Zero Knowledge Prinzipien und besondere Sicherheitsgarantien

Christian Schausberger

B A K K A L A U R E A T S A R B E I T

Nr. 239-003-213-A

eingereicht am
Fachhochschul-Bakkalaureatsstudiengang

Computer- und Mediensicherheit

in Hagenberg

im Juni 2004

Diese Arbeit entstand im Rahmen des Gegenstands

Automatentheorie und Kryptologie 4

im

Wintersemester 2003/2004

Betreuer:

DI Dr. Jürgen Ecker

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 26. Mai 2004

Christian Schausberger

Inhaltsverzeichnis

Erklärung **iii**

Kurzfassung **v**

Abstract **vi**

1 Einführung **1**
1.1 Idee und Geschichte von Zero Knowledge 1
1.2 Die mystische Höhle von Ali Baba 2

2 Definitionen **5**
2.1 Turing Maschinen 5
2.2 Komplexitätsklassen 6
2.2.1 Die Klasse $\mathcal{P}$ 7
2.2.2 Die Klasse $\mathcal{NP}$ 7
2.2.3 Die Klasse $\mathcal{NP}$-vollständig 8
2.3 Bit Commitment Schemes 8
2.4 Zero Knowledge Beweise 9
2.4.1 Ein einfaches Zero Knowledge Protokoll 11
2.4.2 Perfect Zero Knowledge vs. Computational Zero Knowledge 13
2.4.3 Zero Knowledge Beweis Graphenisomorphismus 13
2.4.4 Das Fiat - Shamir Authentifikationsprotokoll 17

3 Besondere Sicherheitsanforderungen **19**
3.1 Serielle vs. parallele Ausführung 19
3.2 Concurrent Zero Knowledge 22
3.3 Resettable Zero Knowledge 23

4 Zusammenfassung und Ausblick **25**

Literaturverzeichnis **27**

Kurzfassung

Mit Hilfe von Zero Knowledge Beweisen ist es möglich ein skeptisches (möglicherweise nicht vertrauenswürdiges) Gegenüber von der Kenntnis eines Geheimnisses zu überzeugen. Das Spannende dabei ist, in welcher Art und Weise dies geschieht. Ein Zero Knowledge Beweis gibt nämlich keinerlei Informationen preis, die über „ich kenne das Geheimnis“ hinausgehen. Es ist also möglich ein Gegenüber von einem Geheimnis so zu überzeugen, dass es nichts lernt, außer, dass Sie das Geheimnis wirklich kennen. Anwendung findet dieses Prinzip beim Identitätsnachweis oder als Subprotokoll von größeren Protokollen (e-cash, e-voting).

Diese Arbeit erläutert nun das Prinzip von Zero Knowledge und die Zero Knowledge zu Grunde liegenden Theorien. Dabei wird versucht die mathematischen Prinzipien so einfach wie möglich zu erklären, sodass es auch einem Leser ohne fundierte mathematische Kenntnisse möglich ist, diese Arbeit zu verstehen.

In den späteren Kapiteln werden noch strengere Definitionen von Zero Knowledge diskutiert, die davon ausgehen, dass das Protokoll zur Beweisführung nicht ordnungsgemäß, und auf viele erdenkliche Weisen missbräuchlich, ausgeführt wird. *Concurrent Zero Knowledge* und *Resettable Zero Knowledge* halten auch ziemlich listigen Angreifern stand und verlieren dabei nicht ihre Zero Knowledge Eigenschaft. Da die Idee von Zero Knowledge an sich erst relativ jung ist, wird in diesem Bereich, im Hinblick auf Sicherheitsaspekte von Protokollen, momentan sehr viel geforscht.

Abstract

With Zero Knowledge it is possible to proof a statement of yours without revealing any further information beyond: this statement is valid. So a possible verifier learns nothing about your secret but he is assured that you are indeed in possession of it.

This work covers the principles of Zero Knowledge and their underlying mathematical theories. It is intended to do this in a way that it should be possible to understand this paper even if the reader does not have extended knowledge of mathematics.

After introducing zero knowledge, stronger definitions of the concept are shown. These are necessary, as in the real world verifiers (and maybe even provers) are not always acting like they should. *Concurrent Zero Knowledge* and *Resettable Zero Knowledge* address these problems and deal with them. The fact that zero knowledge is a relatively young idea explains the extended research done in the last twenty years until now. Zero knowledge protocols have been improved constantly and it may be possible that in the near future a whole new concept may be published.

Kapitel 1

Einführung

1.1 Idee und Geschichte von Zero Knowledge

Das Prinzip von Zero Knowledge lässt sich am besten anhand eines kurzen Beispiels erklären.

Angenommen, Alice möchte Bob davon überzeugen, dass sie faktorisieren kann (in einer effizienten Art und Weise), ohne ihm aber zu verraten, wie sie das anstellt. Bob glaubt ihr natürlich nicht, da er die derzeit gültige Auffassung vertritt, dass Faktorisieren schwierig ist und bei genügend großen Zahlen nicht in vernünftiger Zeit möglich ist. Bob will also wissen, wie Alice dies bewerkstelligt und verlangt von ihr, die Methode offenzulegen, sodass Bob sie prüfen kann. Da Alice die Methode aber lieber nicht preisgeben will (da sonst vielleicht Bob damit an die Öffentlichkeit tritt und behauptet er habe sie erfunden), fordert sie ihn auf ihm Zahlen zu geben, die sie dann für ihn faktorisiert und ihm das Ergebnis der Berechnung präsentiert.

Bob gibt ihr also Zahlen von denen er sich sicher ist, dass Alice sie nie im Leben in vernünftiger Zeit faktorisieren kann. Da Alice ihm aber als Ergebnis zu jeder Zahl die korrekte Faktorisierung präsentiert, die Bob jederzeit einfach und effizient überprüfen kann, wird Bob nach einer bestimmten Anzahl von Zahlen zufrieden sein und Alice glauben. Er hat jedoch nichts über die Methode wie man Zahlen faktorisiert gelernt.

Zero Knowledge Beweise gehen sogar noch einen Schritt weiter und erlauben nicht, dass Bob etwas erfährt, das er nicht auch selbst berechnen hätte können. Das ist in unserem Beispiel klar nicht der Fall, da er jetzt die Zerlegung der Zahlen in ihre Faktoren kennt.

Zum (schriftlich erwähnten) ersten Auftreten eines solchen Beweises kam es sogar schon im Jahre 1535. Der italienische Mathematiker Tartaglia behauptete einen Weg gefunden zu haben, wie man Gleichungen der Form

$$ax^3 + bx^2 + cx + d = 0$$

lösen kann. Er wollte aber aus irgendeinem Grund die Lösung nicht der Öffentlichkeit präsentieren. Ein skeptischer Zeitgenosse namens Fior glaubte Tartaglia nicht und forderte ihn auf, für ihn Gleichungen zu lösen. Er sandte Tartaglia 30 Gleichungen dritter Ordnung und wartete auf das Ergebnis. Tartaglia war es möglich, alle Gleichungen zu lösen. Er schickte Fior die Lösungen, ohne aber die einzelnen Schritte seiner Berechnungen beizulegen. Fior konnte leicht überprüfen, ob diese Lösungen korrekt waren und zu seinem Entsetzen waren alle richtig. Tartaglia hatte also Fior von seiner Behauptung überzeugt, ohne ihm sein Geheimnis zu verraten.

