7 gegangen zu sein. Um das zu sehen, ist zu beachten, dass die Familien der $\vec{x}_i$'s, $\vec{u}_{ij}$'s und $\vec{t}_{ij}$'s, die in den Schritten S8 und S10 zufällig erzeugt wurden, über Λ gleichverteilt sind.

Infolgedessen ist die erwartete Anzahl, wie oft Schritt S7 erreicht wird, exakt 1 (solange nicht $\wp = 0$ ist, weil dann die Anzahl ebenfalls 0 ist). Es ist aber auch klar, dass die Schritte S0 bis S6 nicht mehr als einmal erreicht werden und dass kein Schritt nach S7 häufiger erreicht wird, als Schritt S7 selber. Daher ist die erwartete Zeit, die das Protokoll des Simulators benötigt, nicht mehr als die einmalige Ausführung jeden Schrittes dieses Protokolls, was unter der Berücksichtigung der Effizient des Verifiers insgesamt gesehen effizient ist. □

Satz 6.2
Das Protokoll 6.3 ist effizient und für einen ehrlichen Prover bei einer Kommunikation mit jedem beliebigen Verifier perfekt Zero–Knowledge.

Beweis:
Der Beweis ergibt sich aus dem Lemma 6.4 und dem Lemma 6.5. ■

7 Berechenbare Zero–Knowledge Arguments mit konstanter Rundenzahl

Wie in Kapitel 3.2 beschrieben, bedeutet eine *berechenbare* Geheimhaltung, dass ein unbeschränkter Verifier V^* weiterführende Informationen aus dem Kommunikationsprotokoll mit dem Prover berechnen könnte. Es reicht somit ein einfacherer Algorithmus seitens des Provers im Gegensatz zu perfekten Zero–Knowledge Verfahren. Es existieren in der Praxis zwar keine unbeschränkten Maschinen. Ein Verifier V^* kann jedoch nach der Kommunikation über einen beliebigen Zeitraum hinweg mit einer unbekannt starken Rechenkraft versuchen, weitere Informationen aus dem Kommunikationsprotokoll zu berechnen.

Um das zu verhindern, muss der Sicherheitsgrenzwert n_k der μ-Funktion (Definition 2.13 auf Seite 13) hoch genug gewählt werden. Bei Kommunikationen, die jedoch über Jahre hinweg einem Angriff durch einen Verifier V^* standhalten sollen, ist die Entwicklung einerseits immer stärkerer Maschinen und andererseits neuer informationstheoretischer Grundlagen zur Berechenbarkeit schwieriger Verfahren bzw. der dazu erforderlichen Komplexität nicht seriös kalkulierbar. Der Einsatz der berechenbaren Zero–Knowledge Arguments ist daher in der Praxis auf entsprechende Anwendungsbereiche beschränkt. Zwar kann der Wert n_k beliebig erhöht werden. Der damit verbundene Rechenaufwand nähert sich dann jedoch dem eines perfekten Zero–Knowledge Arguments.

Die Hauptarbeiten der berechenbaren Zero–Knowledge Arguments mit konstanter Rundenzahl stammen 1990 von Feige und Shamir [FS90a, FS90b], 1997 von Bellare, Jakobsson und Yung [BJY97] sowie 2001 (überarbeitet 2003) von Barak [Bar03]. Feige und Shamir entwickelten unter der Voraussetzung der Existenz von Einweg-Funktionen einen $2\frac{1}{2}$ Runden und unter der Voraussetzung der Existenz von Einweg-Permutationen einen 2 Runden Algorithmus. Diese Algorithmen bauen auf dem *witness hiding* Verfahren aus Abschnitt 3.5, Seite 33, auf. Der Algorithmus von Bellare, Jakobsson und Yung ist ein 2-Runden Verfahren, welches als kryptographische Voraussetzung nur noch auf der Existenz von Einweg-Funktionen beruht. Im Gegensatz dazu benutzt Barak als kryptographische Voraussetzung die Existenz kollisionsresistenter Hashfunktionen und nutzt die Entwicklung von Feige, Lapidot und Shamir [FLS00], das Problem der Zero–Knowledge Argument Konstruktion in die Konstruktion zweier einfacher Algorithmen zu unterteilen, die beide in konstanter Rundenzahl ablaufen können.

Das Neue an der Konstruktion von Barak ist, dass bei der Entwicklung des für den Nachweis der Zero–Knowledge Eigenschaft notwendigen Simulators auf den Programmcode des Verifiers zurückgegriffen wird, so dass eine Simulation in streng polynomieller Zeit bewiesen und damit sichergestellt wird. Von Barak und Lindell wird in [BL02] darüber hinaus nachgewiesen, dass ein Zero–Knowledge Argument, welches die *Black-Box* Technik[23] für den Simulator verwendet, indem der zu prüfende Verifier ohne Kenntnis sei-

[23] Siehe Abschnitt 7.2.1, Seite 74.

nes Programmcodes einfach zu einer vorhergehenden Konfiguration zurückgesetzt wird, die Simulation ausschließlich in erwarteter statt in streng polynomieller Zeit durchführen kann.

Satz 7.1 (Hauptergebnis)
Unter der Voraussetzung der Existenz kollisionsresistenter Hashfunktionen kann ein Simulator konstruiert werden, der nicht *als Black-Box funktioniert, so dass ein Zero–Knowledge Argument für jede Sprache in $\mathcal{NP}$ existiert, das folgende Eigenschaften erfüllt:*

1. *Es ist Zero–Knowledge hinsichtlich nicht-uniformer*[24] *Widersacher mit zusätzlichen Informationen.*
2. *Es hat eine konstante Rundenzahl und eine vernachlässigbare Fehlerwahrscheinlichkeit bezüglich der Korrektheit.*
3. *Es bleibt Zero–Knowledge, auch wenn es n-fach simultan bzw. parallel ausgeführt wird, wobei n der Sicherheitsparameter ist.*[25]
4. *Es ist ein Arthur-Merlin Protokoll mit gemeinsamen Münzwürfen für den Prover und den Verifier.*
5. *Der Simulator hat eine* streng *polynomielle statt nur eine* erwartete *polynomielle Laufzeit.*

7.1 Übersicht über das Grobkonzept

Das hier vorgestellte Zero–Knowledge Argument basiert auf der Technik von [FLS00]. Darin wird ein interaktiver Beweis für eine Sprache L so modifiziert, dass der Prover die Möglichkeit zum Täuschen hat, wenn er eine Trapdoor Information[26] σ besitzt, so dass der Verifier immer akzeptiert, auch wenn $x \notin L$ gilt oder der Prover kein Zeugnis für $x \in L$ kennt. Natürlich muss sichergestellt sein, dass es unmöglich ist, während der Interaktion mit dem Verifier diese Trapdoor Information σ zu bekommen, um die Korrektheit sicherzustellen. Auch wenn diese Modifikation auf den ersten Blick sinnlos erscheinen mag, ist sie entscheidend, die Zero–Knowledge Eigenschaft zu erhalten.

Der Grund liegt darin, dass es zwar für den Prover während der Kommunikation mit dem Verifier unmöglich ist, diese Trapdoor Information σ zu bekommen. Für den

[24] siehe dazu Abschnitt 7.2.4

[25] Ein Protokoll, welches diese Eigenschaft aufweist, wird als beschränkt simultan Zero–Knowledge (engl. *boundet concurrent zero-knowledge*) Protokoll bezeichnet. Das steht im Gegensatz zu anderen simultanen Zero–Knowledge Protokollen wie z.B. [DNS99], die Zero–Knowledge bleiben, wenn sie simultan in einer polynomiellen Anzahl ausgeführt werden.

[26] Siehe Definition 2.23, Seite 21.

Simulator ist diese Trapdoor Information dagegen leicht zu erzeugen. Vor der Arbeit von Barak wurde das in der Regel dadurch erreicht, dass der Simulator in einem Black-Box Zugriff den Verifier zurückgesetzt hat, um mit den bislang erworbenen Informationen über das Verhalten des Verifiers das erforderliche σ zu bekommen.

Die vorliegende Konstruktion nutzt einen ähnlichen Grundgedanken, ändert aber den Zugriff auf den Verifier. Das Protokoll wird so konstruiert, dass die Trapdoor Information σ einfach die *Beschreibung* der *Funktion der nächsten Nachricht* des Verifiers ist (d.h. der Code des Verifiers). Dazu ist anzumerken, dass die nächste–Nachricht Funktion des Verifiers schwer zu lernen sein kann (z.B. eine fest verdrahtete Zufallsfunktion). Für eine *Black-Box* Simulation ist es dann aber sehr schwer bzw. unmöglich, die korrekte Trapdoor Information zu erhalten.

Die Technik aus [FLS00] erlaubt es, ein Zeugnis als Trapdoor Information für $\mathcal{NP}$ Sprachen zu benutzen. Das reicht für den vorliegenden Fall nicht aus. Das Problem dabei ist, dass die Laufzeit des Verifiers durch kein festes Polynom beschränkt ist. Um das zu lösen, greift Barak auf die Methode des *Universellen Arguments* zurück. Einfach ausgedrückt, kann mit einem Universellen Argument die Technik aus [FLS00] durch den Einsatz eines Zeugnisses als Trapdoor Funktion für jede Sprache aus $\mathcal{N}time(T(n))$ genutzt werden, auch wenn $T(\cdot)$ super-polynomiell ist (z.B. $T(n) = n^{\log\log n}$).

7.2 Definitionen

Um ein Zero–Knowledge Argument zu konstruieren, der die aufgeführten Eigenschaften aufweist, werden nachfolgend die Definitionen aufgeführt, die entweder neu sind oder von den in Abschnitt 2 beschriebenen abweichen.

7.2.1 Black-Box Simulation

Mit Black-Box Simulation wird ein Protokoll bezeichnet, welches die Kommunikation zwischen einem Prover und einem Verifier simulieren soll, ohne die Strategie bzw. den Algorithmus des Verifiers zu kennen. Um den Verifier zum Akzeptieren zu veranlassen, benötigt der Simulator Kenntnis davon, unter welchen Voraussetzungen der Verifier akzeptierend fortfährt. Aus diesem Grund werden die vom Verifier genutzten Münzwürfe vom Simulator selber erzeugt und dem Simulator über das Zufallsband eingespielt. Damit erscheint für eine feste Sequenz von Münzwürfen der Verifier als erwartete deterministische Turing Maschine, die von dem Simulator wie folgt verwendet wird.

In der Simulation ruft der Simulator den Verifier gemäß Protokoll als jetzt erwartete deterministische Funktion auf und übergibt je nach Bedarf die erforderlichen Münzwürfe. Ist innerhalb der Simulation ein Punkt erreicht, an dem der Prover den Beweis für sein Geheimniss erbringen soll, übersendet der Verifier in der Regel seine Öffnungsaufforderung für vorab vereinbarte Beweise, die der Prover anschließend öffnet. Mit diesem

Wissen über das Verhalten des Verifiers spult der Simulator den Verifier wieder zu einem vorherigen Stand zurück und spielt einen Teil der Interaktion erneut durch, nunmehr mit dem Wissen, was er später dem Verifier beweisen soll. Dadurch kann der Simulator im zweiten Durchlauf entsprechende Berechnungen anstellen, die ihm mit diesem Zusatzwissen ermöglichen, den Öffnungsaufforderungen des Verifiers nachzukommen.

Dabei entstehen zwei Problemfelder, die es dem Simulator im Einzelfall sehr schwer oder sogar unmöglich machen, den Öffnungsaufforderungen des Verifiers trotz des Rücksetzens nachzukommen. Zum Einen kann der Verifier über einen hardcodierten Pseudozufallsgenerator verfügen, der nicht über das vom Simulator gesteuerte Zufallsband determiniert wird, so dass ein Rücksetzen mit einer nicht nur vernachlässigbaren Wahrscheinlichkeit nicht zum Erfolg führt. Zum Zweiten kann die Laufzeit des Verifiers zumindest unter bestimmten Voraussetzungen super-polynomiell sein, so dass dem polynomiellen Simulator keine Möglichkeit gegeben wird, alle Abfragen durchzuführen. Dazu wurde in [BL02] nachgewiesen, dass jeder Simulator, der den Programmcode des Verifieres nicht kennt und somit die Black-Box Technik verwendet, die Simulation nur in *erwarteter* Polynomzeit statt in *streng* polynomieller Zeit ausführen kann.

7.2.2 Statistisch eindeutiges Bit–Commitment–Scheme

Das in diesem Abschnitt verwendete Bit–Commitment–Scheme entspricht der Definition 2.16. Es ist statistisch eindeutig und nur berechenbar geheim.

Definition 7.1 (Statistisch eindeutiges Bit–Commitment–Scheme)
Seien $\mathrm{Com} : \{0,1\} \times \{0,1\}^n \to \{0,1\}^{p(n)}$ *eine polynomiell berechenbare Funktion ohne weitere Hilfseingabe,* $p(\cdot)$ *ein beliebiges Polynom und* n *ein Sicherheitsparameter.* Com *ist ein* ***statistisch eindeutiges Bit–Commitment–Scheme****, wenn gilt:*

- *Berechenbar geheim: Seien* U_n *und* U'_n *gleichverteilte Wahrscheinlichkeitsräume über der Menge* $\{0,1\}^n$*. Dann sind die Zufallsvariablen* $\mathrm{Com}(0, U_n)$ *und* $\mathrm{Com}(1, U'_n)$ *berechenbar ununterscheidbar.*
- *Statistisch eindeutig: Die Mengen* $\mathrm{Com}(0, \{0,1\}^n)$ *und* $\mathrm{Com}(1, \{0,1\}^n)$ *sind disjunkt.*

Notation: Sei $\mathrm{Com}(\sigma) = \mathrm{Com}(\sigma, U_n)$. Es wird $\mathrm{Com}^{-1}(y) = \sigma$ definiert, wobei das eindeutige Bit $\sigma \in \{0,1\}$ ist, für das $y = \mathrm{Com}(\sigma, w)$ gilt (für ein w, falls σ existiert). Andernfalls wird $\mathrm{Com}^{-1}(y) = \perp$ definiert. Wie in Definition 2.16 erfolgt auch hier eine Zeichenkettenverarbeitung durch Konkatenation der unabhängigen Commitments.

7.2.3 Zero–Knowledge Argument of Knowledge

Neben den bisher definierten Zero–Knowledge Beweissystemen, die ausschließlich beweisen, dass ein Wort in einer Sprache vorhanden ist, existieren noch interaktive Systeme, die beweisen, dass „ jemand“ ein Geheimnis bzw. ein Wort aus einer Sprache kennt (proof of knowledge). Diese Systeme werden als *Zero–Knowledge Proof of Knowledge* bzw. *Zero–Knowledge Argument of Knowledge* bezeichnet. Da hier keine Proof of Knowledge benötigt werden, beschränkt sich dieser Abschnitt auf die Einführung von Argument of Knowledge.

In einem Zero–Knowledge Argument of Knowledge möchte der Prover den Verifier davon überzeugen, dass er dazu ein Zeugnis *kennt*, welches beweist, dass $x \in L$ gilt. Dabei ist zu beachten, dass sich der Beweis über die Kenntnis von x von dem Beweis der bloßen Existenz von x unterscheidet. So erfordert z.B. die Kenntnis der Primfaktoren einer Zahl n einen anderen Beweis als der Beweis, dass n zusammengesetzt ist. Aus diesem Grund kann jedes Zero–Knowledge Argument in ein Zero–Knowledge Argument of Knowledge überführt werden. Die Umkehrung gilt im Allgemeinen jedoch nicht.

Die Grundidee zu Zero–Knowledge Proof of Knowledge stammt aus [GMR85] und wurde von Feige, Fiat und Shamir in [FFS87] und in [TW87] formal definiert. Von Bellare und Goldreich wurden in [BG92] einige Lücken dieser Definitionen geschlossen.

Zur Definition des Zero–Knowledge Argument of Knowledge wird der Begriff der *Orakel–Maschine* benötigt.

Definition 7.2 (Orakel–Maschine)
Sei E eine probabilistische Turing Maschine mit einem zusätzlichen Orakel–Band *und den beiden Zuständen* Anrufung des Orakels *(engl. oracle invocation) und* Orakel–Auskunft *(engl. oracle appeared). Die Berechnung von E mit der Eingabe x erfolgt dabei mittels Zugriff auf eine interaktive Funktion $A : \{0,1\}^* \to \{0,1\}^*$, die als* Orakel *bezeichnet wird. Dann heißt E eine* ***Orakel–Maschine****. $E^{A_x}(x)$ ist eine Zufallsvariable, die die Ausgabe von E mit dem Orakel A_x und der Einabe x beschreibt, wobei die Wahrscheinlichkeit von den Münzwürfen von E und A abhängt.*

Die Laufzeit der Orakel–Maschine ist wie bei Turing Maschinen die Anzahl der Schritte während der Berechnung. Die Anrufung des Orakels und dessen Antwort für jede Anfrage wird dabei als ein Schritt definiert. Jede Auskunft des Orakels an die Orakel–Maschine ist dabei unabhängig von den vorhergehenden Anrufungen.