Die Idee eines echten Zero Knowledge Beweises wurde erstmals von Goldwasser, Micali und Racoff in [GMR85] im Jahr 1985 vorgestellt. In ihrem Artikel definierten sie interaktive Beweissysteme, die keine Information außer der Validität der Aussage weitergaben. Im selben Jahr erdachte auch Babai [BM88] seine Arthur Merlin Spiele, die im Wesentlichen zum gleichen Ergebnis kommen. Im Jahr darauf prägten dann Goldreich, Micali und Wigderson in [GMW86] den Begriff Zero Knowledge und zeigten wenig später in [GMW91], dass sich jedes Problem in $\mathcal{NP}$ in Zero Knowledge beweisen lässt.

Da unsere obigen Beispiele keineswegs Zero Knowledge sind (es werden immerhin Lösungen einer Gleichung verraten), wollen wir nun kurz festhalten, was man von einem echten Zero Knowledge Beweis erwartet. Wie soll man sich denn sicher sein, dass Alice bei einem Beweis Bob überzeugen kann, ohne wirklich mehr Information als nötig zu vermitteln? Dazu wollen wir erneut ein Beispiel[1] konstruieren, an dem klar wird, wie man dieses Ziel erreicht.

1.2 Die mystische Höhle von Ali Baba

Stellen wir uns eine Höhle vor wie in Abbildung 1.1 dargestellt. Diese Höhle verzweigt nach dem Eingang in zwei Richtungen. Ein Weg führt nach links, ein zweiter nach rechts. Wenn man die Höhle genau inspiziert, stellt man fest, dass beide Wege in einer Sackgasse enden. Die Höhle enthält allerdings einen Geheimgang, der die beiden Wege miteinander verbindet. Es gibt aber keine Möglichkeit den Geheimgang zu öffnen, außer man kennt das Passwort. Alice kennt dieses Passwort und will es Bob beweisen, ohne ihm aber das Passwort zu verraten. Bob kennt die Höhle, denn er hat als Kind immer dort gespielt und versucht das Passwort herauszufinden, damit er gemütlich vom linken in den rechten Gang wechseln kann. Er ist also überzeugt, dass man dies nur schafft, wenn man wirklich das Passwort kennt und er brennt darauf es zu erfahren. Alice führt folgendes Spielchen mit ihm auf:

[1] sinngemäß aus [QGAB89]

Abbildung 1.1: Ali Babas Höhle

Sie befiehlt Bob vor der Höhle zu warten. Dann geht sie in die Höhle und schleicht willkürlich entweder den linken oder rechten Gang entlang. Bob kann nun die Höhle betreten, aber nur bis zur Verzweigung. Er hat keine Ahnung welchen Weg Alice gewählt hat. Anschließend wirft er eine Münze. Bei Kopf muss Alice aus dem linken Gang herauskommen, bei Zahl aus dem Rechten. Bob wirft also die Münze und schreit das Ergebnis in die Höhle. Alice hört, aus welchem Gang sie herauskommen soll und hat nun folgende Möglichkeiten:

Sie ist den linken Gang entlanggegangen und Bob wirft Kopf. Damit kann sie einfach den Gang wieder zurückgehen, ohne von dem Passwort Gebrauch machen zu müssen. Oder Bob wirft Zahl und sie muss aus dem rechten Gang erscheinen. Da sie aber auf der linken Seite steht, flüstert sie das Passwort. Der Geheimgang öffnet sich und sie kann bequem auf die rechte Seite wechseln. So oder so, sie erscheint aus dem richtigen Gang.

Nun wird Bob bei der ersten Durchführung des Beweises nicht sonderlich beeindruckt sein. Schließlich besteht eine 50% Chance für Alice, von vornherein im richtigen Gang gewesen zu sein. Also wiederholen sie das Experiment solange, bis Bob zufrieden ist. Kennt Alice das Passwort, wird es ihr jedesmal möglich sein, Bob zu verblüffen. Kennt sie das Passwort nicht, werden ihre Chancen, jedesmal den richtigen Weg zu erraten, vernichtend klein. Die Chance, dass Alice 20 Runden übersteht, ohne das Passwort zu kennen, liegt bei $\frac{1}{2^{20}}$. Ist Bob mit dieser geringen Wahrscheinlichkeit noch nicht zufrieden, kann er das Spielchen solange treiben, bis er wirklich überzeugt ist. Bob hat bis zu diesem Zeitpunkt keinerlei Information über das Passwort erlangt, außer dass Alice es kennen muss. Sonst hätte sie es nicht geschafft, jedesmal im richtigen Gang zu erscheinen.

Ein Beweis ist nur dann Zero Knowledge, wenn es möglich ist, auch ohne

Kenntnis des Geheimnisses den Beweis zu erbringen. Wenn man das schafft, kann keine Information über das Geheimnis übermittelt werden, da es schließlich gar nicht an dem Beweis teilnimmt. Um das zu veranschaulichen, erweitern wir das vorige Beispiel.

Nehmen wir an, Bob hat den ganzen Vorgang auf Video gebannt, damit er sich jederzeit wieder über die Richtigkeit von Alice Aussage überzeugen kann. Aus irgendeinem Grund will Alice aber Bob eins auswischen und die Bänder des richtigen Beweises mit denen eines falschen Beweises austauschen. Sie geht also mit jemandem, der Bob ziemlich ähnlich sieht, erneut in die Höhle. Sie macht wieder exakt das Gleiche wie beim vorigen Beweis.

Der einzige Unterschied besteht nun darin, dass sie von ihrem Passwort keinen Gebrauch macht. Sie wird es also in 50% der Fälle nicht schaffen, aus dem richtigen Gang zu kommen. Führt Sie aber doppelt so viele Versuche durch, wird sie genausoviele gültige wie ungültige Versuche auf dem Video finden. Schneidet sie nun die ungültigen Versuche aus dem Video heraus (sie ist eine ziemlich findige Fälscherin, deshalb fällt keinem auf, dass an dem Band herumgeschnippselt wurde), hat sie eine Aufzeichnung, die genauso aussieht wie die von Bob. Tauscht sie also das Band gegen das von Bob aus, wird es ihm nicht möglich sein den Unterschied zu bemerken.

Man spricht von Zero Knowledge, wenn es nicht möglich ist, die Aufzeichnungen eines echten Beweises mit denen einer Fälschung zu unterscheiden.

Ein Zero Knowledge Beweis muss allerdings noch weitere Bedingungen erfüllen. Zum Einen muss es Alice möglich sein, wenn sie wirklich das Passwort kennt, Bob davon (fast immer) zu überzeugen. Man spricht dabei von der Vollständigkeitsbedingung (eng. *Completeness*). Zum Anderen darf es (fast) nicht möglich sein, dass Alice mit normalen Mitteln einen korrekten Beweis hinbekommt, wenn sie das Passwort nicht kennt. Diese Bedingung wird auch Korrektheits- oder Eindeutigkeitsbestimmung genannt (eng. *Soundness*). Diese beiden Bedingungen werden etwas später noch genauer beschrieben und betrachtet.

Kapitel 2

Definitionen

Damit das Beispiel aus 1.2 auch formal behandelt und eventuell später bewiesen werden kann, bedarf es noch den Begriff Zero Knowledge formal zu beschreiben. Dazu benötigen wir jedoch einiges an Definitionen und Prinzipien, von denen Zero Knowledge regen Gebrauch macht. Alle mathematischen Definitionen wurden, wenn nicht explizit anders erwähnt, sinngemäß von Oded Goldreichs sehr gutem Buch *Foundations of Cryptography* [Gol01] übernommen.

2.1 Turing Maschinen

Eine große Frage, die sich die Informationstheoretiker stellen, ist, was sie überhaupt alles berechnen können. Im Jahr 1937 erfand Alan Turing eine theoretische Maschine [Tur37], die, obwohl sie sehr simpel aufgebaut ist, jedes lösbare Problem lösen kann. Sie stellt dabei ein mathematisches Modell dar, das weitestgehend losgelöst von einer tatsächlichen Implementierung eines Computers ist. Eine Turing Maschine besteht im Wesentlichen aus Folgendem: einem unendlich langen Band auf das ein Schreib-/Lesekopf gesetzt wird. Das Band enthält dabei die Informationen in Form von Symbolen (z.B.: 0 oder 1) und je nachdem, was gelesen und geschrieben wird, befindet sich die Maschine in einem anderen Zustand. Aufgrund ihres Zustandes kann die Maschine einige Aktionen setzen. Sie kann das Band vor und zurückbewegen, auf das Band schreiben oder von dem Band lesen. Turing bewies, dass alle Probleme, die in der Mathematik lösbar sind, auch von seiner Maschine gelöst werden können. Dabei wird allerdings noch keine Aussage über die Effizienz der Berechnungen getätigt.

Im Wesentlichen kann man die hier vorgestellten Turing Maschinen auch mit unseren herkömmlichen PCs ersetzen, da sie auf dem Prinzip der Turing Maschinen beruhen und auf keinen Fall mehr berechnen können als diese. Der Korrektheit wegen wird aber auch im späteren Verlauf immer

nur von Turing Maschinen die Rede sein.

Für unsere weitere Verwendung benötigen wir noch drei Unterscheidungen einer Turing Maschine:

Deterministische Turing Maschine Eine deterministische Turing Maschine verhält sich nach einem streng vorgegebenen Algorithmus (ihrem Programm). Für einen Input x wird die Maschine immer den gleichen Output $M(x)$ liefern. Eine Berechnung bei gleichem Input führt also immer zum gleichen Ergebnis.

Probabilistische Turing Maschine Diese Turing Maschinen sind normale deterministische Turing Maschinen, die jedoch zusätzlich die Fähigkeit haben eine zufällige Entscheidung zu treffen. Sie können also Münzen werfen und je nach Ergebnis des Münzwurfes die Berechnung weiterführen. Die Maschine verfügt dabei über ein zusätzliches Zufallsband, von dem es die zufälligen Werte lesen kann. Es wurde gezeigt, dass solche Maschinen gewisse Probleme effizienter lösen können als deterministische Turing Maschinen, jedoch geht dieser Effizienzzuwachs zu Lasten der Korrektheit. Da die Maschine zu bestimmten Zeiten *raten* kann, ist nicht mehr gewährleistet, dass zu einem Input x auch das richtige Ergebnis berechnet wird. Es kann also nötig sein, eine solche Maschine mehrmals die Berechnung durchführen zu lassen, bis man das richtige Ergebnis erhält. Eine probabilistische Turing Maschine kann jedoch auf keinen Fall mehr Probleme lösen als eine deterministische Turing Maschine, es besteht lediglich die Möglichkeit, dass sie gewisse Berechnungen effizienter durchführen kann (aus [Gol01]).

Interaktive Turing Maschinen Eine interaktive Turing Maschine besteht aus 2 Turing Maschinen (deterministisch oder probabilistisch), die durch ein zusätzliches Band miteinander verknüpft sind. Das Ausgabeband der ersten Maschine ist das Eingabeband der zweiten, wobei dieses Band für die Eine nur beschreibbar und für die Andere nur lesbar ist. Es besteht also keine Möglichkeit, einmal geschriebene Informationen nachträglich noch zu ändern. Über dieses Band tauschen die Maschinen Nachrichten aus. Interaktive Turing Maschinen werden für die formale Beschreibung eines Zero Knowledge Beweises verwendet.

2.2 Komplexitätsklassen

Berechnungen werden je nach ihrem Komplexitätsgrad in verschiedene Klassen eingeteilt. Nachfolgende Klassen sind für diese Arbeit von Bedeutung:

2.2.1 Die Klasse $\mathcal{P}$

In dieser Klasse werden alle Probleme zusammengefasst, die mit deterministischen Turing Maschinen in polynomialer Zeit berechnet werden können.

Einfach ausgedrückt sind dies die Probleme, die wir mit unseren Taschenrechnern und Köpfen jeden Tag lösen. Mit polynomialer Zeit meint man, dass die Dauer der Berechnung polynomial mit der Größe der Eingangsvariablen steigt. Eine solche Berechnung wäre beispielsweise das Multiplizieren. Lassen wir unseren Taschenrechner 5 mal 5 berechnen, werden wir das Ergebnis sehr schnell auf unserem Display angezeigt bekommen. Wollen wir hingegen das Ergebnis von 98764536 mal 36475936 wissen, müssen wir nur unmerklich länger auf das Ergebnis warten. Solche Probleme sind also im Wesentlichen immer effizient lösbar und deshalb für die Kryptographie und im Weiteren auch für Zero Knowledge Beweise, die sich die Grundlagen der Kryptographie zu Nutze machen, höchst uninteressant.

Definition Klasse $\mathcal{P}$: Eine Sprache L[1] wird in (deterministischer) polynomialer Zeit wahrgenommen, wenn eine deterministische Turing Maschine M und ein Polynom $p()$ existiert, sodass

- bei Input x, die Maschine M nach höchstens $p(|x|)$ Schritten hält, und
- $M(x) = 1$ nur dann, wenn $x \in L$ gilt.

2.2.2 Die Klasse $\mathcal{NP}$

In dieser Klasse werden alle Probleme zusammengefasst, deren Lösungen mit einer deterministischen Turingmaschine in polynomialer Zeit verifiziert werden können.