Sei $p(x)$ die Wahrscheinlichkeit, dass der Prover P den Verifier V davon überzeugen kann, bei Eingabe von x zu akzeptieren. Dann ist zwinged erforderlich, dass es eine Maschine E gibt, genannt den *Extraktor* (engl. knowledge extractor), der in einer (erwarteten) Zeit proportional zu $\frac{1}{p(x)}$ ein Zeugnis für die Behauptung von P ausgibt.

Die Standardformulierung der Definition von [BG92] wird hier durch eine Definition

ersetzt, die von [Bar03] stammt und für Argumentsysteme anstatt für Beweissysteme formuliert wurde.

Definition 7.3 (Argument of Knowledge Systeme)
Seien $L = L(R)$ und (P, V) ein Argumentsystem für L. Sei $T : \mathbb{N} \to \mathbb{N}$ eine Funktion. Dann heißt (P, V) ein $T(n)$-korrektes Argument of Knowledge *für L, wenn eine probabilistische polynomiell–beschränkte Maschine E existiert (der* Extraktor*), so dass für jeden $T(n)$-großen Prover P^* und für jedes $x \in \{0,1\}^n$ gilt:*

$$Pr[E(P^*, x) \in R(x)] \geq Pr[out_V(P^*(x), V(x)) = 1] + \mu(n),$$

wobei μ eine vernachlässigbare Funktion ist.

(P, V) ist ein Argument of Knowledge, wenn es ein $T(n)$-korrektes Argument of Knowledge und $\frac{1}{T(n)}$ vernachlässigbar klein ist.

7.2.4 Uniforme und Nicht-uniforme Widersacher

Das Standardmodell effizienter Widersacher, also von Maschinen, die zu täuschen versuchen, um weitere Informationen zu erlangen, sind Familien von polynomiell-großen Schaltkreisen. In diesem Fall werden als Modell jedoch Schaltkreise der Größe $T(n)$ für eine *super-polynomielle* Funktion $T : \mathbb{N} \to \mathbb{N}$ (z.B. $T(n) = n^{\log n}$) betrachtet.

Daneben werden teilweise *uniforme* und *beschränkte nicht-uniforme* Widersacher betrachtet. Bei beiden handelt es sich um solche, die mittels probabilistischer polynomiell–beschränkter Turing Maschinen beschrieben werden. Nicht-uniforme haben darüber hinaus eine zusätzliche Hilfseingabe,[27] die als *Ratschlag* (engl. advice) bezeichnet wird und bei Eingaben der Größe n eine Länge von $l(n)$ aufweisen, wobei l eine beliebige feste Funktion $l : \mathbb{N} \to \mathbb{N}$ darstellt, die polynomiell in Bezug auf n ist. Es sei betont, dass die Laufzeit eines solchen Widersachers durch ein beliebiges Polynom beschrieben werden und damit insbesondere länger als die von $l(n)$ sein kann.

Anmerkungen:

- Es ist bekannt, dass in allen kryptographischen Umgebungen, in denen *nicht-uniforme* Widersacher betrachtet werden, o.B.d.A. diese Betrachtung auf deterministische Widersacher beschränkt werden kann, bei denen eine Zufallserzeugung fest eincodiert ist. Das gilt nicht notwendigerweise bei uniformen oder beschränkten nicht-uniformen Algorithmen.
- In den meisten kryptographischen Werken kann ein Beweis bezüglich der Sicherheit gegen *uniforme* Widersacher leicht auf einen Beiweis der Sicherheit gegen nicht-uniforme erweitert werden. Diese Erweiterungen funktionieren ausschließlich bei

[27] Da diese zusätzliche Eingabe nicht gleichverteilt sein muss, sind auch die so beeinflussten Algorithmen nicht mehr gleichverteilt, weshalb diese als „nicht-uniform“ bezeichnet werden.

Verwendung der *Black-Box*-Technik. Da in der hier vorgestellten Arbeit nur mit der *nicht-Black-Box*-Technik gearbeitet wird, kann der Sicherheitsbeweis gegen uniforme Widersacher nicht automatisch auf nicht-uniforme übertragen werden.

7.2.5 Universelles Argument

Universelle Argumente sind eine Variation von (interaktiven) Zero–Knowledge Argumenten, wie sie von Micali [Mic94] und Kilian [Kil92] definiert und konstruiert wurden. Vereinfacht gesagt ist ein *Universelles Argument* ein Zero–Knowledge Argument, um die Klassenzugehörigkeit einer super-polynomiellen Funktion $T(\cdot)$ in $\mathcal{N}time(T(n))$ (anstelle von $\mathcal{NP}$) zu beweisen.

Seien

- $\mathcal{U}$ ein gleichverteilter Wahrscheinlichkeitsraum über $\{0,1\}^*$,
- $R_{\mathcal{U}}$ eine Relation,
- M eine Turing Maschine mit der Beschreibung $\text{desc}(M)$[28],
- $x, z \in 0, 1^*$ Zeichenketten,
- t eine Dualzahl, wenn M (x, z) innerhalb t Schritten akzeptiert,
- $\langle \text{desc}(M), x, t\rangle$ eine Zufallsvariable bezüglich der Ausgabe von M bei Eingabe von x und t,
- $(\langle \text{desc}(M), x, t\rangle, z) \in R_{\mathcal{U}}$,
- $T_M(x, z)$ eine Funktion, die die Anzahl der Schritte ausgibt, die M bei Eingabe von (x, z) ausführt und
- die Sprache $L_{\mathcal{U}} = L(R_{\mathcal{U}})$.

Dann gilt

- $(\langle \text{desc}(M), x, t\rangle, z) \in R_{\mathcal{U}} \Longrightarrow T_M(x, z) \leq t$,
- $|\langle \text{desc}(M), x, t\rangle| = |\text{desc}(M)| + |x| + \log t$ und
- $L_{\mathcal{U}} \in \mathcal{N}time(2^n)$.

Weiter wird definiert,

- $T : \mathbb{N} \to \mathbb{N}$ als eine super-polynomielle Funktion (z.B. $T(n) = n^{\log\log n}$),

[28] siehe dazu Definition 2.11, Seite 12

- ${R_{\mathcal{U}}}^{T(n)}$ als Relation mit: $(\langle \text{desc}(M), x, t\rangle, z) \in {R_{\mathcal{U}}}^{T(n)}$ falls $(\langle \text{desc}(M), x, t\rangle, z) \in R_{\mathcal{U}}$ und $t \leq T(|\langle \text{desc}(M), x, t\rangle|)$,

Damit gilt wiederum

- ${L_{\mathcal{U}}}^{T(n)} = L({R_{\mathcal{U}}}^{T(n)})$,
- ${L_{\mathcal{U}}}^{T(n)}$ ist $\mathcal{N}time(T(n))$-vollständig und
- ${R_{\mathcal{U}}}^{2n} = R_{\mathcal{U}}$ und ${L_{\mathcal{U}}}^{2n} = L_{\mathcal{U}}$.

Ein *Universelles Argument System für* $\mathcal{N}time(T(n))$ ist grundsätzlich ein Zero–Knowledge Argument of Knowledge System für ${L_{\mathcal{U}}}^{T(n)}$ mit folgenden Veränderungen:

- Es werden strengere Anforderungen an die Effizienz eines ehrlichen Provers gestellt. Statt eine obere Schranke für die Komplexität des Provers bezüglich aller Eingaben einer bestimmten Länge festzulegen wird gefordert, dass die Laufzeit des Provers für die gemeinsamen Eingaben $\langle \text{desc}(M), x, t\rangle$ und z polynomiell in $T_M(x, z)$ sein muss und somit insbesondere polynomiell bezüglich t ist.
- Da das Zeugnis z eine Länge von $T(n)$ haben könnte, reicht möglicherweise Polynomialzeit nicht aus, um dieses niederzuschreiben. Daher kann der Extraktor eine Laufzeit von $T(n)^{O(1)}$ aufweisen.

Somit kann die formale Definition der Universellen Argumente erfolgen:

Definition 7.4 (Universelles Argument)
Ein Universelles Argument System für $\mathcal{N}time(T(n))$ *ist ein Paar von Algorithmen* (P,V), *welches folgende Eigenschaft erfüllt:*

Effizient verifizierbar: V *ist ein probabilistischer polynomiell–beschränkter interaktiver Algorithmus.*

Vollständigkeit eines relativ effizienten Provers: *Für jedes* $(\langle desc(M), x, t\rangle, z) \in {R_{\mathcal{U}}}^{T(n)}$ *gilt*

$$Pr[out_V(P(\langle desc(M), x, t\rangle, z), V(\langle desc(M), x, t\rangle)) = 1] = 1$$

Außerdem existiert ein Polynom $p(\cdot)$, *welches die maximale Gesamtlaufzeit von* $P(z)$ *bei gemeinsamer Eingabe* $(desc(M), x, t)$ *angibt:* $p(T_M(x, z)) \leq p(t)$.

Berechenbar Korrekt: *Für jede polynomiell große Schaltkreisfamilie* $\{P^*_n\}_{n\in\mathbb{N}}$ *und jedes* $y = \langle desc(M), x, t\rangle \in \{0,1\}^n \setminus L_{\mathcal{U}}$ *gilt*

$$Pr[out_V(P^*(y), V(y)) = 1] < \mu(n)$$

mit μ *als vernachlässigbarer Funktion.*

Schwacher Beweis des Proof of Knowledge: *Es existiert eine probabilistische Orakel Maschine E mit der Laufzeit $T(n)^{O(1)}$, so dass für jede polynomiell große Schaltkreisfamilie $\{P_n^*\}_{n\in\mathbb{N}}$ und jedes $y = \langle desc(M), x, t\rangle \in \{0,1\}^n$ gilt*

$$Pr[E^{P_n^*}(y) \in R_{\mathcal{U}}{}^{T(n)}(y)] \geq Pr[out_V(P^*(y), V(y)) = 1] + \mu(n)$$

mit μ als vernachlässigbarer Funktion.

Zu dieser Definition existiert folgender Satz.

Satz 7.2 (basierend auf [Mic94, Kil92])
Vorausgesetzt, es existiert eine Familie von Hashfunktionen, die kollisionsresistent gegen Schaltkreise der Größe $n^{\log n}$ ist. Dann gibt es ein Universelles Argument System für $\mathcal{N}time(n^{\log\log n})$, welches folgende Eigenschaften hat:

1. *Das System hat eine konstante Rundenzahl und ist ein Arthur-Merlin Beweissystem.*
2. *Das System ist Zeugnis–ununterscheidbar.*

Für jedes $\varepsilon > 0$ existiert so ein System $(desc(M), x, t)$ mit einer Kommunikationskomplexität von m^ε, wobei $m = |\langle desc(M), x, t\rangle|$ gilt.

Die Wahl von $\mathcal{N}time(n^{\log\log n})$ ist willkürlich gewählt, um die Darstellung zu vereinfachen. Sie kann durch $\mathcal{N}time(f(n))$ für jede Funktion $f(\cdot)$ mit $f(n) = n^{o(\log n)}$ ersetzt werden. Barak und Goldreich haben in [BG01a] gezeigt, dass unter der Standardannahme der Existenz von kollisionsresistenten Hashfunktionen gegen polynomiell große Schaltkreise Universelle Arguments für die Sprache $L_{\mathcal{U}}$ existieren. Der Beweis des Satzes 7.2 folgt direkt den Ausführungen in [Mic94, Kil92] und wird vollständig ersetzt durch die Ergebnisse in [BG01a]. Er wird hier nicht weiter ausgeführt.

Anmerkung: Es ist sehr unwahrscheinlich, die Korrektheitsbedingung zu einer statistischen Korrektheit zu verschärfen, die auch gegen *ineffiziente* Prover stand hält: Falls L einen interaktiven Beweis mit einem polynomiellen Verifier hat, gilt $L \in$ **PSPACE** und es wird angenommen, dass $\mathcal{N}time(n^{\log\log n}) \not\subseteq$ **PSPACE** gilt.

7.3 Zero–Knowledge Argument für uniforme Verifier

In diesem Abschnitt wird ein Arthur-Merlin Beweissystem für $\mathcal{NP}$ mit konstanter Rundenzahl konstruiert, welches für *uniforme* Verifier Zero–Knowledge ist (d.h. Verifier, die als Turing Maschinen implementierbar sind und keine externe Eingabe als Ratschlag erhalten). Das Protokoll verwendet einen *nicht-Black-Box* Simulator, der in streng polynomieller Zeit arbeitet. Das Protokoll in diesem Abschnitt erfüllt nicht alle Eigenschaften 1

– 5, die in der Definition 7.1 aufgeführt sind, da es lediglich bei *uniformen* Verifiern Zero–Knowledge ist und bei simultaner Ausführung nicht Zero–Knowledge bleibt (zumindest ist dies nicht bekannt). Dennoch stellt es die Hauptidee der Konstruktion anschaulich dar.

7.3.1 FLS Protokoll

In der vorgestellten Konstruktion wird eine Technik genutzt, die schon in anderen Zero–Knowledge Protokollen verwendet wurde und erstmals von **F**eige, **L**apidot und **S**hamir in [FLS00] erstmals vorgestellt wurde.

Die sogenannte FLS Technik erlaubt es, das Problem der Konstruktion eines Zero–Knowledge Proofs (oder Arguments) auf das Problem der Konstruktion zweier einfacher Objekte zu reduzieren: Ein *Zeugnis–ununterscheidbares* Beweissystem und ein (wie es hier genannt wird) *Generator Protokoll* (GenProt). Zeugnis–ununterscheidbare Systeme wurden in Abschnitt 3.5 beschrieben. Das Generator Protokoll wird später definiert (Definition 7.5).

Ein Zero–Knowledge Protokoll, welches mittels der FLS Technik konstruiert wird, wird als *FLS Protokoll* bezeichnet. Protokoll 7.1 beschreibt ein generisches FLS Zero–Knowledge Protokoll. Um eine spezielle Variante davon abzuleiten, müssen das Generator Protokoll, welches in den Schritten P,V1.x genutzt wird, die Sprache Λ und das Zeugnis–ununterscheidbare System in dem Schritt P,V2.x spezifiziert werden.

Protokoll 7.1 (Generisches FLS Zero–Knowledge Protokoll)
Gemeinsame Eingabe:

- 1^n: Sicherheitsparameter
- $x \in \{0,1\}^n$: Behauptung $x \in L$, die bewiesen werden soll.

Hilfseingabe des Provers: z: Zeugnis dafür, dass $x \in L$ gilt.

Schritte P,V1.x: (Generator ProtokollGenProt) Prover und Verifier einigen sich auf ein *Generator Protokoll* (GenProt). Dann sei $\tau = \text{transcript}_{(P,V)}(\cdot)$.[29]

Schritte P,V2.x: (Zeugnis–ununterscheidbarer Beweis für $x \in L$ oder $\tau \in \Lambda$) Der Prover beweist mit einem Zeugnis–ununterscheidbaren Argument dem Verifier, dass entweder $x \in L$ oder $\tau \in \Lambda$ gilt, wobei Λ eine feste Sprache ist, die Teil der Protokollspezifikation ist. Der Verifier akzeptiert, wenn der Beweis erfolgreich abgeschlossen wurde. Formal gilt, der Prover beweist $(x, \tau) \in M$, wobei die Sprache M wie folgt definiert ist: $(x, \tau) \in M$ genau dann, wenn $x \in L$ oder $\tau \in \Lambda$.

[29] transcript bezeichnet das Kommunikationsprotokoll aus Definition 2.10, Seite 12.