Die Lösungen für solche Probleme können also nicht unbedingt effizient berechnet werden, da die Zeit, die zur Berechnung notwendig ist, schlimmstenfalls exponential mit der Größe der Eingangsvariable steigt. Ein gutes Beispiel für ein solches Problem ist das Faktorisieren. Für dieses Problem steht uns derzeit kein vernünftiger Algorithmus zur Verfügung, obwohl nicht bewiesen ist, dass es einen solchen nicht gibt. Wir können im Wesentlichen nur durch alle möglichen Zahlen dividieren, um die richtige Faktorzerlegung zu berechnen. Das bereitet uns bei kleinen Zahlen noch keine großen Schwierigkeiten, aber mit zunehmender Größe der Zahl stoßen wir schnell auf unsere Grenzen. Deshalb beruhen viele der kryptographischen Verfahren auf solchen Problemen. Dem RSA Verfahren [RSA78] liegt die

[1] Eine Sprache L besteht aus einer Menge von Wörtern endlicher Länge über einem endlichen Alphabet.

(angenommene) Schwierigkeit des Faktorisierens zu Grunde, das Diffie Hellman Problem [DH76] beruht auf der Schwierigkeit, diskrete Logarithmen zu berechnen. Da allerdings nicht bewiesen ist, dass diese Probleme wirklich schwer sind (d.h. nicht in der Klasse $\mathcal{P}$ liegen), wären diese Verfahren null und nichtig, wenn ein effizienter Algorithmus dafür gefunden wird.

Auf der anderen Seite fällt es uns nicht schwer zu überprüfen, ob eine vorliegende Faktorisierung einer Zahl korrekt ist. Dies lässt sich in polynomialer Zeit bewerkstelligen.

Definition Klasse $\mathcal{NP}$: Eine Sprache L liegt in $\mathcal{NP}$, wenn eine Relation R_L existiert, die in polynomialer Zeit überprüft werden kann, und $x \in L$ nur dann gilt, wenn es ein y gibt, sodass $(x, y) \in R_L$. Ein solches y wird als *Zeuge* für die Mitgliedschaft von x in L bezeichnet.

2.2.3 Die Klasse $\mathcal{NP}$-vollständig

Diese Klasse stellt eine Teilklasse von $\mathcal{NP}$ dar. Jedes Problem, das $\mathcal{NP}$-vollständig ist, liegt also auch in $\mathcal{NP}$. Das Besondere an dieser Unterklasse ist jedoch, dass sich jedes Problem in $\mathcal{NP}$ auf die $\mathcal{NP}$-vollständigen Probleme reduzieren lässt. Eine Sprache L ist in polynomialer Zeit auf eine Sprache L' reduzierbar, wenn es eine, in polynomialer Zeit berechenbare Funktion f gibt, sodass $x \in L$ nur dann gilt, wenn $f(x) \in L'$.

Da nicht bewiesen ist, dass Probleme in $\mathcal{NP}$ (und somit auch in $\mathcal{NP}$-vollständig) nicht in $\mathcal{P}$ liegen, brennen die Wissenschaftler natürlich darauf, eines der $\mathcal{NP}$-vollständigen Probleme zu lösen, da dadurch automatisch auch alle anderen $\mathcal{NP}$ Probleme lösbar wären, also $\mathcal{P} = \mathcal{NP}$. Solche Probleme sind beispielsweise das Travelling Salesman Problem, das Knapsack Problem oder das Problem der Graphenisomorphie, das in 2.4.3 noch genauer beschrieben wird. Eine Beschreibung dieser Probleme bietet [Woe03].

Definition Klasse $\mathcal{NP}$-vollständig Eine Sprache L liegt in $\mathcal{NP}$-vollständig wenn sie in $\mathcal{NP}$ liegt und jede Sprache in $\mathcal{NP}$ in polynomialer Zeit auf diese reduzierbar ist.

2.3 Bit Commitment Schemes

Bit Commitment Schemes bieten die Möglichkeit, dass sich ein Sender S auf einen Wert festlegen kann (eng. *commit*), ohne dass der Empfänger R diesen erfährt. Nachdem er sich festgelegt hat, ist es ihm nicht mehr möglich sich anders zu entscheiden, also den Wert zu verändern. R hat keine Möglichkeit den festgelegten Wert zu erfahren, ohne dass S es will. Bit Commitment Schemes stellen ein Analogon zu versiegelten Umschlägen der physikalischen Welt dar. Eine Nachricht wird im Umschlag versiegelt und kann nachher

nicht mehr verändert werden. Der Andere kann die Nachricht aber nicht lesen, solange der Umschlag verschlossen ist.

Bit Commitment Schemes bilden die Grundlage für die meisten Zero Knowledge Protokolle. In unserem Beispiel aus Kapitel 1.2 legt sich Alice ebenfalls auf einen Wert fest. Sie wählt einen von 2 Gängen und da Bob ihr bis zur Abzweigung folgt, kann sie sich nicht mehr umentscheiden.

Die Möglichkeit sich auf ein einzelnes Bit in dieser Art und Weise festzulegen, wurde erstmals in [BCC88] vorgeschlagen. In dieser Arbeit wurde auch der Begriff des Bit Commitment Scheme geprägt.

Ein Bit Commitment Scheme läuft üblicherweise in 2 Phasen ab. In der *Festlegungsphase* (eng. *commit stage*) legt sich S auf Bit $\sigma \in \{0, 1\}$ in einer Art und Weise fest, dass er es nachher nicht mehr verändern kann und R keine Möglichkeit hat, zu erfahren auf welches Bit S sich festgelegt hat. Die dabei entstehende Nachricht wird *Commitment* genannt. S übermittelt R das Commitment. In der *Öffnungsphase* (eng. *reveal stage*) wird R das Bit σ bekanntgegeben. Durch das Bit Commitment Scheme wird sichergestellt, dass S das richtige Bit bekanntgeben muss. Würde S $1 - \sigma$ übermitteln, würde R diesen Schwindel sofort bemerken und z. B. das Protokoll abbrechen.

Ein Bit Commitment Scheme hat zumindest folgende 2 Eigenschaften:

- **Eindeutigkeit** (eng. *binding property*): Dem Sender ist es nicht möglich, den Wert nach der Festlegungsphase zu verändern.
- **Geheimhaltung** (eng. *hiding property*): Der Empfänger kann ein Commitment von 1 nicht von einem Commitment von 0 unterscheiden.

Um diese Eigenschaften zu erreichen bedienen sich die Commitment Schemes kräftig der Methoden der Kryptographie. Das bedeutet aber, dass sie nur unter denselben Annahmen funktionieren, wie z. B. dass es schwierig ist zu faktorisieren. Falls bewiesen wird, dass Probleme in $\mathcal{NP}$ doch in polynomialer Zeit lösbar sind, also $\mathcal{P} = \mathcal{NP}$, sind auch Commitment Schemes in der derzeitigen Form nicht mehr möglich.

2.4 Zero Knowledge Beweise

Wir sollten jetzt die nötigen Grundlagen haben, um eine genauere Formulierung eines Zero Knowledge Beweises in Angriff zu nehmen.

Wenn ein Beweisführer, im Weiteren P (für *Prover*) genannt, eine Aussage (z. B. ich besitze den Private Key) per Zero Knowledge beweisen will, erlangt der Überprüfer, im Weiteren V (für *Verifier*) genannt, dabei keine

zusätzliche Information, außer dass die Aussage richtig ist. Was ist aber mit zusätzlicher Information gemeint? Unter zusätzlicher Information, oder auch Wissen, verstehen wir alles, was ein V nach einer Interaktion mit P besser (oder effizienter) berechnen kann, als vor der Interaktion. Erfährt V von P zum Beispiel, dass $1 + 1 = 2$, dann können wir getrost behaupten, dass V nichts dabei gelernt hat, da er das Ergebnis auch selbst hätte berechnen können ohne P zu fragen. Verrät P jedoch die Faktorzerlegung einer großen Zahl n, sind die Möglichkeiten von V, Berechnungen bezüglich n durchzuführen, eindeutig gestiegen, da er jetzt die Zerlegung von n kennt. Er kann jetzt z. B. die verschlüsselten Nachrichten von P entschlüsseln, wenn es sich bei n um den Modul im RSA Public Key von P handelt. Ein solcher Beweis wäre klar nicht das Ziel von Zero Knowledge.

Das Geheimis, das es zu beweisen gilt, muss aber irgendwie in die Nachrichten des Beweises einfließen, ansonsten könnte jeder einen richtigen Beweis hinbekommen (*Soundness*). Wenn also das Geheimnis in irgendeiner Form in den Nachrichten enthalten ist, wie kann es dann Zero Knowledge sein?

Die Antwort darauf liegt in den Prinzipien der Kryptographie. Unter der Annahme, dass es Probleme gibt, die schwierig zu lösen sind (also Probleme in $\mathcal{NP}$), stört es nicht, dass Informationen in der Nachricht enthalten sind, solange es für V nicht attraktiv ist, zu versuchen, sie aus der Nachricht zu berechnen. Mit nicht attraktiv meinen wir, dass es sich für V nicht lohnen würde, da er es nicht in polynomialer Zeit schaffen kann, eine Lösung zu erhalten.

Wir bezeichnen also solche Beweise als Zero Knowledge, bei denen die Möglichkeiten von V Berechnungen durchzuführen, nach Interaktion mit P nicht steigen. Diese Eigenschaft wird nachgewiesen, wenn ein Simulator (der ohne das Geheimnis auskommen muss) für den Beweis präsentiert werden kann.

Definition Simulator: Wir wollen einen Simulator wie folgt definieren.

Ein Simulator ersetzt sowohl P als auch V. Er vereint beide Aufgaben in einer Maschine. Außerdem hat er Zugriff auf das Zufallsband der Turingmaschine, die V darstellt. Dadurch kennt der Simulator die Nachrichten, die er von V geschickt bekommt bereits im Voraus. Dadurch ist es ihm möglich, die Nachrichten von P so nachzuahmen, dass er einen Beweis auch dann schaffen kann, wenn er das Geheimnis nicht kennt. Wird die Kommunikation von solch einem Simulator aufgezeichnet, kann sie zu einem späteren Zeitpunkt nicht, oder zumindest nicht in polynomialer Zeit, von der Kommunikation eines echten Beweises unterschieden werden. Die mathematisch korrekte Beschreibung eines Simulators wird hier nicht aufgeführt, da sie für das Verständnis von Zero Knowledge nicht nötig ist. Eine Beschreibung, die selbst höchsten Ansprüchen genügen dürfte, findet sich in [Gol01].

Folgendes Beispiel (aus [Eck04]) soll dies verdeutlichen. Dieses Beispiel beruht auf der Schwierigkeit, diskrete Logarithmen zu berechnen und wird auch noch für spätere Diskussionen als Grundlage dienen.

2.4.1 Ein einfaches Zero Knowledge Protokoll

Sei G eine Gruppe großer Ordnung, b ein fixes Element großer Ordnung dieser Gruppe und y ein beliebiges Element dieser Gruppe. P will V davon überzeugen, dass er den Logarithmus von y zur Basis b kennt, also jenes x, so dass $y = b^x$.

1. P wählt zufällig eine Zahl e und schickt an V das Element $B = b^e$.
2. V wirft eine Münze.

   Bei Kopf muss P e bekanntgeben und V prüft, ob $B = b^e$.

   Bei Zahl muss P $x+e$ bekanntgeben und V prüft, ob $y \cdot B = b^{x+e}$.
3. Schritte (1) und (2) werden solange wiederholt, bis V überzeugt ist.

In diesem Beispiel sind G, b und y öffentlich bekannt, x ist in diesem Fall das Geheimnis, dessen Kenntnis von P bewiesen, aber nicht preisgegeben werden soll. Analysieren wir die einzelnen Schritte, um zu zeigen, dass dieser Beweis Zero Knowledge ist.

- Im ersten Schritt legt sich P auf einen zufälligen Wert e fest und schickt b^e an V. Da b öffentlich bekannt ist und e ein zufälliger Wert kann V aus dieser Nachricht nichts lernen.
- Wird im zweiten Schritt nun Kopf geworfen, gibt P e bekannt und V lernt wieder nichts. Der interessantere Fall ist, wenn Zahl geworfen wird. In diesem Fall wird $x + e$ übertragen. Damit V aus dieser Nachricht x berechnen kann, müsste er den Wert von e kennen. e wurde aber in der Form b^e übertragen. Damit steht V vor einem diskreten Logarithmus Problem. Da derzeit kein Algorithmus bekannt ist, der dieses Problem in polynomialer Zeit lösen kann, kann V e nicht in vernünftiger Zeit berechnen und so auch x nicht herausfinden. V lernt also nichts über x, außer dass P x kennen muss.

Als nächstes stellen wir uns die Frage, ob die beiden zusätzlichen Bedingungen *Vollständigkeit* und *Korrektheit* gewährleistet sind.

- **Vollständigkeit** Wenn P, wie behauptet, im Besitz von x ist, wird es ihm jedes Mal gelingen, V eine Nachricht zu schicken, die seinen Überprüfungen standhält. Nach genügend Runden wird V zufrieden sein, wenn seine Überprüfungen stimmen. Wenn sich also beide an das Protokoll halten, wird P V in jedem Fall überzeugen können und die Bedingung ist erfüllt.

- **Korrektheit** Wenn P x nicht kennt und er trotzdem versuchen will den Beweis anzutreten, können folgende zwei Fälle eintreten:

  (a) P hofft, dass V Kopf wirft. Also wählt P zufällig ein e und schickt dieses an V. Wirft V Kopf so hat P eine Runde überstanden. Wirft V aber Zahl, müsste P $x+e$ schicken. Da er aber x nicht kennt, kann er das nicht und ist als Betrüger entlarvt.

  (b) P hofft, dass Zahl geworfen wird. Er wählt wieder ein e, schickt aber $B = b^e \cdot y^{-1}$. Wird tatsächlich Zahl geworfen, berechnet V, ob $y \cdot B = b^{x+e}$ und wird zufrieden sein. Kommt jedoch Kopf wird V die Schwindelei erneut auffliegen lassen. Mit jeder neuen Runde sinken also die Chancen für P eine weitere Runde zu überstehen um 50%. Nach n Runden liegt die Wahrscheinlichkeit, dass P Erfolg hat, nur mehr bei $\frac{1}{2^n}$. V ist also vor falschen Beweisen gefeit, wenn er das Protokoll mehrere Runden lang durchführt und somit ist auch die zweite Bedingung erfüllt.