Generator Protokoll

Die folgende Definition des Generator Protokolls wird von der beabsichtigten Anwendung bestimmt. Die Anforderungen an ein Generator Protokoll garantieren dabei, dass das Resultat ein Zero–Knowledge Proof oder Zero–Knowledge Argument System ist, wenn es in die Basisversion von Protokoll 7.1 eingesetzt wird. Es wird hier nur ein *uniformes* Generator Protokoll definiert, da in diesem Abschnitt ein Protokoll entwickelt wird, welches Zero–Knowledge nur gegen Verifier ist, dessen Strategie in einer uniformen probabilistischen polynomiell–beschränkten Turing Maschine implementiert werden kann. Die formale Definition lautet:

Definition 7.5 (Uniformes Generator Protokoll)
Sei GenProt *ein Protokoll für zwei Teilnehmer, die Prover (P) und Verifier (V) genannt werden. Sei* $\Lambda \subseteq \{0,1\}^*$ *eine beliebige Sprache aus* $\mathcal{N}time(T(n))$ *für eine beliebige (polynomiell berechenbare) Funktion* $T : \mathbb{N} \longrightarrow \mathbb{N}$ *(z.B.* $T(n) = n^{\log\log n}$ *oder* $T(n) = n^3$*).* GenProt *heißt ein* uniformes Generator Protokoll *(in Bezug auf die Sprache* Λ*), wenn es folgende Bedingungen erfüllt:*

Korrektheit: *(Diese Bedingung garantiert, dass das Protokoll, in welches* GenProt *eingefügt wird, ebenfalls korrekt ist.) Sei* $\tau = \text{transcript}_{(P,V)}(\cdot)$ *aus* GenProt*. Wenn der Verifier der festgelegten Vorgehensweise folgt, gilt unabhängig vom Verhalten des Provers* $Pr[\tau \in \Lambda] < \mu(n)$ *mit* μ *als vernachlässigbarer Funktion.*

Uniforme Simulation: *(Diese Bedingung garantiert, dass das Protokoll, in welches* GenProt *eingefügt wird, Zero–Knowledge gegen uniforme Verifier ist.) Es existiert ein* Simulator S_{GenProt}*, der folgendes erfüllt: Sei* V^* *ein beliebiger polynomiell–beschränkter Verifier, der in weniger als* $2n$ *Bits beschrieben werden kann, wobei* n *der Sicherheitsparameter ist. Dann läuft* S_{GenProt} *bei Eingabe von* $desc(V^*)$ *in polynomieller Zeit in Bezug auf* V^* *und gibt folgendes Tupel* (v, σ) *aus:*

1. *v ist berechenbar ununterscheidbar von der Sicht des V^* bei einer Ausführung in* GenProt *mit P als Prover.*
2. *Sei* $\tau = \text{transcript}_{(P,V^*)}(\cdot)$ *, welches in der Sicht v enthalten ist. Dann ist* $\tau \in \Lambda$ *und σ ist ein Zeugnis dafür. Desweiteren muss die Zeit, in der „σ ist Zeugnis für τ“ verifiziert werden kann, polynomiell in Bezug auf die Laufzeit von V^* sein.*[30]

Anmerkung: Die beiden Anforderungen zusammen implizieren, dass Λ eine harte Sprache ist, weil in einer realen Ausführung mit einem ehrlichen Verifier das Kommunikationsprotokoll τ fast nie in Λ ist, wohingegen in der berechenbaren, ununterscheidbaren, simulierten Ausführung τ immer in Λ ist.

[30] Diese Anforderung ist wichtig, wenn $\Lambda \in \mathcal{N}time(T(\cdot))$ für eine super-polynomielle Funktion $T(\cdot)$ betrachtet wird.

Nun kann der Hauptsatz bewiesen werden, der für die FLS Technik benötigt wird.

Satz 7.3
Sei GenProt *ein Generator Protokoll in Bezug auf eine $\mathcal{N}time(t)$ Sprache Λ (wobei $T : \mathbb{N} \longrightarrow \mathbb{N}$ eine polynomiell berechenbare Funktion ist). Sei* ZuProt *ein Zeugnis–ununterscheidbares Beweis oder Argument System für $\mathcal{NP} \cup \mathcal{N}time(T)$ Sprachen. Sei L eine $\mathcal{NP}$ Sprache und* FLSProt *das Argument für L, welches dadurch entsteht, wenn* GenProt *und* ZuProt *in die Konstruktion 7.1 eingefügt werden. Dann ist* FLSProt *ein uniformes Zero–Knowledge Argument für L.*

Anmerkung: Satz 7.3 ist absichtlich so spezifiziert worden, dass es möglich ist, in einem uniformen Fall sowohl $\Lambda \in \mathcal{NP}$ als auch $\Lambda \in \mathcal{N}time(T)$ für einige super-polynomielle $T(\cdot)$, zu behandeln. Im ersten Fall ist es ausreichend, ein standard Zeugnis–ununterscheidbares Beweissystem für $\mathcal{NP}$ wie in [FS90a] zu nehmen. Im zweiten Fall wird ein Zeugnis–ununterscheidbares *Universelles Argument* gemäß Abschnitt 7.2.5 benutzt.

Beweisskizze:
Hier wird der Beweis von Satz 7.3 lediglich skizziert, da er vollständig von dem des nichtuniformen (Satz 7.5) ersetzt wird. Um zu zeigen, dass FLSProt ein Zero–Knowledge Argument ist, müssen die drei Eigenschaften Vollständigkeit, Korrektheit und Zero–Knowledge bewiesen werden.

Vollständigkeit: Die Vollständigkeit folgt aus der Tatsache, dass, falls die gemeinsame Eingabe $x \in L$ ist, die Aussage ($x \in L$ oder $\tau \in \Lambda$) wahr ist. Außerdem kann das Zeugnis z für x als ein Zeugnis für diese Aussage dienen.

Korrektheit: Angenommen, dass $x \notin L$ gilt und dass τ das Kommunikationsprotokoll der ersten Stufe (Schritt P,V1.x) von FLSProt bezeichnet. Bedingt durch die Korrektheitseigenschaft des Generator Protokolls GenProt, ist mit sehr hoher Wahrscheinlichkeit $\tau \notin \Lambda$. Daher wird die kombinierte Aussage ($x \in L$ oder $\tau \in \Lambda$) ebenfalls mit sehr hoher Wahrscheinlichkeit falsch sein und somit wird es dem Prover durch die Korrektheit des Zeugnis–ununterscheidbaren Beweis/Argument Systems nicht gelingen, den Verifier zu überzeugen.

Uniformes Zero–Knowledge: Um zu zeigen, dass FLSProt gegenüber einem uniformen Verifier Zero–Knowledge ist, wird der Simulator aus Protokoll 7.2 verwendet. Er benutzt den Simulator S_{GenProt} aus dem Generator Protokoll GenProt, um sowohl eine Simulation v für die erste Stufe zusammen mit einem Zeugnis σ zu erhalten als auch um mit dem Algorithmus des ehrlichen Provers aus dem Zeugnis–ununterscheidbaren System ZuProt die wahre Aussage ($x \in L$ oder $\tau \in \Lambda$) zu beweisen. Der Simulator benutzt das Zeugnis σ als Hilfseingabe für den Prover Algorithmus von ZuProt. Zusammen mit der Eigenschaft des effizienten Provers sichert die Vollständigkeit von ZuProt, dass der Simulator in strenger probabilistischer Polynomialzeit läuft. Die Zeugnis–ununterscheidbarkeit von ZuProt garantiert, dass die Ausgabe des Simulators tatsächlich

berechenbar ununterscheidbar von der Sicht des Verifiers in der realen Kommunikation ist. □

Protokoll 7.2 (Simulator für FLSProt)
Eingabe:

- 1^n: Sicherheitsparameter
- $x \in \{0,1\}^n$: Behauptung (simulierter Beweis für „$x \in L$“)
- $\mathsf{desc}(V^*)$: Beschreibung der Turing Maschine des Verifiers

Dabei bezeichne V_x^* den Verifier V^* mit fest eincodiertem x. Da V^* eine Turing Maschine ist, kann angenommen werden, dass die Beschreibung $\mathsf{desc}(V_x^*)$ von V_x^* höchstens $2n$ Bits lang ist.

Simulierte Schritte P,V1.x: (Simuliertes Generator Protokoll) Sei $(v, \sigma) = S_{\text{GenProt}}(V_x^*)$ mit S_{GenProt} als Simulator für das Generator Protokoll GenProt. Mit V_{xv}^* wird der restliche Verifier V_x^* mit fest eincodiertem v bezeichnet.

Simulierte Schritte P,V2.x: (Ehrlicher Zeugnis–ununterscheidbarer Beweis für ($x \in L$ **oder** $\tau \in \Lambda$)) Ausführung des Algorithmus des ehrlichen Provers für das Zeugnis–ununterscheidbare System ZuProt, um die Aussage ($x \in L$ oder $\tau \in \Lambda$) mit einem Zeugnis σ zu beweisen. V_{xv}^* wird als die Strategie des Verifiers benutzt. Dann bezeichne v' die Sicht von V_{xv}^* in dieser Interaktion.

Ausgabe: Das Tupel (v, v') mit den beiden Sichten dieser zwei Stufen.

7.3.2 Ein uniformes Generator Protokoll

Nachdem nun die FLS Technik beschrieben wurde, müssen die beiden Komponenten *Zeugnis–ununterscheidbares System* und *Generator Protokoll* spezifiziert werden. Da das Protokoll eine konstante Rundenzahl aufweisen und vom Arthur-Merlin Typ sein soll, müssen allerdings beide Komponenten konstant und Arthur-Merlin sein. Für das Zeugnis–ununterscheidbare System wird das in Abschnit 7.2.5 vorgestellte Zeugnis–ununterscheidbare Universelle Argument genutzt. Nach Satz 7.2 existiert unter den geforderten Voraussetzungen solch ein System für $\mathcal{N}time(n^{\log\log n})$. Daher verbleibt als Aufgabe, ein Generator Protokoll für beliebige Sprachen $\Lambda \in \mathcal{N}time(n^{\log\log n})$ mit konstanter Rundenzahl und Arthur-Merlin Eigenschaft zu konstruieren.

Protokoll 7.3 zeigt das geforderte uniforme Generator Protokoll. Es besteht aus zwei Runden, wobei die einzige Nachricht des Verifiers eine Zufallszeichenkette ist.

Protokoll 7.3 (Uniformes Generator Protokoll)
Gemeinsame Eingabe:
1^n: Sicherheitsparameter

Schritt P1: (Erzeugung eines wahllosen Commitments) Der Prover berechnet $w \in_R \mathrm{Com}(0^{3n})$[31] und sendet w an den Verifier.

Schritt V2: (Versenden einer Zufallszeichenkette) Der Verifier wählt eine Zeichenkette $r \in_R \{0,1\}^n$ und sendet diese.

Das Kommunikationsprotokoll diese uniformen Generator Protokoll ist das Tupel $\tau = (w, r)$.

Es erfolgt jetzt die Spezifikation der Sprache Λ. Es sei nochmal darauf hingewiesen, dass für eine Zeichenkette y die Funktion $\mathrm{Com}^{-1}(y)$ das eindeutige x derart bezeichnet, für dass das y das Commitment für x oder für 0^{3n} ist, falls es kein entsprechendes x gibt. Das heißt $x = \mathrm{Com}^{-1}(y)$, falls es ein s gibt, so dass $\mathrm{Com}(x, s) = y$ gilt. Die Entscheidung, ob $x = \mathrm{Com}^{-1}(y)$ gilt, erfolgt dabei in $\mathcal{NP}$.

Λ wird nun wie folgt definiert: Seien $\tau = (w, r)$ und $\Pi = \mathrm{Com}^{-1}(w)$. Dann ist $\tau \in \Lambda$, wenn die Turing Maschine, die durch Π beschrieben wird, und w als Eingabe erhält, ein r innerhalb von $|r|^{\frac{\log\log|r|}{5}}$ Schritten ausgibt.[32]

Es ist offensichtlich, dass $\Lambda \in \mathcal{N}time(n^{\log\log n})$ gilt. Nichtdeterministisch ist es möglich, $\Pi = \mathrm{Com}^{-1}(w)$ zu erhalten, womit dann genug Zeit vorhanden ist, die Turing Maschine, die durch Π beschrieben wird, in $n^{\frac{\log\log n}{5}}$ Schritten zu simulieren.

Damit kann nun der folgende Satz beschrieben werden.

Satz 7.4
Protokoll 7.3 ist ein uniformes Generator Protokoll (gemäß Definition 7.5).

Beweis:
Um Satz 7.4 zu beweisen, müssen die Korrektheitseigenschaft und die uniforme Simulierbarkeit des Protokolls 7.3 bewiesen werden.

Lemma 7.1
Sei P^* ein beliebiger (möglicherweise täuschender) Prover für das Protokoll 7.3 und bezeichne τ das Kommunikationsprotokoll von P^*'s Ausführung mit einem ehrlichen Verifier. Dann ist $Pr[\tau \in \Lambda] \leq 2^{-n}$.

Beweis:
Für jede erste Nachricht w des Provers sei $\Pi_w = \mathrm{Com}^{-1}(w)$ und sei $f(w)$ die Ausgabe der von Π_w beschriebenen Turing Maschine bei Eingabe von w nach $n^{\frac{\log\log n}{5}}$ Schritten, wenn die Maschine hält, und 0^n sonst. Für jede Zeichenkette w beträgt die Wahrscheinlichkeit, dass ein zufälliges $r \in_R \{0,1\}^n$ equivalent zu $f(w)$ ist, höchstens 2^{-n}. Daher

[31] Com bezeichnet das statistisch eindeutige Bit–Commitment–Scheme aus Definition 7.1, Seite 75

[32] Es wird nochmals darauf hingewiesen, dass $|r|^{\frac{\log\log|r|}{5}}$ willkürlich gewählt wurde. Es ist lediglich sicherzustellen, dass Λ in $\mathcal{N}time(n^{\log\log n})$ liegt.

ist unabhängig von der ersten Nachricht des Provers die Wahrscheinlichkeit, dass τ in Λ liegt, höchstens 2^{-n}. □

Lemma 7.2
Es existiert ein Simulator S_{GenProt} für Protokoll 7.3 für jeden probabilistischen polynomiell–beschränkten Verifier V^*, dessen Beschreibung höchstens $2n$ Bits lang ist, mit der Ausgabe

$$S_{\text{GenProt}}(V^*) = (v, \sigma)$$

Dabei ist v berechenbar ununterscheidbar von $\mathsf{view}_{V^*}(\cdot)$ und σ ist ein Zeugnis dafür, dass das durch v bedingte Kommunikationsprotokoll τ in Λ liegt.

Beweis:
Der Algorithmus 7.4 ist ein Simulator für das Protokoll 7.3.

Protokoll 7.4
Eingabe:

- 1^n: Sicherheitsparameter
- $\mathsf{desc}(V^*)$: Beschreibung einer probabilistischen polynomiell-beschränkten Turing Maschine. Die Länge von V^* ist höchstens $2n$.
- (Zufallserzeugung für V^*): Bezeichne m die Anzahl der Zufallbits, die V^* nutzt, und sei $\mathsf{PRG} : \{0,1\}^{\frac{n}{2}} \to \{0,1\}^m$ ein Pseudo Zufallsgenerator. Dann wird mit einem zufällig gewählten $u \in_R \{0,1\}^{\frac{n}{2}}$ ein Zufallsstring $s = \mathsf{PRG}(u)$ berechnet.

Simulierter Schritt P1: (Commitment für V^*'s Programm) Bezeichne Π den Algorithmus für die nächste Nachricht von V^* mit fest eincodiertem s als Zufall $\Pi = \mathsf{desc}(V_s^*)$. Es ist dabei zu beachten, dass $|\mathsf{desc}(\Pi)| < 3n$ Bits gilt ($\mathsf{desc}(V^*)$, $\mathsf{desc}(\mathsf{PRG})$ und die Saat u). Der Simulator berechnet $w \in_R \mathrm{Com}(\Pi)$ und „sendet“ w an den simulierten Schritt V2.

Simulierter Schritt V2: (Berechnung der Antwort von V^*) Der Simulator berechnet die Antwort auf w von V^* mit der Zufallszeichenkette s mit $r = \Pi(w)$.

Ausgabe: Das Tupel (v, σ) mit $v = (s, z)$ und σ als Zeugnis für $(w, r) \in \Lambda$ (σ enthält das Programm $\Pi = V_s^*$ und die Münzwürfe, die in der Berechnung des Commitments w benutzt wurden).

Die beiden Eigenschaften, die von (v, σ) vorausgesetzt werden, sind:

1. v ist berechenbar ununterscheidbar von $\mathsf{view}_{V^*}(\cdot)$ in der realen Ausführung. Das ergibt sich aus Folgendem:

   - PRG ist ein Pseudo Zufallsgenerator und dessen Ausgabe s somit berechenbar ununterscheidbar von V^*'s Zufallsband aus einer realen Ausführung.

- Com ist ein Commitment–Scheme und damit ist Com(Π) ununterscheidbar von Com(0^{3n}).

2. Für das v entsprechende Kommunikationsprotokoll $\tau = (w, r = V_s^*(w))$ gilt folgendes.

   - $\tau \in \Lambda$, da $w = \text{Com}(\Pi)$ mit der Eigenschaft, dass Π bei Eingabe von w innerhalb polynomiell vieler (und damit weniger als $n^{\frac{\log\log n}{5}}$) Schritten ein r ausgibt.
   - σ ist ein Zeugnis für $\tau \in \Lambda$. σ enthält Π und die Münzwürfe vom Commitment w und ist daher dafür ein Zeugnis. Zu beachten ist dabei, dass die Zeugniseigenschaft von σ für τ in polynomieller Zeit in Bezug auf die Laufzeit von V^* verifiziert werden kann.

Der Algorithmus 7.4 ist ein *nicht Black-Box* Simulator, der die Beschreibung des Verifier als Eingabe bekommt und diese anders benutzt, als eine simple Black-Box Funktion oder ein Orakel. Weiterhin ist wichtig zu beachten, dass der Algorithmus in probabilistischer *streng* polynomiell–beschränkter Zeit läuft. □

Aus Lemma 7.1 und Lemma 7.2 folgt die Korrektheit von Satz 7.4. ■

Als Resultat folgt, dass mit dem Generator Protokoll von Protkoll 7.3 und dem Universellen Argument System für $\mathcal{N}time(n^{\log\log n})$ aus Abschnitt 7.2.5, beides eingefügt in das generische FLS Protokoll 7.1, ein Zero–Knowledge Argument für $\mathcal{NP}$ zur Verfügung steht. Es hat dabei eine konstante Rundenzahl und ist vom Arthur-Merlin Typ. Der Simulator für das Zero–Knowledge Argument ist ein nicht Black-Box Simulatur und läuft dabei in probabilistischer *strenger* Polynomialzeit.

Tatsächlich ist das Protokoll nicht nur Zero–Knowledge gegen *voll* uniforme Verifier sondern auch gegen Verifier, die beschränkt *nicht* uniform sind. Das sind Verifier, die in $\frac{n}{2}$ Bits beschrieben werden können, wobei n der Sicherheitsparamter ist.[33]

Trotzdem ist ein uniformes und beschränkt nicht-uniformes Zero–Knowledge Protokoll nicht ausreichend. Zum Beispiel ist nicht bekannt, wie eine sequenzielle Komposition für uniforme oder auch beschränkt nicht-uniforme Zero–Knowledge Argumente bewiesen werden können [GK96b]. Im nächsten Abschnitt wird daher gezeigt, wie die bisherige Konstruktion modifiziert werden kann, um ein nicht-uniformes Zero–Knowledge Argument für $\mathcal{NP}$ zu erlangen.

[33] Der Wert $\frac{n}{2}$ ist leicht willkürlich. Durch das „Skalieren“ des Sicherheitsparameters für jedes Polynom $p(\cdot)$ erhält man ein Argument System, welches gegen Verifier sicher ist, die in höchstens $p(n)$ Bits beschrieben werden können.

7.4 Zero–Knowledge Argument für nicht-uniforme Verifier

In diesem Abschnitt wird ein nicht-uniformes Zero–Knowledge Argument System konstruiert, dass auf den Eigenschaften vom vorherigen Abschnitt aufbaut. Es erfüllt die Eigenschaften 1 – 2 und 4 – 5, die in der Definition 7.1 aufgeführt sind. Lediglich die Eigenschaft 3 (beschränkt simultan Zero–Knowledge) wird noch nicht erfüllt werden. Eine Modifikation, die dann auch noch die letzte Eigenschaft aufweist, wird im anschließenden Abschnitt gezeigt.

7.4.1 FLS' Protokoll

Genau wie das uniforme Protokoll aus Abschnitt 7.3 nutzt das nicht-uniforme die FLS Technik. Es muss jedoch eine leichte Abschwächung der Korrektheitsbedingung des Generator Protokolls aus Definition 7.5 vorgenommen werden. Es wird die Möglichkeit eingeräumt, dass das Kommunikationsprotokoll τ nicht nur mit vernachlässigbarer Wahrscheinlichkeit aus Λ stammen kann. Es wird jedoch auch in diesem Fall unmöglich sein, ein Zeugnis dafür vorzuzeigen, dass τ aus Λ stammt. Ein derartiges Generator Protokoll ist hinreichend, um in die Konstruktion 7.1 eingefügt zu werden, wenn in der zweiten Stufe (Schritte P,V2.x) statt eines Beweises der bloßen Mitgliedeigenschaft ein Zeugnis–ununterscheidbarer Beweis (oder Argument) *der exakten Kenntnis* (engl. Proof / Argument of Knowledge)[34] benutzt wird. Im Gegensatz dazu wird die Simulationseigenschaft der Definition 7.5 verschärft und verlangt zusätzlich die Simulierbarkeit für *nicht-uniforme* Verifier.

Ein Protokoll, welches das *nicht-uniforme Generator Protokoll* zusammen mit einem Zeugnis–ununterscheidbaren Argument of Knowledge auf diese Weise nutzt, wird ein *FLS'* Protokoll genannt. Zur Vollständigkeit wird die Beschreibung des FLS' Protokolls nachfolgend nochmals zusammenhängend aufgezeigt.

Protokoll 7.5 (Generisches FLS' Zero–Knowledge Protokoll)
Gemeinsame Eingabe:

- 1^n: Sicherheitsparameter
- $x \in \{0,1\}^n$: Behauptung $x \in L$, die bewiesen werden soll.

Hilfseingabe des Provers: z: Zeugnis dafür, dass $x \in L$ gilt.

Schritte P,V1.x: (Generator ProtokollGenProt') Prover und Verifier einigen sich auf ein nicht-uniformes *Generator Protokoll* (GenProt'). Dann sei $\tau = \text{transcript}_{(P,V)}(\cdot)$.

Schritte P,V2.x: (Zeugnis–ununterscheidbares Argument of Knowledgefür $x \in L$ oder $\tau \in \Lambda$) Der Prover beweist mit einem Zeugnis–ununterscheidbaren

[34] Zur Beschreibung von Argument of Knowledge siehe Abschnitt 7.2.3 auf Seite 76.

> Argument of Knowledge dem Verifier, dass entweder $x \in L$ oder $\tau \in \Lambda$ gilt, wobei Λ eine feste Sprache ist, die Teil der Protokollspezifikation ist. Der Verifier akzeptiert, wenn der Beweis erfolgreich abgeschlossen wurde. Formal gilt, der Prover beweist $(x, \tau) \in M$, wobei die Sprache M wie folgt definiert ist: $(x, \tau) \in M$ genau dann, wenn $x \in L$ oder $\tau \in \Lambda$.

Die formale Definition des nicht-uniformen Generator Protokolls sieht wie folgt aus:

Definition 7.6 (Nicht-Uniformes Generator Protokoll)
Sei GenProt' *ein Protokoll mit einem* Prover *und einem* Verifier. *Sei* $\Lambda \subseteq \{0,1\}^*$ *eine beliebige Sprache aus* $\mathcal{N}time(T(n))$ *für beliebige (polynomiell berechenbare) Funktionen* $T : \mathbb{N} \to \mathbb{N}$. GenProt' *heißt ein* ***Nicht-Uniformes Generator Protokoll*** *(in Bezug auf die Sprache* Λ*), wenn es folgende zwei Bedingungen erfüllt:*

Berechenbar Korrekt: *Für jeden (möglicherweise täuschenden) Prover* P^* *mit der Größe von* $T(n)^{O(1)}$ *gilt: Sei* $\tau = \text{transcript}_{(\mathsf{P}^*,\mathsf{V})}(\cdot)$ *von* GenProt'. *Dann ist die Wahrscheinlichkeit vernachlässigbar klein, dass* P^* *am Ende der Interaktion ein Zeugnis* $\tau \in \Lambda$ *erfolgreich ausgeben kann.*

Nicht-uniforme Simulation: *Es existiert ein probabilistischer polynomiell–beschränkter Simulator* $\mathsf{S}_{\text{GenProt}'}$, *der folgendes erfüllt:*

Sei V^* *ein polynomiell großer Verifer. Dann gibt* $\mathsf{S}_{\text{GenProt}'}$ *bei Eingabe von* $\mathsf{desc}(\mathsf{V}^*)$ *ein Tupel* (v, σ) *mit folgenden Eigenschaften aus:*

1. *v ist berechenbar ununterscheidbar von* $\mathsf{view}_{\mathsf{V}^*}(\cdot)$ *bei einer Ausführung von* GenProt' *mit dem beschriebenen Algorithmus des Provers.*
2. *Bezeichne τ das Kommunikationsprotokoll, welches in der Sicht v enthalten ist. Dann ist $\tau \in \Lambda$ und σ ein Zeugnis dafür. Desweiteren muss die Zeit, in der „σ ist Zeugnis für τ“ verifiziert werden kann, polynomiell in Bezug auf die Laufzeit von* V^* *sein.*

Anmerkungen: Das Erfordernis der berechenbaren Korrektheit bezieht sich auf $T(n)^{O(1)}$ große Widersacher statt auf polynomiell große. Falls Λ in $Ntime(T(n))$ liegt kann es bei einer super-polynomiellen Funktion $T(\cdot)$ sein, dass es super-polynomielle Schritte benötigt, nur um ein Zeugnis dafür niederzuschreiben.

Weiterhin sei darauf hingewiesen, dass im Gegensatz zur Definition 7.5 in dieser Definition nicht vorausgesetzt wird, dass das Entscheidungsproblem für Λ hart ist. Lediglich das Suchproblem in Λ, ein Zeugnis zu finden, muss hart sein.

Hier nun der Satz und Beweis über das nicht-uniforme Analogon zu Satz 7.3:

Satz 7.5
Sei GenProt' *ein nicht-uniformes Generator Protokoll in Bezug auf eine* $\mathcal{N}time(t)$ *Sprache* Λ *(wobei* $T : \mathbb{N} \longrightarrow \mathbb{N}$ *eine polynomiell berechenbare Funktion ist). Sei* ZuProt *ein Zeugnis–ununterscheidbares Argument System für* $\mathcal{NP} \cup \mathcal{N}time(T)$ *Sprachen. Sei* L *eine* $\mathcal{NP}$ *Sprache und* FLSProt' *das Argument für* L*, welches dadurch entsteht, wenn* GenProt' *und* ZuProt *in das Protokoll 7.5 eingefügt werden. Dann ist* FLSProt' *ein nicht-uniformes Zero–Knowledge Argument für* L*.*

Beweis:
Der Beweis von Satz 7.5 ist ähnlichem dem Beweis von Satz 7.3, der in Abschnitt 7.3.1 skizziert wurde. Um zu zeigen, dass FLSProt' ein Zero–Knowledge Argument ist, müssen die drei Eigenschaften Vollständigkeit, Korrektheit und Zero–Knowledge bewiesen werden. Davon wird zuerst die Korrektheit bewiesen, da in diesem Beweis der größte Unterschied zum Beweis von Satz 7.3 liegt.

Korrektheit: Es wird ein Widerspruchsbeweis geführt. Sei $P^{\text{FLSProt}'}$ ein polynomiell großer Prover für FLSProt'. Für den Widerspruch wird angenommen, dass die Ausführung von $P^{\text{FLSProt}'}$ und dem ehrlichen Verifier für beliebige $x \notin L$ mit einer nicht vernachlässigbaren Wahrscheinlichkeit von ε zu einem akzeptierenden Zustand führt. $P^{\text{FLSProt}'}$ wird genutzt, um einen täuschenden Prover $P^{\text{GenProt}'}$ zu konstruieren, der der berechenbaren Korrektheit des Generator Protokolls GenProt' widerspricht.

Der Prover $P^{\text{GenProt}'}$ arbeitet wie folgt: Bei der Interaktion mit dem Verifier des Protokolls GenProt' nutzt der Prover $P^{\text{GenProt}'}$ die Strategie des Provers $P^{\text{FLSProt}'}$ in der ersten Stufe von FLSProt' bei Eingabe von x. Bezeichne τ das Kommunikationsprotokoll und v die Sicht des Provers dieser Interaktion. Nachdem die Interaktion beendet ist, wird der Prover $P^{\text{GenProt}'}$ den restlichen Teil des Provers $P^{\text{FLSProt}'}$ mit dem Zustand v ausführen. Dieser zweite, restliche Teil des Provers wird als P^{ZuProt} bezeichnet, da er die Strategie der zweiten Stufe von FLSProt', der Zeugnis–ununterscheidbaren Beweisstufe, spezifiziert. Der Prover $P^{\text{GenProt}'}$ wendet dabei den Extraktor des Zeugnis–ununterscheidbaren Systems für P^{ZuProt} an. Dieses kostet $T(n)^{O(1)}$ Schritte. Wenn die Extraktion erfolgreich ist, erhält $P^{\text{GenProt}'}$ ein Zeugnis für die Behauptung „$x \in L$ oder $\tau \in \Lambda$“. Da $x \notin L$ angenommen wurde, bedeutet das, dass ein Zeugnis für $\tau \in \Lambda$ erlangt wurde.

Um den Widerspruchsbeweis zur berechenbaren Korrektheit von GenProt' zu führen, reicht es daher aus zu zeigen, dass die Ausführung mit nicht-vernachlässigbarer Wahrscheinlichkeit erfolgreich ist. Wenn die Wahrscheinlichkeit über das ganze Protokoll einer erfolgreichen Ausführung von $P^{\text{FLSProt}'}$ mit einem ehrlichen Verifier ε ist, dann gilt für mindestens $\frac{\varepsilon}{2}$ der Ausführungen der ersten Stufe, dass es eine $\frac{\varepsilon}{2}$ Wahrscheinlichkeit dafür gibt, die zweite Stufe erfolgreich zu beenden. Das impliziert mit einer Wahrscheinlichkeit von $\frac{\varepsilon}{2}$, dass es dem Prover P^{ZuProt} mit einer Wahrscheinlichkeit von $\frac{\varepsilon}{2}$ gelingt, den Verifier von „$x \in L$ oder $\tau \in \Lambda$“ zu überzeugen. Damit existiert aber eine nicht-vernachlässigbare Wahrscheinlichkeit einer erfolgreichen Extraktion, was

ein Widerspruch zur Definition darstellt.

Vollständigkeit: Die gemeinsame Eingabe seien x und ein Zeugnis z für $x \in L$. Der Algorithmus des ehrliche Provers in FLSProt' ist zuerst der Algorithmus des ehrlichen Provers aus GenProt' und anschließend der Algorithmus des ehrlichen Provers aus dem Zeugnis–ununterscheidbaren System, um mit dem Zeugnis z die Behauptung „$x \in L$ oder $\tau \in \Lambda$“ zu beweisen. Da das Zeugnis–ununterscheidbare System bei effizienten Provern die Vollständigkeit garantiert, wird auch FLSProt' nur polynomielle Zeit beanspruchen (wenn $\Lambda \in \mathcal{NP}$ gilt).

Zero–Knowledge: Algorithmus 7.6 ist ein Simulator für das Protokoll GenProt'. Das Verfahren entspricht dem Simulator im Beweis zum Satz 7.3. Der Simulator nutzt den Simulator S_{GenProt} aus dem Generator Protokoll GenProt', um zuerst eine Simulation v für die erste Stufe zusammen mit einem Zeugnis σ zu erhalten, welches mit dem Kommunikationsprotokoll in v vereinbar ist. Anschließend nutzt er den Algorithmus des ehrlichen Provers aus dem Zeugnis–ununterscheidbaren System ZuProt, um die wahre Behauptung „$x \in L$ oder $\tau \in \Lambda$“ zu beweisen. Dabei nutzt es das Zeugnis σ als Hilfseingabe für den Proveralgorithmus von ZuProt.

Protokoll 7.6 (Simulator für FLSProt')
Eingabe:

- 1^n: Sicherheitsparameter
- $x \in \{0,1\}^n$: Behauptung (simulierter Beweis für „$x \in L$“)
- $\mathsf{desc}(V^*)$: Beschreibung des polynomiell großen Verifiers

Dabei bezeichne V_x^* den Verifier V^* mit fest eincodiertem x.

Simulierte Schritte P,V1.x: (Simuliertes Generator Protokoll) Sei $(v, \sigma) = S_{\text{GenProt}'}(V_x^*)$ mit $S_{\text{GenProt}'}$ als Simulator für das Generator Protokoll GenProt'. Mit V_{xv}^* wird der restliche Verifier V_x^* mit fest eincodiertem v bezeichnet.

Simulierte Schritte P,V2.x: (Ehrlicher Zeugnis–ununterscheidbarer Beweis für ($x \in L$ **oder** $\tau \in \Lambda$)) Ausführung des Algorithmus des ehrlichen Provers für das Zeugnis–ununterscheidbare System ZuProt, um die Aussage ($x \in L$ oder $\tau \in \Lambda$) mit einem Zeugnis σ zu beweisen. V_{xv}^* wird als die Strategie des Verifiers benutzt. Dann bezeichne v' die Sicht von V_{xv}^* in dieser Interaktion.

Ausgabe: Das Tupel (v, v') mit den beiden Sichten dieser zwei Stufen.

Damit muss nur noch das folgende Lemma bewiesen werden:

Lemma 7.3
Seien V^* ein polynomiell großer Verifier für FLSProt', *$x \in L$ und (V_R, V'_R) die Zufallsvariable der Sicht von V^* bei der Kommunikation mit einem ehrlichen Prover und der Eingabe von x (wobei R für* real *im Gegensatz zur simulierten Ausführung steht, V_R die Sicht in der ersten Stufe und V'_R die Sicht in der zweiten Stufe ist.) Sei (V_S, V'_S) die Zufallsvariable von der Ausgabe des Protokolls 7.6 bei Eingabe von x. Dann sind (V_R, V'_R) und (V_S, V'_S) berechenbar ununterscheidbar.*

Beweis:
Der Beweis folgt einem Mischargument (engl. *hybrid argument*). Es wird bewiesen, dass beide Verteilungen von der gemischten Verteilung (V_S, V'_R) berechenbar ununterscheidbar sind, wobei V_S die Simulation der ersten Stufe und V'_R die reale Ausführung der zweiten Stufe repräsentiert. Das heißt, (V_S, V'_R) ist die Ausgabe eines „Hybridsimulators“, der den Simulator von GenProt' in der ersten Stufe aber das Zeugnis für x (anstatt des Zeugnisses für τ) in der zweiten Stufe nutzt.