Können wir nun für dieses Beispiel einen **Simulator** bauen? Ein Simulator ist nicht im Besitz von x und kann somit den Beweis nicht überstehen. Ein Simulator muss dies allerdings nicht können, wir verlangen lediglich von ihm, dass er es schafft einen Output zu liefern, der nicht von einem ehrlichen P zu unterscheiden ist. Dazu könnte man wie in Kapitel 1.2 das Protokoll einfach doppelt so oft laufen lassen. Der Simulator würde dann in 50% der Fälle eine korrekte Nachricht liefern können. Löscht man dann die ungültigen Nachrichten aus der Aufzeichnung, ist es nicht mehr möglich den Output des Simulators und den von P zu unterscheiden.

Eine zweite Möglichkeit ist es, dem Simulator eine tolle Eigenschaft zu gewähren. Er darf in die Zukunft schauen. Auf diese Weise weiß der Simulator immer welches Bit V als nächstes wählen wird und kann so immer die richtige Nachricht an V schicken. Eine Aufzeichnung dieses Versuchs wird auch nicht von einem gültigen Beweis unterscheidbar sein.

Der Inhalt der Bänder muss dabei keineswegs identisch sein. Wir sprechen dann von nicht unterscheidbar, wenn es keinen Algorithmus gibt, der erkennen kann, welche der beiden Aufzeichnungen die der Simulation ist.

Zusammenfassung: Ein Zero Knowledge Beweis muss 3 Anforderungen erfüllen.

1. *Zero Knowledge Eigenschaft* Nach Interaktion von V mit P erfährt V nichts, was er nicht auch selbst berechnen hätte können. Seine Fähigkeiten ein bestimmtes Problem zu lösen dürfen also nicht steigen. Die Zero Knowledge Eigenschaft ist immer dann gegeben, wenn es möglich ist, eine Simulation für die Kommunikation von P und V zu präsentieren.

2. *Vollständigkeit* Wenn P und V ehrlich an dem Protokoll teilnehmen, muss es P mit hoher Wahrscheinlichkeit möglich sein V von seiner Aussage zu überzeugen.

3. *Korrektheit* Wenn P nicht ehrlich ist (er also V von einer falschen Aussage überzeugen will), muss V dies erkennen können und mit hoher Wahrscheinlichkeit ablehnen, ganz gleich in welcher Art und Weise P zu tricksen versucht.

Außerdem beruht das ganze Zero Knowledge Konzept (wie auch der Großteil der Kryptographie) darauf, dass es Probleme gibt, die nicht in vertretbarer Zeit zu lösen sind und für die es einfach ist, eine Lösung für ein solches Problem zu überprüfen.

2.4.2 Perfect Zero Knowledge vs. Computational Zero Knowledge

Zero Knowledge kann noch in mehrere Arten unterteilt werden. Diese unterscheiden sich durch die Strenge der Definition und bieten somit auch unterschiedliche Sicherheiten.

- **Perfect Zero Knowledge** Darunter versteht man einen Beweis, bei dem es in der Tat nicht möglich ist, eine Aufzeichnung eines echten Beweises von der Aufzeichnung eines Simulators zu unterscheiden, auch nicht wenn man unendliche Rechenpower besitzt (also nicht an polynomiale Zeit gebunden ist). Das vorherige Beispiel bietet einen solchen Perfect Zero Knowledge Beweis. Eine solche Aufzeichnung stellt normalerweise die Kommunikationsbänder der beiden Turing Maschinen dar. Die Bänder beider Maschinen zusammen ermöglichen ein Nachvollziehen der Kommunikation.

- **Computational Zero Knowledge** In der Praxis ist es allerdings oft nicht nötig, einen perfekten Simulator bauen zu können. In diesen Fällen gibt man sich damit zufrieden, wenn eine Simulation nicht mit vertretbarem Aufwand (also in polynomialer Zeit) von einem echten Beweis unterscheidbar ist. Einem Angreifer mit unbegrenzten Möglichkeiten wird es möglich sein, die Simulation zu erkennen. Da es solche Angreifer (zumindest derzeit) nicht gibt, verliert man durch die weniger strenge Definition nichts an Sicherheit.

2.4.3 Zero Knowledge Beweis Graphenisomorphismus

Ausflug Graphentheorie Einen Graphen G beschreiben wir als die Menge seiner Knoten und Kanten, wobei $G = (V, E)$. V stellt dabei die Menge der Knoten des Graphen dar ($V = \{1, ..., 7\}$) und E beschreibt die Kanten

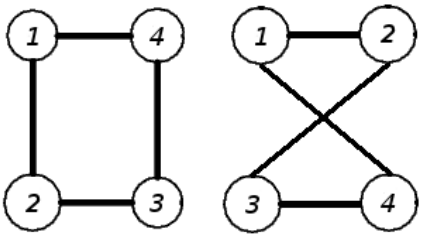

Abbildung 2.1: $V = \{1, 2, 3, 4\}$, $E = \{(1, 2), (2, 3), (3, 4), (4, 1)\}$

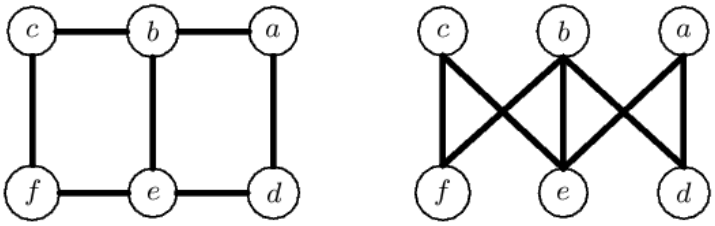

Abbildung 2.2: Sind diese Graphen isomorph?

zu den Ecken ($E = \{(1, 2), (1, 4), (3, 5)\}$). Für eine genaue Einführung in die Graphentheorie siehe [Vol91].

Graphen werden auch gerne aufgezeichnet, jedoch ist die bildliche Darstellung nicht eindeutig, da zwei unterschiedlich ausschauende Graphen die gleiche formale Beschreibung haben können. Abbildung 2.1 soll das verdeutlichen.

Wenn zwei Graphen die gleiche Anzahl an Knoten und Kanten haben und man es schafft, die Knoten des einen Graphen so umzubenennen, dass die formale Beschreibung beider Graphen gleich ist, heißen diese isomorph. Das *Graphenisomorphieproblem* (GIP) ist, bei zwei gegebenen Graphen zu entscheiden, ob diese isomorph sind. Da bis jetzt kein effizienter Algorithmus bekannt ist, der dies entscheiden könnte, bleibt im wesentlichen nur die Brute Force Methode übrig. Man muss also alle Möglichkeiten durchprobieren, was für kleine Graphen noch recht praktikabel sein mag, aber bei größeren Graphen recht schnell sehr aufwändig wird. Tatsächlich liegt das GIP in $\mathcal{NP}$-vollständig.

Zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ sind dann isomorph, wenn es eine bijektive Abbildung $\phi : V_1 \to V_2$ gibt, so dass für alle $u, v \in V_1$

$$\text{aus } (u, v) \in E_1 \text{ folgt, dass } (\phi(u), \phi(v)) \in E_2 \text{ gilt.}$$

Folgendes Beispiel soll dieses Problem erläutern.

Sind die beiden Graphen G_1 (links) und G_2 (rechts) aus Abbildung 2.2 isomorph? Wir sehen uns schrittweise die Lösung an.

- Zuerst sehen wir uns die formale Beschreibung der beiden Graphen an.

  $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$

  $V_1 = V_2 = \{a, b, c, d, e, f\}$

  $E_1 = \{(a, b), (a, d), (b, c), (b, e), (c, f), (d, e), (e, f)\}$

  $E_2 = \{(a, e), (a, d), (b, d), (b, e), (b, f), (c, e), (c, f)\}$

- Wir versuchen den Isomorphismus zu finden, indem wir die Knoten b und e vertauschen, wobei der Knoten b zu e wird und umgekehrt. Wir verändern sie im Graphen G_1 und erhalten folgendes Ergebnis:

  $G' = (V', E')$

  $V' = (a, b, c, d, e, f)$

  $E' = \{(a, e), (a, d), (b, d), (b, e), (b, f), (c, e), (c, f)\}$

- Wenn wir jetzt G' mit G_2 vergleichen, werden wir festellen, dass $G' = G_2$. Wir haben also einen Isomorphismus für dieses Beispiel gefunden (es gibt noch drei weitere, falls jemand Zeit zum Suchen hat).

In [Gol01] wird ein perfekter Zero Knowledge Beweis für Graphenisomorphie dargestellt, den wir hier ebenfalls beleuchten möchten. Wir bezeichnen mit $\mathcal{GI}$ die Menge aller paarweise isomorphen Graphen. Der Beweis ist deshalb besonders interessant, weil das Problem der Graphenisomorphie ein $\mathcal{NP}$-vollständiges Problem darstellt. Ein Beweis für ein Problem dieser Art hat zur Folge, dass auch alle anderen $\mathcal{NP}$ Probleme Zero Knowledge Beweise haben müssen, da sich ja jedes $\mathcal{NP}$ Problem auf ein $\mathcal{NP}$-vollständiges reduzieren lässt. Gezeigt wurde dies schon in [GMW86] und in einer neueren Auflage in [GMW91].

In diesem Beweis will P V davon überzeugen, dass er einen Isomorphismus von 2 Graphen kennt. Er könnte natürlich einfach den Isomorphismus bekanntgeben, aber das wäre nicht Zero Knowledge und somit nicht in unserem Sinne. Stattdessen passiert folgendes:

- Wir benötigen 2 Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ und einen Isomorphismus ϕ zwischen diesen beiden Graphen.

- G_1 und G_2 sind P und V bekannt, ϕ ist nur P bekannt.

1. P erzeugt im ersten Schritt eine zufällige Kopie von G_2, die isomorph zu G_2 sein muss. Dazu erstellt er eine zufällige Permutation π der Knoten V_2 und konstruiert einen Graphen mit den Knoten V_2 und Ecken F, wobei sich F aus der Permutation ergibt.

$$F = \{(\pi(u), \pi(v)) : (u, v) \in E_2\}$$

P sendet dann $H = (V_2, F)$ an V.

Wenn die Graphen G_1 und G_2 isomorph sind, dann ist auch H isomorph zu diesen beiden Graphen, da H isomorph zu G_2 ist. Wenn allerdings G_1 und G_2 nicht isomorph zueinander sind, kann es keinen Graphen geben, der zu beiden isomorph ist.

2. V empfängt H und fordert P auf, ihm einen Isomorphismus zwischen H und einem der beiden Graphen G_1 und G_2 zu zeigen. Dazu wählt er zufällig $\sigma \in \{1, 2\}$ und schickt dieses an P zurück.

3. P soll nun einen Isomorphismus zwischen G_σ und H herzeigen. Wenn das empfangene σ den Wert 2 hat, schickt P einfach π zurück. Sonst (also $\sigma \neq 2$) wird $\pi \circ \phi$ zurückgesandt. $\pi \circ \phi$ stellt dabei einen Isomorphismus von G_1 und H dar.

4. V prüft nun, ob die erhaltene Nachricht einen Isomorphismus zwischen G_σ und H darstellt. Ist dies der Fall, akzeptiert V, ansonsten lehnt er ab.

Für das obige Protokoll weden nun folgende Behauptungen aufgestellt, die gleich bewiesen werden.

Behauptung: Das obige Protokoll stellt einen perfekten Zero Knowledge Beweis dar. Weiters genügt es folgenden Ansprüchen:

1. Wenn G_1 und G_2 isomorph sind (also $(G_1, G_2) \in \mathcal{GI}$), wird V den Beweis (wenn er nach dem Protokoll abläuft) jedesmal akzeptieren.

2. Wenn G_1 und G_2 nicht isomorph sind (also $(G_1, G_2) \notin \mathcal{GI}$), wird V mit einer Wahrscheinlichkeit von mindestens $\frac{1}{2}$ ablehnen.

3. Das Protokoll ist Zero Knowledge, da für diesen Beweis ein Simulator präsentiert werden kann, dessen Output nicht von einem echten Beweis unterschieden werden kann.

Beweis: Wenn G_1 und G_2 isomorph sind, muss auch der erzeugte Graph H isomorph zu diesen Graphen sein. Deshalb wird es P immer gelingen (wenn er ϕ kennt), V einen gültigen Beweis zu liefern und V wird jedesmal akzeptieren, vorausgesetzt beide halten sich an das Protokoll. (Behauptung 1)

Sind die Graphen G_1 und G_2 nicht isomorph zueinander, kann es keinen Graphen H geben, der zu beiden isomorph ist. Daraus folgt, dass egal in welcher Art und Weise P versucht, H zu konstruieren, wird es ein $\sigma \in \{1, 2\}$

geben, sodass G_σ und H nicht isomorph sind. P kann jedoch am Anfang raten, welchen Isomorphismus V von ihm sehen will. Wenn er Glück hat und richtig geraten hat, wird er den Isomorphismus präsentieren können. Hat er jedoch falsch getippt muss er sich leider als Schwindler enttarnen lassen. V wird also, wenn er dem Protokoll treu bleibt, mit einer Wahrscheinlichkeit von mindestens $\frac{1}{2}$ ablehnen. (Behauptung 2).

Bleibt noch zu zeigen, dass dieses Protokoll auch wirklich Zero Knowledge ist. Ein Simulator, der die Kommunikation zwischen P und V nachbilden kann, den Isomorphismus ϕ jedoch nicht kennt, hat folgende Möglichkeiten. Er weiß welches σ als nächstes gewählt wird, da er wie zuvor wieder in die Zukunft blicken kann. Er wählt also G_σ und erzeugt für diesen Graphen einen Isomorphismus π. Aus diesem erzeugt er einen Graphen H. Wenn er dann den Beweis antreten soll (also den Isomorphismus zwischen G_σ und H herzeigen), schickt er einfach π und sein Gegenüber wird zufrieden sein. Die Kommunikation der Simulation wird für jeden genauso aussehen wie ein echter Beweis. (Behauptung 3).