Die Verteilung (V_S, V'_R) ist aufgrund der Zeugnis–ununterscheidbaren Eigenschaft von ZuProt von (V_S, V'_S) berechenbar ununterscheidbar. Falls es einen Algorithmus U gibt, der zwischen diesen beiden Verteilungen mit einer Wahrscheinlichkeit von ε unterscheiden kann, muss es insbesondere eine einzelne Sicht $v = V_S$ von der ersten Stufe mit der Wahrscheinlichkeit ε geben, dass U zwischen (V_S, V'_R) und (V_S, V'_S) unterscheiden könnte. Seien τ das Kommunikationsprotokoll, das in v enthalten ist, und σ das Zeugnis für τ, wie es vom Simulator geliefert wird. Seien U^*_v der Unterscheidungsschaltkreis U mit fest eincodiertem v und V^*_v der restliche Verifier V^* mit fest eincodiertem v. Dann könnte U^*_v mit einer Wahrscheinlichkeit von ε zwischen einer Kommunikation von V^*_v mit einem ehrlichen Prover des Zeugnis–ununterscheidbaren Systems, das σ als Hilfseingabe nutzt, und einer Kommunikation von V^*_v mit einem ehrlichen Prover des Zeugnis–ununterscheidbaren Systems, das z (als Zeugnis für x) als Hilfseingabe nutzt, unterscheiden. Dies ist aber aufgrund der Zeugnis–ununterscheidbaren Eigenschaft von ZuProt nur mit vernachlässigbarer Wahrscheinlichkeit möglich, so dass ε ebenfalls nur vernachlässigbar ist.

Die Verteilung (V_S, V'_R) ist aufgrund der Simulationseigenschaft des Generator Protokoll GenProt' berechenbar ununterscheidbar von (V_R, V'_R). Angenommen, es gibt einen Algorithmus U, der zwischen beiden Verteilungen mit einer Warhscheinlichkeit von ε unterscheiden kann. Dann kann ein Unterscheidungsschaltkreis U' konstruiert werden, der wie folgt im Widerspruch zur Simulationseigenschaft des Generator Protokolls steht. Der Unterscheidungsschaltkreis U' hat das Zeugnis z fest eincodiert. Wenn er eine Zeichenkette v als Eingabe erhält, führt er den Algorithmus des ehrlichen Provers aus dem Zeugnis–ununterscheidbaren Beweis für die Behauptung „$x \in L$ oder $\tau \in \Lambda$“ aus (wobei τ das in v enthaltene Kommunikationsprotokoll ist) und dabei das Zeugnis y nutzt. Er simuliert den Verifier, indem er den restlichen Verifier V^* mit der Eingabe v nutzt. Bezeichne v' die Sicht des Verifiers in dem Zeugnis–ununterscheidbaren Beweis. Der

Unterscheidungsschaltkreis U' liefert $U(v, v')$ zurück. Wie zu sehen, kann U' mit einer Wahrscheinlichkeit von ε zwischen der Sicht von V^* in einer realen Ausführung von GenProt' und der Ausgabe von $S_{\text{GenProt}'}(V^*)$ unterscheiden. Aufgrund der Simulationseigenschaft von GenProt' ist ε daher vernachlässigbar. □

Mit dem Nachweis der Zero–Knowledge Eigenschaft durch Lemma 7.3 sowie der zuvor erfolgten Vollständigkeit und Korrektheit ist der Beweis von Satz 7.5 abgeschlossen. ■

7.4.2 Ein nicht-uniformes Generator Protokoll

Nachdem Satz 7.5 bewiesen ist, verbleibt noch die Konstruktion eines *nicht-uniformen* Generator Protokolls in Bezug auf eine $Ntime(n^{\log\log n})$ Sprache Λ. Diese basiert auf der der uniformen Konstruktion von Protokoll 7.3.

Es sei nochmals daran erinnert, wie das Protokoll 7.3 die Bedingung der uniformen Simulation erfüllt. Der Simulator (Algorithmus 7.4) berechnet ein Commitment w für die „nächste-Nachricht Funktion“ des uniformen Verifiers als eine eigene Nachricht in dem Protokoll. Das Problem bei der Verwendung des statistisch eindeutigen Commitment–Schemes ist, dass notwendiger Weise gelten muss, dass die Länge des Commitments w länger sein wird als die Länge der Beschreibung der nächste-Nachricht Funktion. Sobald aber *nicht-uniforme* Verifier berücksichtigt werden, muss für die Beschreibung der nächste-Nachricht Funktion eine beliebige polynomielle Länge zulässig sein. Insbesondere kann diese länger als die Kommunikationskomplexität des hier betrachteten Protokolls sein. Statt eines statistisch eindeutigen Commitment–Schemes wird daher ein lediglich berechenbar eindeutiges Commitment–Scheme verwendet.

Ein berechenbar eindeutiges Commitment–Scheme, das es erlaubt, Commitments von Nachrichten zu erstellen, die länger als die Ausgabe sind, kann so konstruiert werden, dass eine Komposition eines standardmäßigen statistisch eindeutigen Commitment–Scheme mit einer kollisions-resistenten Hashfunktion erstellt wird. Dabei wird ersichtlich, warum die Korrektheitseigenschaft der Definition des Generator Protokolls GenProt' abgeschwächt werden muss: Da lediglich ein berechenbar eindeutiges Commitment–Scheme verwendet wird, kann auch das dadurch erstellte Protokoll nur berechenbar korrekt sein. Das Protkoll 7.7 ist das konstruierte nicht-uniforme Generator Protokoll GenProt'.

Protokoll 7.7 (Nicht-Uniformes Generator Protokoll)
Gemeinsame Eingabe:
1^n: Sicherheitsparameter

Schritt V1: (Auswahl einer Hashfunktion) Der Verifier wählt eine zufällige Hashfunktion $h \in_R \mathcal{H}_n$ und sendet h dem Prover.

Schritt P2: (Erzeugung eines wahllosen Commitments) Der Prover berechnet ein zufälliges $w \in_R \text{Com}(h(0^{3n}))$ und sendet w an den Verifier.

Schritt V3: (Versenden einer Zufallszeichenkette) Der Verifier wählt eine Zeichenkette $r \in_R \{0,1\}^n$ und sendet diese.

Das Kommunikationsprotokoll dieses nicht-uniformen Generator Protokolls ist das Tripel $\tau = (h, w, r)$.

Nun wird die Sprache Λ definiert. Sei $\tau = (h, w, r)$. Dann gilt $\tau \in \Lambda$, wenn ein Programm Π existiert, so dass $w = \mathrm{Com}(h(\Pi))$ gilt und $\Pi(w)$ eine Ausgabe r innerhalb von $|r|^{\frac{\log\log|r|}{5}}$ Schritten erzeugt. Dieses kann in $\mathcal{N}time(n^{\frac{\log\log n}{5}})$ verifiziert werden. Ein Zeugnis für $(h, w, r) \in \Lambda$ ist ein Tupel (Π, s), so dass $w = \mathrm{Com}(\Pi, s)$ gilt und $\Pi(w)$ eine Ausgabe r innerhalb von $|r|^{\frac{\log\log|r|}{5}}$ erzeugt.

Anmerkung: Es kann sein, dass Λ einfach zu entscheiden ist (tatsächlich könnte $\Lambda = \{0,1\}^*$ sein), aber die Korrektheitsbedingung von Definition 7.6 verlangt lediglich, dass es unmöglich ist, ein Zeugnis dafür zu finden.

Satz 7.6
Protokoll 7.7 ist ein nicht-uniformes Generator Protokoll in Bezug auf eine Sprache Λ (entsprechend Definition 7.6).

Beweis:
Zum Beweis, dass Protokoll 7.7 die Definition 7.6 erfüllt, müssen zwei Eigenschaften erfüllt sein: Die berechenbare Korrektheit und die nicht-uniforme Simulation.

Berechenbare Korrektheit: Sei P^* eine $n^{O(\log\log n)}$ große Strategie für Protokoll 7.7 und bezeichne τ das Kommunikationsprotkoll der Ausführung von P^* mit dem ehrlichen Verifier. Dann ist die Wahrscheinlichkeit vernachlässigbar klein, dass P^* ein Zeugnis für $\tau \in \Lambda$ auf sein Hilfsband schreibt.

Andernfalls sei angenommen, dass P^* ein Zeugnis mit nicht vernachlässigbarer Wahrscheinlichkeit ε habe. Dann gilt für mindestens einen $\frac{\varepsilon}{2}$ Teil von $h \in_R \mathcal{H}_n$, dass P^* mit einer Wahrscheinlichkeit von mindestens $\frac{\varepsilon}{2}$ ein Zeugnis für das Kommunikationsprotokoll, welches mit h beginnt, besitzt. Sei ein derartiges $h \in \mathcal{H}_n$ fest. O.B.d.A. sei angenommen, dass P^* deterministisch sei. Dann ist die Nachricht w, welches die Antwort auf h ist, ebenfalls fest. Mit dieser Annahme und der Wahl von $r \in_R \{0,1\}^n$ ist der Prover mit einer Wahrscheinlichkeit von $\frac{\varepsilon}{2}$ in der Lage, ein Programm Π auszugeben (innerhalb von $|r|^{\frac{\log\log|r|}{5}}$ Schritten), so dass w ein Commitment von $h(\Pi)$ und $\Pi(w) = r$ ist. Das bedeutet, dass bei einer unabhängigen Wahl von $r \in_R \{0,1\}^n$ und $r' \in_R \{0,1\}^n$ mit einer Wahrscheinlichkeit von $\frac{\varepsilon^2}{4}$ zwei Programme Π, Π' erlangt werden, so dass w ein Commitment sowohl zu $h(\Pi)$ als auch zu $h(\Pi')$ ist und dass $\Pi(w) = r$ und $\Pi'(w) = r'$ gilt. Da w ein statistisch eindeutiges Commitment–Scheme ist, folgt daraus $h(\Pi) = h(\Pi')$. Da $r \neq r'$ angenommen werden kann (mit einer Wahrscheinlichkeit von $1-2^{-n}$), folgt daraus $\Pi(w) \neq \Pi'(w)$ und damit, dass Π und Π' unterschiedliche Programme sind. Das bedeutet aber, dass Π und Π' *Kollisionen* für h sind. Damit existiert ein $n^{O(\log\log n)}$ großer

Algorithmus, der für einen ε großen Teil von $h \in_R \mathcal{H}_n$ eine Kollision für h mit einer Wahrscheinlichkeit von $O(\varepsilon^2)$ erzeugt. Das steht im Widerspruch zur Kollisionsresistenz gegen $n^{\log n}$ große Widersacher der $\mathcal{H}_n$ Familie. □

Nicht-uniforme Simulation: Der Beweis dafür, dass Protokoll 7.7 die Simulationseigenschaft erfüllt, ist ähnlich dem der uniformen (Lemma 7.2). Es muss gezeigt werden, dass ein Simulator $S_{\text{GenProt}'}$ für Protokoll 7.7 existiert, so dass für jeden polynomiell großen Verifier V^* $S_{\text{GenProt}'}(V^*)$ ein Tupel (v, σ) ausgibt, so dass v berechenbar ununterscheidbar von $\text{view}^*_V(\cdot)$ ist und dass σ ein Zeugnis dafür ist, dass das zu v gehörende Kommunikationsprotokoll τ aus Λ ist.

Protokoll 7.8
Eingabe:

- 1^n: Sicherheitsparameter
- V^*: Ein polynomiell großer Schaltkreis (o.B.d.A. ist V^* deterministisch).

Simulierter Schritt V1: (Auswahl einer Hashfunktion) Berechnung von $h = V^*()$: Die erste Nachricht von V^*.

Simulierter Schritt P2: (Commitment für V^*'s Programm) Bezeichne Π den Algorithmus für die nächste Nachricht von V^*. Berechnung eines zufälligen $w \in_R \text{Com}(h(\Pi), s)$, wobei $s \in_R \{0,1\}^{poly(n)}$ Münzwürfe für das Commitment–Scheme sind.

Simulierter Schritt V3: (Berechnung der Antwort von V^*) Berechnung der Antwort von V^* auf die Nachricht w. Das heißt, $r = \Pi(w) = V^*(w)$.

Ausgabe: Das Tupel (v, σ) mit v als die Sicht w und $\sigma = (\Pi, s)$ als Zeugnis für $(h, w, r) \in \Lambda$.

Protokoll 7.8 ist ein Simulator für Protokoll 7.7. Die Ausgabe von Protokoll 7.8 ist ein Tupel (v, σ), so dass $v = w$ ist (was das Kommunikationsprotokoll (h, w, r) mit $h = V^*(\cdot)$ und $r = V^*(w)$ impliziert) und $\sigma = (\Pi, s)$ ein Zeugnis für $v \in \Lambda$ ist. Die zwei Eigenschaften, die von (v, σ) benötigt werden, sind folgende:

1. v ist berechenbar ununterscheidbar von $\text{view}^*_V(\cdot)$ in einer realen Ausführung. Dieses folgt aus dem Umstand, dass Com ein Commitment–Scheme ist und somit $\text{Com}(h(\Pi))$ ununterscheidbar von $\text{Com}(h(O^{3n}))$ ist.

2. Das zu v passende Kommunikationsprotokoll τ ist aus Λ und σ ist ein Zeugnis dafür. Das zu v passende Kommunikationsprotokoll ist (h, w, r). Das Tupel $\sigma = (\Pi, s)$ ist ein Zeugnis dafür, da $w = \text{Com}(h(\Pi), s)$ ist und $\Pi(w)$ in einer polynomiellen Anzahl von Schritten (und damit kleiner als $|r|^{\frac{\log\log|r|}{5}}$) r ausgibt. Zu beachten ist dabei, dass die Zeit zum Verifizieren, dass σ ein Zeugnis für τ ist, tatsächlich polynomiell zur Laufzeit von V^* ist.

Protokoll 7.8 ist wie Protokoll 7.4 ein *nicht Black-Box* Simulator. Weiterhin ist anzumerken, das es in *streng* probabilistischer polynomiell–beschränkter Zeit läuft. ■

7.5 Beschränkt simultan Zero–Knowledge

In diesem Abschnitt wird gezeigt, wie das Zero–Knowledge Protokoll des vorherigen Abschnitts modifiziert werden muss, um ein Zero–Knowledge Protokoll zu erhalten, welches beschränkt simultan betrieben wird. Ein Zero–Knowledge Protokoll ist beschränkt simultan Zero–Knowledge, wenn es bei einer n-fachen simultanen bzw. parallelen oder auch konkurrierenden Ausführung Zero–Knowledge bleibt, wobei n den Sicherheitsparameter bezeichnet. Da dieser Sicherheitsparameter beliebig skaliert werden kann, bedeutet das, dass für *jedes* feste Polynom $p(\cdot)$ ein Protokoll konstruiert werden kann, das bei $p(n)$-facher Ausführung Zero–Knowledge bleibt.

Dieses Protokoll hier wird abhängig von $p(\cdot)$ sein und insbesondere eine Kommunikationskomplexität größer als $p(n)$ haben. Das steht im Gegensatz zu dem unbeschränkt simultanen Zero–Knowledge von [DNS99], wo ein einzelnes Protokoll existiert, welches für jede $p(n)$-fache Ausführung für jedes Polynom $p(\cdot)$ Zero–Knowledge bleibt.

Beschränkt simultan Zero–Knowledge wird hier nachfolgend definiert. Vorher wird jedoch formal definiert, was eine simultane Ausführung bedeutet:

Definition 7.7 (Simultane Ausführung)
Seien (P, V) ein Protokoll mit zwei Teilnehmern, V^* eine beliebige interaktive Maschine und $\{(a_i, b_i)\}_{i=1}^t$ die Menge von t Eingaben für das Protokoll (P, V). Eine t-fach simultane Ausführung *von (P, V), die von V^* mit der Eingabe $\{(a_i, b_i)\}_{i=1}^t$ koordiniert wird, liegt vor, wenn folgende Eigenschaften erfüllt sind:*

1. *Ausführung von t unabhängigen Kopien von P, wobei die i-te Kopie a_i als Eingabe erhält.*
2. *V^* erhält die $b_1, \ldots, b_t$.*
3. *In jedem Schritt gibt V^* die Nachricht (i, m) aus. Die i-te Kopie von P erhält dabei die Nachricht m. Der Verifier V^* bekommt die Antwort des Provers.*

Definition 7.8 (Beschränkt simultan Zero–Knowledge)
Sei (P, V) ein interaktives Beweis oder Argument System für eine Sprache $L = L(R)$. Dann heißt (P, V) ***beschränkt simultan Zero–Knowledge****, wenn ein probabilistischer polynomiell–beschränkter Algorithmus S existiert, so dass für jeden polynomiell großen V^* und jede Liste $\{(x_i, y_i)\}_{i=1}^n$ mit $(x_i, y_i) \in R$ die beiden folgenden Zufallsvariablen berechenbar ununterscheidbar sind:*

1. *Die Sicht von V^* in einer n-fach simultanen Ausführung von (P, V) mit der Eingabe $\{(x_i, y_i)\}_{i=1}^n$.*

2. $S(V^*, x_1, \ldots, x_n)$

Das hier vorgestellte Protokoll ist ein FLS' Protokoll und benutzt ein nicht-uniformes Generator Protokoll, welches dem von Protokoll 7.7 ähnelt. Tatsächlich ist der einzige Unterschied zwischen den Generator Protokollen des Protokolls aus diesem Abschnitt und des Protokolls 7.7, dass eine längere Zeichenkette r als Nachricht des Verifiers (Schritt V2) benutzt wird. Das heißt, dass $r \in_R \{0,1\}^{n^4}$ statt $r \in_R \{0,1\}^n$ benutzt wird.[35] Protokoll 7.9 ist das modifizeirte Generator Protokoll.

Protokoll 7.9 (Nicht-Uniformes Generator Protokoll)
(für beschränkt simultanes Zero–Knowledge)

Gemeinsame Eingabe:
1^n: Sicherheitsparameter

Schritt V1: (Auswahl einer Hashfunktion) Der Verifier wählt eine zufällige Hashfunktion $h \in_R \mathcal{H}_n$ und sendet h dem Prover.

Schritt P2: (Erzeugung eines wahllosen Commitments eines Hashes) Der Prover berechnet ein zufälliges $w \in_R \mathrm{Com}(h(0^{3n}))$ und sendet w an den Verifier.

Schritt V3: (Versenden einer langen Zufallszeichenkette) Der Verifier wählt eine Zeichenkette $r \in_R \{0,1\}^{n^4}$ und sendet diese.

Das Kommunikationsprotokoll dieses nicht-uniformen Generator Protokolls ist das Tripel $\tau = (h, w, r)$.

Darüber hinaus wird auch die Definition der Sprache Λ verändert. Sei $\tau = (h, w, r)$. Dann gilt $\tau \in \Lambda$, wenn ein Programm Π existiert, so dass $w = \mathrm{Com}(h(\Pi))$ gilt und *eine Zeichenkette y existiert* mit $|y| \leq \frac{|r|}{2}$ und $\Pi(w, y)$ eine Ausgabe r innerhalb von $|r|^{\frac{\log\log|r|}{5}}$ Schritten erzeugt. Dieses kann in $\mathcal{N}time(n^{\frac{\log\log n}{5}})$ verifiziert werden. Ein Zeugnis für $(h, w, r) \in \Lambda$ ist ein Tripel (Π, s, y), so dass $w = \mathrm{Com}(\Pi, s)$ und $|y| \leq \frac{|r|}{2}$ gilt sowie $\Pi(w, y)$ eine Ausgabe r innerhalb von $|r|^{\frac{\log\log|r|}{5}}$ erzeugt. Das bedeutet, dass die Sprache so verändert wird, dass das verschlüsselte Programm Π, welches r ausgibt, nicht nur das w als Eingabe erhält sondern darüber hinaus eine zusätzliche Eingabe y bekommen kann, solange diese nicht zu lang ist (genauer nicht länger als $|y| \leq \frac{|r|}{2}$).

Das modifizierte Protokoll ist weiterhin ein nicht-uniformes Generator Protokoll, wie folgender Satz zeigt:

Satz 7.7
Protokoll 7.9 ist ein nicht-uniformes Generator Protokoll in Bezug auf Λ.

[35] Der Wert n^4 ist ebenfalls ein bisschen willkürlich. Es wurde nicht versucht, die Relation zwischen der Kommunikationskomplexität und der Anzahl der simultanen Sitzungen zu optimieren.

Beweisskizze:
Der Beweis wird lediglich skizziert, da er überwiegend identisch zu dem des Satzes 7.6, Seite 94, ist. Der Umstand, dass die Nachricht r länger ist, ändert nichts an dem Beweis, so dass lediglich gezeigt werden muss, dass die Änderung von Λ nichts schadet.

In der Tat ist der Beweis der nicht-uniformen Simulation unverändert, da der Simulator 7.8, der in dem Beweis zum Satz 7.6 präsentiert wurde, ebenfalls ein Simulator für das Protokoll 7.9 ist. Das Zeugnis, das von diesem Simulator ausgegeben wird, ist auch für die modifizierte Sprache Λ ein gültiges Zeugnis (es verwendet einfach ein leeres Wort für die Zeichenkette y).

Der Beweis der berechenbaren Korrektheit ist leicht verändert. Zur Erinnerung sei darauf hingewiesen, dass der Beweis zu Satz 7.6 darauf beruht, einen täuschenden Prover in einen Algorithmus zu überführen, der Kollisionen der Hashfunktion sucht. Dabei wurde folgende Beobachtung gemacht: Wenn für einige Werte w gilt, dass ein Programm Π existiert mit $\Pi(w) = r$ und bei einer zufälligen Wahl von $r' \in_R \{0,1\}^n$ mit einer Wahrscheinlichkeit von ε ein zweites Programm Π' existiert mit $\Pi'(w) = r'$, dann unterscheidet sich Π mit einer Wahrscheinlichkeit von mindestens $\varepsilon - 2^{-n}$ von Π'. Hier wird diese Beobachtung genutzt, um zu zeigen wie ein täuschender Prover genutzt wird, um ein Kollisionspaar Π und Π' für die Hashfunktion zu erhalten.

Die entscheidende Beobachtung dabei ist folgende: Wenn für einige Werte w gilt, dass ein Programm Π existiert mit $\exists y \in \{0,1\}^{m/2};\ \Pi(w,y) = r$ und bei einer zufälligen Wahl von $r' \in_R \{0,1\}^m$ mit einer Wahrscheinlichkeit von ε ein zweites Programm Π' existiert mit $\exists y' \in \{0,1\}^{m/2};\ \Pi'(w,y') = r'$, dann unterscheidet sich Π mit einer Wahrscheinlichkeit von mindestens $\varepsilon - 2^{-m/2}$ von Π'. Denn für den Fall $\Pi = \Pi'$ gilt $r' \in \Pi(w, \{0,1\}^{m/2})$, was höchstens mit einer Wahrscheinlichkeit von $2^{-m/2}$ eintritt. □

7.5.1 Beschränkt simultan Zero–Knowledge Argument

Das zu konstruierende Zero–Knowledge Protokoll wird das Protokoll 7.9, und das Zeugnis–ununterscheidbare Universelle Argument System von Satz 7.2 in der Ausführung von Protokoll 7.5, Seite 88, benutzen. Der Beweis dafür, dass es beschränkt simultan Zero–Knowledge ist, ist nicht so modular aufgebaut wie in den vorherigen Beweisen. Das heißt, dass das vollständige Zero–Knowledge Argument als Ganzes erörtert wird, anstatt Aussagen der einzelnen Komponenten zu beweisen. Die in Protokoll 7.10 folgende Konstruktion ist ein beschränkt simultanes Zero–Knowledge Argument für $\mathcal{NP}$.

Nachrichtenlänge: Es ist notwendig, die Länge der Nachrichten, die vom Prover gesendet werden, zu betrachten. Nachrichten, die eine Länge von weniger als n^2 Bits aufweisen, werden als *kurz* bezeichnet. Es wird bei der anschließenden Beobachtung entscheidend sein, dass alle Nachrichten der Prover von Protokoll 7.10 „kurz“ sind. (Das gilt nicht für den Verifier, da die Nachricht r aus Schritt V1.3 eine Länge von n^4 hat.) In der ersten Stufe kann angenommen werden, dass die Nachrichten der Prover in Schrit

P1.2 (das Commitment $w = \mathrm{Com}(h(0^{3n}))$) kurz ist, da es Konstruktionen von Commitment–Schemes gibt, bei denen die Länge des Commitments einer Nachricht der Länge n höchstens n^2 ist. In der zweiten Stufe wird der Umstand ausgenutzt, dass gemäß Satz 7.2, Seite 80, für jedes $\varepsilon > 0$ ein Zeugnis–ununterscheidbares Universelles Argument mit einer Kommunikationskomplexität von m^ε existiert, wobei m die Länge der Instanz ist. Daher kann sichergestellt werden, dass alle Nachrichten aller Prover in der zweiten Stufe eine Länge von höchstens n^2 haben. (Auch wenn die Länge der Behauptung, die in dieser Stufe bewiesen wird, länger als n^4 ist.)

Protokoll 7.10 (Beschränkt simultanes Zero–Knowledge Protokoll)
Gemeinsame Eingabe:

- 1^n: Sicherheitsparameter
- $x \in \{0,1\}^n$: Behauptung $x \in L$, die bewiesen werden soll.

Hilfseingabe des Provers: z: Zeugnis dafür, dass $x \in L$ gilt.

Schritte P,V1.x: ***Generator Protokoll***

Schritt V1.1: (Auswahl einer Hashfunktion) Der Verifier wählt eine zufällige Hashfunktion $h \in_R \mathcal{H}_n$ und sendet h dem Prover.

Schritt P1.2: (Erzeugung eines wahllosen Commitments eines Hashes) Der Prover berechnet ein zufälliges $w \in_R \mathrm{Com}(h(0^{3n}))$ und sendet w an den Verifier. *(Kurze Nachricht.)*

Schritt V1.3: (Versenden einer langen Zufallszeichenkette) Der Verifier wählt eine Zeichenkette $r \in_R \{0,1\}^{n^4}$ und sendet diese.

Das Kommunikationsprotokoll dieser Stufe ist das Tripel $\tau = (h, w, r)$.

Schritte P,V2.x: ***Zeugnis–ununterscheidbares Universelles Argument***

Schritte P,V2.1.x: (Zeugnis–ununterscheidbares Universelles Argument für $x \in L$ oder $\tau \in \Lambda$) Der Prover beweist mit einem Zeugnis–ununterscheidbaren Universellen Argument dem Verifier, dass entweder $x \in L$ oder $\tau \in \Lambda$ gilt. *(Alle Nachrichten des Provers hier sind kurz.)*

Der Hauptsatz dieses Abschnitts lautet wie folgt:

Satz 7.8
Protokoll 7.10 ist ein beschränkt simultanes Zero–Knowledge Argument.

Beweis:
Das Protokoll 7.10 baut auf den Protokollen 7.5 von Seite 88, 7.7 von Seite 93 und 7.9 von Seite 97 auf, dessen Vollständigkeit und Korrektheit bewiesen wurde. Da sich bei einer

parallelen bzw. simultanen Ausführung an den Eigenschaften der Vollständigkeit und Korrektheit nichts ändert, verbleibt die Notwendigkeit, die Zero–Knowledge Eigenschaft des Protokolls 7.10 zu beweisen.

Für den Beweis wird ein Simulator benötigt, dessen Ausgabe untersucht werden muss. Zuerst wird ein Überblick über die Funktionalität des Simulators gegeben. Anschließend erfolgt die detaillierte Beschreibung und Analyse.

Überblick über den Simulator Sei V^* ein polynomiell großer Algorithmus, der die Strategie eines Verifier in einer n-fach simultanen Ausführung von Protokoll 7.10 beschreibt. Zur Erinnerung an die Definition 7.7, Seite 96, sei nochmal darauf hingewiesen, dass bei einer n-fach simultanen Ausführung eines Protokolls die Ausführung von n Sitzungen mit eingeschlossen sind, die je nach Entscheidung des Verifieres verschachtelt sind. Mit m_j wird die j-te Nachricht aller Prover (über alle Sitzungen hinweg) bezeichnet. Jede Nachricht m_j bezieht sich dabei auf eine bestimmte Sitzung i (wobei $1 \leq i \leq n$ gilt). Da die Ausführungen simultan ablaufen gilt nicht notwendiger Weise, dass die einzelnen Sitzungen konsekutiv sind, aber jede einzelne ist für sich geordnet (d.h. die erste Nachricht in Sitzung i kommt immer vor der zweiten Nachricht der Sitzung i). Die Sicht des Verifiers in der Ausführung ist die Folge $(m_1, \ldots, m_{cn})$, wobei c die konstante Anzahl der Nachrichten der Prover in Protokoll 7.10 ist. Die Aufgabe des Simulators muss daher sein, eine Folge $(m_1, \ldots, m_{cn})$ zu erzeugen, die ununterscheidbar von der Sicht des Verifiers in der realen Ausführung ist.

Anmerkung: In diesem Abschnitt wird i als Index einer Sitzung (d.h. $1 \leq i \leq n$) und j und k als Index aller Nachrichten der Prover (d.h. $1 \leq j, k \leq cn$) sein. Teilweise werden die Nachrichten der Prover oder des Verifiers mit der Sitzungsnummer und Schrittnummer (z.B. „r ist die Nachricht des Verifiers in Schritt V1.3 in der i-ten Sitzung“) bezeichnet.

Der Simulator wird diese Folge inkrementell erzeugen. Das heißt, dass die Nachricht m_j nur erzeugt wird, wenn schon $(m_1, \ldots, m_{j-1})$ erzeugt wurde und dass eine einmal erzeugte Nachricht nicht rückwirkend wieder verändert wird. Daher ist es möglich und sinnvoll sich den Simulator so vorzustellen, als wenn er mit dem Verifier V^* interagiert. Natürlich besteht der Unterschied dieser Interaktion mit einer realen Interaktion darin, dass der Simulator den „unfairen“ Vorteil hat, den Code von V^* zu kennen.

Der erste naive Ansatz ist zu versuchen, den Simulator von Protokoll 7.8 von Seite 95 n-fach unabhängig aufzurufen. Das kann jedoch nicht funktionieren. Angenommen, dass die k-te Nachricht des Provers in Schritt P1.2 in der i-ten Sitzung berechnet werden muss. Im vorherigen Schritt (Schritt V1.1) sendet der Verifier eine Hashfunktion h worauf in Schritt P1.2 ein Commitment w für $h(\Pi)$ mit Π als irgendein Programm berechnet wird. Wenn den Anweisungen des einzelnen Simulators gefolgt wird, besteht Π einfach aus dem Programm des restlichen Verifiers V^* an diesem Punkt. Sei nun angenommen, dass der Verifier sich entscheidet, an diesem Punkt einige weitere Sitzungen zu planen (d.h. unterschiedliche zu i). Das heißt, dass die nächsten Nachrichten $m_{k+1}, \ldots, m_{j-1}$,

die der Verifier empfängt, Teile von Sitzungen sind, die sich von i unterscheiden (wobei m_j die nächste Nachricht des Provers nach m_k ist, die sich auf die i-te Sitzung bezieht). Angenommen, dass der Verifier die Zeichenkette r entsprechend Schritt V1.3 der i-ten Sitzung sendet, dann berechnet der Verifier r als Funktion der empfangenen Nachrichten $m_{k+1}, \ldots, m_{j-1}$. Da Π der restliche Verifier am Punkt k ist, gilt *nicht* $\Pi(w) = r$ und somit auch nicht $(h, w, r) = \Lambda$, was bedeutet, dass der Versuch, den Simulator einfach unabhängig laufen zu lassen, fehl schlägt. Trotzdem ist der Ansatz in Ordnung: Es folgt nämlich $r = \Pi(w, m_{k+1}, \ldots, m_{j-1})$. Das heißt, r ist die Antwort von V^*, nachdem er die Nachrichten $m_1, \ldots, m_{j-1}$ empfangen hat, wobei $m_1, \ldots, m_{j-1}$ bereits Teil der Beschreibung von Π und $m_k = w$ ist.

Die entscheidende Beobachtung hierbei ist, dass aus der Tatsache, dass alle Nachrichten der Prover „kurz“ sind (mit einer Länge von höchstens n^2) und dass es höchstens cn Nachrichten gibt (wobei c die konstante Anzahl der Runden in Protokoll 7.10 ist), folgt, dass

$$|m_{k+1}| + \ldots + |m_{j-1}| \leq O(n^3) < \frac{n^4}{2} = \frac{|r|}{2}$$

gilt.

Somit existiert ein Zeugnis dafür, dass das Kommunikationsprotokoll (h, w, r) aus Λ ist, weil es gemäß der Definition von Λ in diesem Abschnitt nämlich nicht erforderlich ist, $\Pi(w) = r$ zu beweisen sondern lediglich zu zeigen, dass $\Pi(w, y) = r$ für *beliebige kurze Zeichenketten* y gilt (d.h. für y mit $|y| < \frac{|r|}{2}$).

Aus diesem Grund kann die Simulation an dieser Stelle fortgeführt werden. In dieser Sitzung wird einfach die Strategie des ehrlichen Provers für das Zeugnis–ununterscheidbare Universelle Argument benutzt, zusammen mit Π, $(m_{k+1}, \ldots, m_{j-1})$ und den Münzwürfen, die in der Berechnung des Commitments w verwendet wurden.

Somit kann die Beschreibung des hier verwendeten Simulators erfolgen:

Protokoll 7.11 (Beschreibung des Simulators)
Eingabe:

- $x_1, \ldots, x_n$: Zu beweisende Behauptung in der i-ten Sitzung, dass $x_i \in L$ gilt.
- $\mathsf{desc}(V^*)$: Beschreibung des polynomiell großen Verifiers, der n simultane Ausführungen koordiniert.

Initialisierung: Der Simulator erzeugt eine leere Tabelle $\mathcal{A}$ der Länge n. Nachdem der simulierte Schritt P1.2 der i-ten Sitzung verarbeitet und die Nachricht w berechnet wurde, enthält $\mathcal{A}[i]$ den Inhalt (Π, s), so dass $w = \mathrm{Com}(h(\Pi), s)$ gilt, wobei h die vom Verifier gewählte Hashfunktion aus Schritt V1.1 der i-ten Sitzung ist. Nachdem der Simulator vom

Verifier die Nachricht r in Schritt V1.3 erhalten hat, fügt er eine Zeichenkettte y mit der Eigenschaft $\Pi(w, y) = r$ zu $\mathcal{A}[i]$ hinzu, die eine Länge von weniger als $\frac{n^4}{2}$ hat. Das heißt, dass $\mathcal{A}[i]$ an diesem Punkt ein Zeugnis (Π, s, y) für $(h, w, r) \in \Lambda$ enthält, wobei h, w und r die jeweiligen Nachrichten aus den Schritten V1.1, P1.2 bzw. V1.3 der simulierten i-ten Sitzung sind.

Simulation der Schritte: Für $j = 1, \ldots, cn$ berechnet der Simulator die j-te simulierte Nachricht des Provers m_j auf folgende Weise:

Die vorher berechneten Nachrichten $(m_1, \ldots, m_{j-1})$ werden V^* zugeführt und damit die j-te Nachricht des Verifiers (i, m) erzeugt (wobei i die Sitzung ist, für die die Nachricht des Verifiers bestimmt ist). Daraufhin wird die Nachricht m_j entsprechend des aktuellen Schritts in dem simulierten Beweis der i-ten Sitzung berechnet:

Schritt P1.2: (Commitment des Programms) Wenn die Nachricht des Verifiers für den Schritt V1.1 der i-ten Sitzung bestimmt ist, dann berechnet der Simulator folgendes:

- Bezeichne h die Nachricht des Verifiers.
- Berechnung der Beschreibung des folgenden Programms Π:
  $$\Pi(w, y) \text{ erzeugt } V^*(m_1, \ldots, m_{j-1}, w, y).$$
  Anmerkung: Die Werte $m_1, \ldots, m_{j-1}$ wurden vorher berechnet.
- $w = \mathrm{Com}(h(\Pi), s)$ wobei s die Zufallszeichenketten für das Commitment ist.
- Speicherung von Π und s in $\mathcal{A}[i]$.
- Die j-te Nachricht m_j wird zu w.

Schritt V1.3: (Nachrichtenempfang) Wenn die Nachricht des Verifiers für den Schritt V1.3 der i-ten Sitzung bestimmt ist, dann berechnet der Simulator folgendes:

- Bezeichne r die Nachricht des Verifieres, wobei $r = V^*(m_1, \ldots, m_{j-1})$ ist.
- Bezeichne k den Index über alle Nachrichten der Prover in Schritt P1.2 in derselben Sitzung, das heißt, m_k ist die Nachricht w derselben Sitzung.
- Bezeichne $y = (y_1, \ldots, y_{j-k-1})$ die Folge $(m_{k+1}, \ldots, m_{j-1})$. Dabei ist zu beachten, dass $j \leq cn$ ist und daher $|y| \leq cn \cdot 10n^2 < \frac{n^4}{2}$ für genügend große n gilt.
- y wird zu $\mathcal{A}[i]$ hinzugefügt. Dabei ist zu beachten, dass $\mathcal{A}[i]$ bereits (Π, s) enthält, so dass $w = \mathrm{Com}(h(\Pi), s)$ und $\Pi(w, y) = r$ gelten.

Schritte P2.x: (Zeugnis–ununterscheidbares Universelles Argument) In diesen Schritten wird einfach den Strategien der ehrlichen Prover für Zeugnis–ununterscheidbare Universelle Argumente gefolgt, um zu beweisen, dass entweder $x_i \in L$ gilt oder dass das Kommunikationsprotokoll τ der ersten Stufe der i-ten Sitzung aus Λ stammt. Dabei ist zu beachten, dass $\mathcal{A}[i]$ als ein Zeugnis dafür verwendet werden kann, dass das Kommunikationsprotokoll der ersten Stufe tatsächlich aus Λ ist.

Das Lemma, welches beweisen werden muss, lautet wie folgt:

Lemma 7.4
Bezeichne S einen Simulator gemäß Protokoll 7.11. Dann gilt für jeden polynomiell großen Verifier V^* und für jede Eingabe $\{(x_i, y_i)\}_{i=1}^n$ mit „y_i ist ein Zeugnis für $x_i \in \Lambda$“, dass die beiden folgenden Zufallsvariablen X und Y berechenbar ununterscheidbar sind:

- $X = \mathsf{view}_{V^*}(\{(x_i, y_i)\}_{i=1}^n)$ *einer n-fach simultanen Ausführung von Protkoll 7.10.*
- $Y = \mathsf{out}_S(V^*, (x_1, \ldots, x_n))$.

Beweis:
Bezeichne S' einen beliebigen Algorithmus, der bei Eingabe von $(V^*, \{(x_i, y_i)\}_{i=1}^n)$ derselben Strategie wie S bei Eingabe von V^* und $(x_1, \ldots, x_n)$ folgt, mit der Ausnahme, dass S' bei der Simulation der Schritte der Zeugnis–ununterscheidbarer Universellen Argumente (Schritte P2.x) in der i-ten Sitzung y_i als Eingabe für den ehrlichen Prover Algorithmus verwendet.

Sei $Z = \mathsf{out}_{S'}(V^*, (V^*, \{(x_i, y_i)\}_{i=1}^n))$. Das Lemma wird bewiesen, indem gezeigt wird, das w berechenbar ununterscheidbar sowohl von X als auch von Y ist und somit kein Algorithmus existiert, der X von Y berechenbar unterscheiden kann.

1. Z ist berechenbar ununterscheidbar von Y:

   Es ist zu beachten, dass der einzige Unterschied zwischen Z und Y das Zeugnis ist, welches als Eingabe für den Prover des Zeugnis–ununterscheidbaren Universellen Arguments dient. Darüber hinaus gilt, dass auf die von dem Prover des Zeugnis–ununterscheidbaren Systems genutzte Zufallszeichenkette von anderen Teilen der Simulation weder zugegriffen noch von diesen verwendet wird. Damit folgt die Behauptung grundsätzlich dem Umstand, dass die Zeugnis–ununterscheidbarkeit unter simultaner Komposition abgeschlossen ist. Da das Protokoll darüber hinaus Nachrichten enthält, die sich nicht auf das Zeugnis–ununterscheidbare Universelle Argument beziehen, wird der Beweis noch weiter geführt.

   Seien die Sitzungen in Bezug auf die Ausführungsreihenfolge im ersten Schritt der zweiten Stufe (der Zeugnis–ununterscheidbaren Universellen Argument Stufe) sortiert. Bezeichne Z_i eine Verteilung, bei der in den ersten i Sitzungen der Strategie

von S und in den letzten $n-i$ Sitzungen der Strategie von S' gefolgt wird, wobei $Z_0 = Z$ und $Z_n = Y$ gilt. Dann reicht es aus zu beweisen, dass für alle $0 \leq i < n$ Z_i berechenbar ununterscheidbar von Z_{i+1} ist.

Angenommen, es gibt einen Unterscheidungsalgorithums U, der mit der Wahrscheinlichkeit ε zwischen Z_i und Z_{i+1} unterscheidet. Sei weiterhin eine Auswahl von Münzwürfen beliebig aber fest angenommen, die in allen Sitzungen vor der $i+1$-ten und in der ersten Sitzung verwendet wird, in der U zwischen Z_i und Z_{i+1} abhängig von diesen Münzwürfen mit einer Wahrscheinlichkeit von ε unterscheiden kann. Dabei ist zu beachten, dass nun die zu beweisende Behauptung in Y und Z identisch ist. Da die Ausführung aller Sitzungen außer der i-ten in Y und Z dieselben sind und da die Strategien für diese Ausführungen eine Funktion der Eingabe ist, könnten die Münzwürfe und möglicherweise die öffentlichen Nachrichten der i-ten Sitzung aus dem Unterscheidungsalgorithums U einen Unterscheider für das Zeugnis–ununterscheidbare Universelle Argument System machen.

2. Z ist berechenbar ununterscheidbar von X:

   Der einzige Unterschied zwischen Z und X besteht darin, dass das Commitment in Schritt P1.2 über $h(\Pi)$ anstatt über $h(0^{3n})$ berechnet wird. Darüber hinaus gilt, dass auf die von Z und X genutzte Zufallszeichenkette von anderen Teilen der Simulation weder zugegriffen noch von diesen verwendet wird (da die Zeugnis–ununterscheidbare Stufe y_i als Zeugnis verwendet). Die Behauptung wird unten nochmals bewiesen, obwohl sie prinzipiell von der Sicherheit des Commitment–Scheme bezüglich Mehrfachabfragen impliziert wird.

   Nun werden die Sitzungen bezüglich der Reihenfolge der Nachrichten in Schritt P1.2 sortiert. X_i wird definiert als eine Verteilung, in der in den ersten i Sitzungen $\mathrm{Com}(h(0^{3n}))$ und in den letzten $n-i$ Sitzungen die Strategie des Simulators S mit $\mathrm{Com}(h(\Pi))$ benutzt wird. Dabei ist zu beachten, dass $X_0 = X$ und $X_n = Z$ gilt.

   Angenommen, es gibt einen Unterscheidungsalgorithums U, der mit der Wahrscheinlichkeit ε zwischen X_i und X_{i+1} unterscheidet. Sei weiterhin eine Auswahl von Münzwürfen beliebig aber fest angenommen, die in allen Sitzungen mit Ausnahme der i-ten verwendet wird, so dass U zwischen X_i und X_{i+1} abhängig von diesen Münzwürfen mit einer Wahrscheinlichkeit von ε unterscheiden kann. U kann dann in einen Unterscheider zwischen einem Commitment $h(\Pi)$ und einem Commitment $h(0^{3n})$ überführt werden, indem alle Nachrichten vor der Nachricht von Schritt P1.2 der i-ten Sitzung fest eincodiert werden, wodurch die späteren Nachrichten eine Funktion der Eingabe und der vorhergehenden Nachrichten sind.

Somit ist X berechenbar ununterscheidbar von Y und der Simulator 7.11 garantiert die Zero–Knowledge Eigenschaft des Protokolls 7.10 von Seite 99. □

Abschließend ist Satz 7.8 und damit auch das Hauptergebnis in Satz 7.1 bewiesen, dass das Protokoll 7.10 ein beschränkt simultanes Zero–Knowledge Argument ist. ■

8 Zusammenfassung und weiterführende Themen

Zero–Knowledge Arguments sind eine für die praktische Anwendung sinnvolle und notwendige Erweiterung der Zero–Knowledge Proofs. Das ist nicht nur an der Vielzahl wissenschaftlicher Veröffentlichungen zu diesem Thema abzulesen, die hier nur zu einem kleinen Teil angeführt bzw. wiedergegeben werden konnten. Aus dem Umstand, dass sich die vorliegende Arbeit fast ausschließlich mit den Zero–Knowledge Arguments befasst, darf aber nicht geschlossen werden werden, dass eine echte Teilung zwischen den Beweis- und den Argumentensystemen existiert. Die Forschung in beiden Bereichen ist sehr eng miteinander verbunden und viele neue Erkenntnisse, die bei der Veröffentlichung einer Arbeit z.B. für Beweissysteme bekannt werden, gelten auch für Argumentensysteme und umgekehrt.

Als ein Beispiel darf dazu die von Barak 2001 [Bar03] erstmals veröffentlichte Erkenntnis sein, dass nachweisbar interaktive Argumentensysteme mit konstanter Rundenzahl und einem Simulator, der in strenger Polynomialzeit arbeitet, nicht mit einem Black-Box Simulator konstruierbar sind. Dieses Ergebnis, dass einen erheblichen Einfluss auf die Art und Weise hat, wie ein Argumentensystem zu konstruieren ist, ist auch auf Beweissysteme übertragbar.

Alle drei in dieser Arbeit vorgestellten Argumentensysteme als jeweilige Vertreter für perfekte bzw. berechenbare Zero–Knowledge Arguments in polynomieller bzw. konstanter Rundenzahl dienen sowohl der wissenschaftlichen Forschung als auch den praktischen Informatikern als Grundlagen für weitere Arbeiten. In der Praxis ist es damit möglich geworden, unter Ausnutzung der speziellen Eigenschaften moderner Computertechniken und Kommunikationsmöglichkeiten kryptographische Verfahren zu entwickeln, die genau auf den Einsatzzweck abgestimmt werden können.[36]

Über die in dieser Arbeit aufgeführten Verfahren und Veröffentlichungen hinaus existieren im Bereich Zero–Knowledge Proofs und Zero–Knowledge Arguments aber noch eine Vielzahl weiterer Arbeiten. Aufgrund des fast unüberschaubaren Umfangs sollen und können nicht alle Forschungsgebiete von Bedeutung erwähnt werden. Es existieren jedoch Teilbereiche innerhalb der Kryptographie, die sich zwar nicht direkt oder allein mit Zero–Knowledge Arguments beschäftigen, die aber Erkenntnisse mit Tragweite auch für die Argumentensysteme hervorgebracht haben.

Davon sollen einige beispielhafte Themen zum Abschluss aufgeführt werden.

[36] Z.B. werden bei leistungsschwachen Teilnehmern wie Chipkarten Argumentensysteme eingesetzt, die die notwendige Sicherheit durch eine polynomielle Rundenzahl erreichen, da die Kommunikationszeiten unkritisch und Angriffe auf die Kommunikationswege überschaubar sind. Im Gegensatz dazu wird bei einer störanfälligen Funk- oder Internetverbindung, bei der die Teilnehmer eine starke Rechenleistung haben und Angreifer mit unbekanntem Leistungsvermögen vorhanden sind, Argumentensysteme eingesetzt, deren Sicherheit durch die hohe Rechenleistung erreicht wird und damit deren Rundzahl auf eine geringe Konstante begrenzt werden kann.

Nicht–interaktive Zero–Knowledge Proof

Diese Beweissysteme bestehen ebenfalls aus einem Prover und einem Verifier, die eine gemeinsame, gleichverteilte Zufallszeichenkette lesen können. Im Gegensatz zu den in dieser Arbeit vorgestellten interaktiven Beweissystemen wird hier jedoch nur eine Nachricht vom Prover an den Verifier geschickt. Ein Nachrichtenfluss vom Verifier zum Prover existiert nicht. Nicht–interaktive Zero–Knowledge Proof Systeme wurden von Blum, Feldman und Micali in [BFM88] eingeführt. Eine weitere Konstruktion wurde von Feige, Lapidot und Shamir in [FLS00] beschrieben, von der Teile im Verlauf der vorliegenden Arbeit in dem Verfahren von Barak genutzt werden.

Ehrlicher Verifier versus unehrlicher Verifier

Von Goldreich, Sahai und Vadhan wird in [GSV98] gezeigt, wie ein interaktives Beweissystem, welches statistisch Zero–Knowledge bei einem ehrlichen Verifier ist, in ein Beweissystem überführt werden kann, welches statistisch Zero–Knowledge bei jedem (auch täuschenden) Verifier ist. Dabei werden weder die Voraussetzungen bezüglich der Berechungskomplexität noch bezüglich der Komplexität eines Verifiers verschärft.

Concurrent Zero–Knowledge

Dabei handelt es sich um Systeme, bei denen mehrere Prover gleichzeitig, überlappend und im Allgemeinen unabhängig voneinander mit einem Verifier kommunizieren. Bei der simultanen Ausführung von Zero–Knowledge Beweissystemen können solche Systeme, die nur für die Kommunikation eines Provers mit einem Verifier konstruiert wurden, versagen. Insbesondere können zwei Prover, die in Absprache untereinander mit einem Verifier interagieren, die Nachrichten des Verifers an den zweiten Prover dazu nutzen, um Informationen zur Täuschung durch den ersten Prover zu gewinnen. Die erste Veröffentlichung über Probleme, die beim Interagieren mehrerer Prover mit einem Verifier auftreten (Multi-Prover Zero–Knowledge Proof), stammt von Ben-Or, Goldwasser, Kilian und Wigderson in [BOGKW88]. Erstmals vorgestellt wurden Zero–Knowledge Systeme, die auch bei simultaner bzw. konkurrierender Ausführung ihre Eigenschaften bewahren, 1998 von Dwork, Naor und Sahai in [DNS99] mit einem Zeitschrankenmodell. Später veröffentlichten Kilian, Petrank und Rackoff in [KPR98] sowie Richardson und Kilian in [RK99] Verfahren in einem Standardmodell mit polynomieller Rundenzahl. Kilian und Petrank verbesserten die Komplexität in [KP00] auf polylogarithmische Runden. Dwork und Sahei zeigten in [DS98], wie das Zeitschrankenmodell weiter reduziert werden kann. Damgård zeigt in [Dam99] ein Modell, welches unter bestimmten Voraussetzungen ein konkurrierendes Zero–Knowledge Argument System mit konstanter Rundenzahl ergibt. Nicht zuletzt ist dass von Barak in [Bar03] beschriebene Protokoll ebenfalls für eine paralelle Ausführung geeignet.

Resettable Zero–Knowledge

Eine Verschärfung der Anforderungen an ein interaktives Beweissystem stellt die Möglichkeit dar, dass eine der beteiligten Parteien von dem „Gegner“ zurück gesetzt wird, um einen Angriff auf die Verschlüsselung durchzuführen. Dieser Umstand hat bezüglich der Probleme herkömmlicher Zero–Knowledge Systeme Ähnlichkeiten mit dem parallelen Zugriff mehrerer Prover auf einen Verifier. Veröffentlichungen dazu sind unter anderem von Canetti, Goldreich, Goldwasser und Micali in [CGGM00], von Barak, Goldreich, Goldwasser und Lindell in [BGGL01] sowie von Micali und Reyzin in [MR01a] erschienen. Dort werden Algorithmen beschrieben, die derartigen Angriffen standhalten.

Literatur

[Bab85] Babai, László: *Trading group theory for randomness.* In: *STOC '85* [STO85], Seiten 421–429.

[Bar03] Barak, Boaz: *How to Go Beyond the Black-Box Simulation Barrier*, Januar 2003. Überarbeitete Version. Vorher erschienen in FOCS 2001, Seiten 106–115, Oktober 2001.

[BC86] Brassard, Gilles und Claude Crépeau: *Non-Transitive Transfer of Confidence: A* Perfect *Zero-Knowledge Interactive Protocol for SAT and Beyond.* In: *FOCS '86* [FOC86], Seiten 188–195.

[BCC88] Brassard, Gilles, David Chaum und Claude Crépeau: *Minimum disclosure proofs of knowledge.* Journal of Computer and System Sciences, 37(2):156–189, Oktober 1988.

[BCY89] Brassard, Gilles, Claude Crépeau und Moti Yung: *Everything in NP can be argued in perfect zero-knowledge in a bounded number of rounds.* In: *Advances in Cryptology – EUROCRYPT '89*, Nummer 434 in *Lecture Notes in Computer Science*, Seiten 192–195, Berlin Heidelberg, Deutschland, 1989. Springer-Verlag.

[BCY91] Brassard, Gilles, Claude Crépeau und Moti Yung: *Constant-round perfect zero-knowledge computationally convincing protocols.* Theoretical Computer Science, 84(1):23–52, 1991.

[Bel02] Bellare, Mihir: *A Note on Negligible Functions.* Technischer Bericht, University of California at San Diego, La Jolla, California, USA, März 2002.

[BFM88] Blum, Manuel, Paul Feldman und Silvio Micali: *Non-interactive zero-knowledge and its applications.* In: *STOC '88* [STO88], Seiten 103–112.

[BG92] Bellare, Mihir und Oded Goldreich: *On Defining Proofs of Knowledge.* In: *CRYPTO '92* [CRY92], Seite 390 ff.

[BG01a] Barak, Boaz und Oded Goldreich: *Universal Arguments and their Applications.* Electronic Colloquium on Computational Complexity (ECCC), 093, 2001.

[BG01b] Bellare, Mihir und Shafi Goldwasser: *Lecture Notes on Cryptography.* Vorlesung zu: „Cryptography and Cryptanalysis“ sowie „Cryptography and network security“, MIT Laboratory of Computer Science, Cambridge, Massachusetts, USA, August 2001.

[BGGL01] Barak, Boaz, Oded Goldreich, Shafi Goldwasser und Yehuda Lindell: *Resettably-Sound Zero-Knowledge and its Applications*. Cryptology ePrint Archive, Report 2001/063, 2001. – URL http://eprint.iacr.org/2001/063.ps.gz – Zugriffsdatum 22.11.2002.

[BJY97] Bellare, Mihir, Markus Jakobsson und Moti Yung: *Round-Optimal Zero-Knowledge Arguments Based on any One-Way Function*, July 1997. Überarbeitete Version. Vorher erschienen in Advances in Cryptology – EUROCRYPT '97, Nummer 1233 in Lecture Notes in Computer Science, Springer-Verlag, 1997.

[BL02] Barak, Boaz und Yehuda Lindell: *Strict Polynomial-time in Simulation and Extraction*. In: *STOC '2002* [STO02], Seiten 484–493.

[Blu81] Blum, Manuel: *Coin flipping by telephone*. In: *Advances in Cryptology – CRYPTO '81*, Seiten 11–15, August 1981.

[BOGKW88] Ben-Or, Michael, Shafi Goldwasser, Joe Kilian und Avi Wigderson: *Multi-prover interactive proofs: How to remove intractability assumptions*. In: *STOC '88* [STO88], Seiten 113–131.

[CEv88] Chaum, David, Jan-Hendrik Evertse und Jeroen van de Graaf: *An improved protocol for demonstrating possession of discrete logarithms and some generalizations*. In: *Advances in Cryptology – EUROCRYPT '87*, Nummer 304 in *Lecture Notes in Computer Science*, Seiten 127–141, Berlin Heidelberg, Deutschland, 1988. Springer-Verlag.

[CGGM00] Canetti, Ran, Oded Goldreich, Shafi Goldwasser und Silvio Micali: *Resettable zero-knowledge (extended abstract)*. In: *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC 2000)*, Seiten 235–244, Portland, Oregon, USA, 2000. ACM Press.

[Cha87] Chaum, David: *Demonstrating that a public predicate can be satisfied without revealing any information about how*. In: *Advances in Cryptology – CRYPTO '86*, Nummer 263 in *Lecture Notes in Computer Science*, Seiten 195–199, Berlin Heidelberg, Deutschland, 1987. Springer-Verlag.

[CRY90] *Advances in Cryptology*, Nummer 435 in *Lecture Notes in Computer Science*, Berlin Heidelberg, Deutschland, 1990. Springer-Verlag.

[CRY92] *Advances in Cryptology*, Nummer 740 in *Lecture Notes in Computer Science*, Berlin Heidelberg, Deutschland, 1992. Springer-Verlag.

[CRY98] *Advances in Cryptology*, Nummer 1462 in *Lecture Notes in Computer Science*, Berlin Heidelberg, Deutschland, 1998. Springer-Verlag.

[Dam99] Damgård, Ivan: *Concurrent Zero-Knowledge is Easy in Practice*. Cryptology ePrint Archive, Report 1999/014, 1999. – URL http://eprint.iacr.org/1999/014.ps.gz – Zugriffsdatum 22.11.2002.

[DNS99] Dwork, Cynthia, Moni Naor und Amit Sahai: *Concurrent zero-knowledge*, 1999. Überarbeitete Version. Vorher erschienen in Proceedings of the 30th ACM Symposium on Theory of Computing (STOC '98), 1998, S. 409–418.

[DS98] Dwork, Cynthia und Amit Sahai: *Concurrent Zero-Knowledge: Reducing the Need for Timing Constraints*. In: *CRYPTO '98* [CRY98], Seiten 442–457.

[DS02] Dwork, Cynthia und Larry Stockmeyer: *2-round zero knowledge and proof auditors*. In: *STOC '2002* [STO02], Seiten 322–331.

[FFS87] Feige, Uriel, Amos Fiat und Adi Shamir: *Zero knowledge proofs of identity*. In: *STOC '87* [STO87], Seiten 210–217.

[FLS00] Feige, Uriel, Dror Lapidot und Adi Shamir: *Multiple Non-Interactive Zero Knowledge Proofs Under General Assumptions*. SIAM Journal on Computing, 29(1):1–28, 2000.

[FOC86] *Proceedings of 27th IEEE Symposium on Foundation of Computer Science*, 1986.

[For87] Fortnow, Lance: *The complexity of perfect zero-knowledge*. In: *STOC '87* [STO87], Seiten 204–209.

[FS90a] Feige, Uriel und Adi Shamir: *Witness indistinguishable and witness hiding protocols*. In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC '90)*, Seiten 416–426, Baltimore, Maryland, USA, 1990. ACM Press.

[FS90b] Feige, Uriel und Adi Shamir: *Zero Knowledge Proofs of Knowledge in Two Rounds*. In: *CRYPTO '89* [CRY90], Seiten 526–544.

[GK96a] Goldreich, Oded und Ariel Kahan: *How To Construct Constant-Round Zero-Knowledge Proof Systems for NP*. Journal of Cryptology: The journal of the International Association for Cryptologic Research, 9(3):167–189, Sommer 1996.

[GK96b] Goldreich, Oded und Hugo Krawczyk: *On the composition of Zero-Knowledge Proof Systems*. SIAM Journal on Computing, 25(1):169–192, 1996.

[GMR85] Goldwasser, Shafi, Silvio Micali und Charles Rackoff: *The Knowledge Complexity of Interactive Proof-Systems*. In: *STOC '85* [STO85], Seiten 291–304.

[GMW86] Goldreich, Oded, Silvio Micali und Avi Wigderson: *Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design*. In: *FOCS '86* [FOC86], Seiten 174–187.

[Gol99] Goldreich, Oded: *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*. Springer-Verlag, Berlin Heidelberg, Deutschland, 1999.

[Gol01] Goldreich, Oded: *Foundations of Cryptography*, Band 1: Basic Tools. Cambridge University Press, Cambrige, 2001.

[Gol02] Goldreich, Oded: *Zero-Knowledge twenty years after its invention*, Dezember 2002.

[GS86] Goldwasser, Shafi und Michael Sipser: *Private coins versus public coins in interactive proof systems*. In: *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC '86)*, Seiten 59–68, Berkeley, USA, Mai 1986. ACM Press.

[GSV98] Goldreich, Oded, Amit Sahai und Salil Vadhan: *Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge*. In: *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC '98)*, Seiten 399–408, Dallas, Texas, USA, 1998. ACM Press.

[HILL95] Håstad, Johan, Russell Impagliazzo, Leonid A. Levin und Michael Luby: *Construction of Pseudorandom Generator from any One-Way Function*, 1995. Überarbeitete Version. Vorversionen erschienen in Proceedings of the 21st ACM Symposium on Theory of Computing (STOC '89) und Proceedings of the 22nd ACM Symposium on Theory of Computing (STOC '90), ACM-Press.

[HM96] Halevi, Shai und Silvio Micali: *Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing*. In: *Advances in Cryptology – CRYPTO '96*, Nummer 1109 in *Lecture Notes in Computer Science*, Seiten 201–215, Berlin Heidelberg, Deutschland, 1996. Springer-Verlag.

[HT98] Hada, Satoshi und Toshiaki Tanaka: *On the Existence of 3-Round Zero-Knowledge Protocols*. In: *CRYPTO '98* [CRY98], Seite 408ff. – URL http://eprint.iacr.org/1999/009.ps.gz – Zugriffsdatum 23.12.2002.

[IY87] Impagliazzo, Russell und Moti Yung: *Direct Minimum-Knowledge Computations*. In: *Advances in Cryptology – CRYPTO '87*, Nummer 293

in *Lecture Notes in Computer Science*, Seiten 40–51, Berlin Heidelberg, Deutschland, 1987. Springer-Verlag.

[Kil92] Kilian, Joe: *A note on efficient zero-knowledge proofs and arguments*. In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC '92)*, Seiten 723–732, Victoria, British Columbia, Canada, 1992. ACM Press.

[KP00] Kilian, Joe und Erez Petrank: *Concurrent Zero-Knowledge in Poly-logarithmic Rounds*. Cryptology ePrint Archive, Report 2000/013, 2000. – URL http://eprint.iacr.org/2000/013.ps.gz – Zugriffsdatum 22.11.2002.

[KPR98] Kilian, Joe, Erez Petrank und Charles Rackoff: *Lower Bounds for Zero Knowledge on the Internet*. In: *Proceedings of 39th IEEE Symposium on Foundation of Computer Science (FOCS '98)*, Seiten 484–492, Palo Alto, California, USA, November 1998.

[Mat01] Mattern, Friedemann: *Sicherheit in verteilten Systemen*, 2001. – URL http://www.inf.ethz.ch/vs/edu/SS2000/DS/VertSys2000_22.pdf – Zugriffsdatum 01.11.2001.

[Mey92] Meyer, Bernd: *Bit-Commitment-Schemes und Zero-Knowledge*. Diplomarbeit, Universität des Saarlandes, Juli 1992.

[Mic94] Micali, Silvio: *CS proofs*. In: *Proceedings of 35th IEEE Symposium on Foundation of Computer Science (FOCS '94)*, Seiten 436–453, Santa Fe, New Mexico, USA, November 1994.

[MR01a] Micali, Silvio und Leonid Reyzin: *Min-round Resettable Zero-Knowledge in the Public-Key Model*. In: *Advances in Cryptology – EUROCRYPT 2001*, Nummer 2045 in *Lecture Notes in Computer Science*, Seiten 373–393, Berlin Heidelberg, Deutschland, 2001. Springer-Verlag.

[MR01b] Micali, Silvio und Leonid Reyzin: *Soundness in the Public-Key Model*. In: *Advances in Cryptology – CRYPTO 2001*, Nummer 2139 in *Lecture Notes in Computer Science*, Seiten 542–565, Berlin Heidelberg, Deutschland, 2001. Springer-Verlag.

[MvV01] Menezes, Alfred, Paul van Oorschot und Scott Vanstone: *Handbook of APPLIED CRYPTOGRAPHY*. CRC Press, 5. Auflage, August 2001. – URL http://www.cacr.math.uwaterloo.ca/hac/ – Zugriffsdatum 09.02.2003.

[Nao91] Naor, Moni: *Bit Commitment Using Pseudo-Randomness*. Journal of Cryptology: the journal of the International Association for Cryptologic Research, 4(2):151–158, 1991.

[NOVY92] Naor, Moni, Rafail Ostrovsky, Ramarathnam Venkatesan und Moti Yung: *Perfect Zero-Knowledge Arguments for NP Can Be Based on General Complexity Assumptions (Extended Abstract)*. In: *CRYPTO '92* [CRY92], Seiten 196–214.

[OVY91] Ostrovsky, Rafail, Ramarathnam Venkatesan und Moti Yung: *Fair Games Against an All-Powerful Adversary*. Erstmals veröffentlicht in DIMACS workshop, 1990. Journal Version in AMS DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Vol 13. (Jin-Yi Cai ed.) S. 155–169, 1993, 1991.

[Pfi98] Pfitzmann, Birgit: *Sicherheit*. Folien zur Stammvorlesung „Praktische Informatik“, UNIVERSITÄT DES SAARLANDES, Saarbrücken, Deutschland, 1998. – URL http://www-krypt.cs.uni-sb.de/WS9899/Vorlesung9899/Folien/pdf.4auf1.bunt/Pfit_98aFolien12bunt.4auf1.pdf – Zugriffsdatum 13.08.2002.

[Pfi99] Pfitzmann, Birgit: *Higher cryptographic protocols*. Vorlesungsmaterial zu „Höhere kryptographische Protokolle“, UNIVERSITÄT DES SAARLANDES, Saarbrücken, Deutschland, July 1999. – URL http://www-krypt.cs.uni-sb.de/SS00/VorlKryptProt/KryptProt990715.pdf – Zugriffsdatum 13.08.2002.

[Pfi00] Pfitzmann, Andreas: *Sicherheit in Rechnernetzen*. Vorlesungsmaterial zu „Datensicherheit und Kryptographie“, Technische Universität Dresden, Dresden, Deutschland, Oktober 2000. – URL http://dud.inf.tu-dresden.de/~pfitza/DSuKrypt.pdf – Zugriffsdatum 09.02.2003.

[QGB90] Quisquater, Jean-Jacques, Louis Guillou und Tom Berson: *How to explain zero-knowledge protocols to your children*. In: *CRYPTO '89* [CRY90], Seiten 628–631.

[Rey01] Reyzin, Leonid: *Zero-Knowledge with Public Keys*. Doktorarbeit, Massachusetts Institute of Technology, Juni 2001.

[RK99] Richardson, Ranso und Joe Kilian: *On the Concurrent Composition of Zero-Knowledge Proofs*. In: *Advances in Cryptology – EUROCRYPT '99*, Nummer 1592 in *Lecture Notes in Computer Science*, Seite 415ff., Berlin Heidelberg, Deutschland, 1999. Springer-Verlag.

[Sta02] Stammnitz, Peter: *Einführung in Kanalcodierungsverfahren - Mathematische Grundlagen*, 2002. – URL http://bs.hhi.de/~stammnit/Skript/Kap\protect\T1\textunderscore4/Kap\protect\T1\textunderscore4.pdf – Zugriffsdatum 21.01.2003.

[STO85] *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, Providence, Rhode Island, USA, 1985. ACM Press.

[STO87] *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, New York, New York, USA, 1987. ACM Press.

[STO88] *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, Chicago, Illinois, USA, 2–4 Mai 1988. ACM Press.

[STO02] *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, Montreal, Quebec, Canada, Mai 2002. ACM Press.

[TW87] Tompa, Martin und Heather Woll: *Random Self-Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information*. In: *Proceedings of 28th IEEE Symposium on Foundation of Computer Science (FOCS '87)*, Seiten 472–482, 1987.

[Wel90] Wellner, Ingrid: *Zur Theorie der Zero Knowledge Proofs*. Diplomarbeit, Universität des Saarlandes, März 1990.

[Yao82] Yao, Andrew Chi-Chih: *Theory and applications of trapdoor functions*. In: *Proceedings of 23th Annual Symposium on Foundations of Computer Science (FOCS '82)*, Seiten 80–91, November 1982.

Index