Die Beschreibung des Simulators folgt der Definition auf Seite 10 (und ist daher wesentlich einfacher) und weicht wesentlich vom Original in [Gol01] ab.

2.4.4 Das Fiat - Shamir Authentifikationsprotokoll

Das Zero Knowledge Prinzip hat mittlerweile eine Menge von Anwendungen gefunden. In diesem Kapitel soll ein Protokoll genauer vorgestellt werden. Die Zero Knowledge Protokolle eignen sich besonders gut zum Nachweis der Identität. Sie finden auch bei Signaturen [Cha90] oder als Unterprotokolle beim elektronischen Geldverkehr [OO89] Anwendung. Zwei der bekannteren Zero Knowledge Protokolle wurden 1988 von Guillou und Quisquater in [GQ89] und von Fiat und Shamir im Jahr 1986 in [FS86] vorgestellt. Letzteres werden wir nun etwas genauer betrachten.

Das Protokoll wurde im Hinblick auf eine effiziente Ausführung auf Smartcards entwickelt und umfasst deshalb, neben dem eigentlichen Algorithmus, auch eine Menge anderer Anforderungen, wie z. B. ein Trusted Center, welches die Smart Cards ausstellt. Für das Verständnis des Protokolls sind diese jedoch nicht wichtig. Außerdem ist es auch möglich, das Protokoll ohne Trusted Center mit Hilfe einer PKI auszuführen.

Setup Sei I ein String, der die Identität des Users darstellt, n das Produkt von zwei Primzahlen p und q, und f eine Zufallsfunktion, die I in den Bereich $[0, n) = \{0, 1, ..., n-1\}$ abbildet.

1. P berechnet die Werte $v_j = f(I, j)$ für kleine j.
2. P wählt k unterschiedliche Werte für j, für die v_j ein quadratischer Rest $\pmod n$ ist. Berechne s_j als $\sqrt{v_j} \pmod n$.

Die Werte s_j sind geheim zu halten, I, f, n und die Werte für j werden veröffentlicht. Der Einfachheit halber wird im weiteren Protokoll angenommen, dass j die Werte $1, ..., k$ annimmt.

Protokoll

1. P sendet I an V.
2. V berechnet $v_j = f(I, j)$ für $j = 1, ..., k$.
   Schritte (3) bis (6) werden für $i = 1, ..., t$ wiederholt.
3. P wählt zufällig $r_i \in [0, n)$ und sendet $x_i = {r_i}^2 \pmod n$ an V.
4. V sendet einen binären Vektor $(e_{i1}, ..., e_{ik})$ an P.
5. P sendet schließlich folgendes an V:
$$y_i = r_i \prod_{j=1}^{k} s_j^{e_{ij}} \pmod n$$
6. V prüft, ob:
$$x_i = y_i^2 \prod_{j=1}^{k} v_j^{e_{ij}} \pmod n$$

Vollständigkeit: Wenn beide das Protokoll einhalten, wird P den Beweis jedesmal erfolgreich antreten können und V muss akzeptieren, da

$$y_i^2 \prod_{j=1}^{k} v_j^{e_{ij}} = r_i^2 \prod_{j=1}^{k} (s_j^2 v_j)^{e_{ij}} = r_i^2 = x_i \pmod n$$

Korrektheit: Wenn P die einzelnen s_j nicht kennt, dann kann P versuchen die Werte für e_{ij} zu erraten und dann folgendes zu schicken.

$$x_i = r_i^2 \prod_{j=1}^{k} v_j^{e_{ij}} \pmod n \text{ und } y_i = r_i$$

Da in jeder Runde aber ingesamt k Werte zu raten wären, liegt die Wahrscheinlichkeit, dass P das gelingen wird nur bei $\frac{1}{2^k}$. Wird das Protokoll t Runden ausgeführt, sinkt diese sogar auf $\frac{1}{2^{kt}}$.

Zero Knowledge: Wenn unser Simulator in die Zukunft schauen kann, ist es ihm möglich, die korrekten Werte e_{ij} zu erraten. Dadurch kann er immer richtige x_i errechnen und wird den Überprüfungen standhalten. Diese Kommunikation kann dabei nicht von einem echten Beweis unterschieden werden.

Kapitel 3

Besondere Sicherheitsanforderungen

Eigentlich bleibt im Bezug auf Sicherheit in Zero Knowledge Beweisen nur mehr wenig zu wünschen übrig. Die Zero Knowledge Eigenschaft kann mathematisch bewiesen werden, und Information, die nicht übertragen wird, kann auch nicht abgehört werden. Die meisten Protokolle wurden jedoch als 2 Parteien Protokoll entwickelt, weshalb Schwierigkeiten auftreten können, wenn diese Protokolle nicht im Sinne ihres Erfinders ausgeführt werden. Das vorliegende Kapitel soll nun diese Probleme behandeln und veranschaulichen. Im Falle von Concurrent und Resettable Zero Knowledge soll nur die Idee verdeutlicht werden, da eine genaue Ausführung den Rahmen dieser Arbeit sprengen würde.

3.1 Serielle vs. parallele Ausführung

Nachdem Zero Knowledge Beweise, wie wir sie bisher kennengelernt haben, eine große Rundenanzahl erfordern, um den Sicherheitsanforderungen gerecht zu werden, wäre es wünschenswert diese zu beschleunigen. Eine Möglichkeit wäre, das Protokoll nicht seriell sondern parallel ablaufen zu lassen. Dabei werden im Allgemeinen die Nachrichten, die ausgetauscht werden müssen, auf ein Minimum reduziert. Mit dieser Überlegung erstellten auch Feige, Fiat und Shamir in [FFS87] eine parallele Version ihres Protokolls, welches wir bereits in Kapitel 2.4.4 besprochen haben.

Paralleles Feige Fiat Shamir Protokoll: Dieses Protokoll bietet im Wesentlichen die selben Sicherheiten wie die serielle Version, ohne aber mehrere Runden laufen zu müssen. In ihrem Artikel erklären Fiat und Shamir, dass dieses Protokoll nicht mehr Zero Knowledge ist, da es nicht mehr möglich ist einen Simulator dafür zu präsentieren. Sie zeigten jedoch, dass keine Information übermittelt wird, mit der V etwas Sinnvolles anfangen

könnte. Das Protokoll läuft im Wesentlichen wieder gleich ab wie in Kapitel 2.4.4, jedoch mit folgenden Unterschieden:

1. P berechnet alle x_i im Vorhinein und schickt diese an V.
2. V erstellt und sendet darauf die binäre Matrix e_{ij}.
3. P berechnet dann alle y_i und schickt diese an V zur Überprüfung.

In ihrem Artikel [SI92] zeigten Sakurai und Itoh jedoch, dass es mit der parallelen Version des Protokolls möglich ist, Signaturen zu fälschen, die mit dem Signaturschema von Fiat und Shamir erstellt wurden. Um uns dies beweisen zu können, müssen wir uns noch kurz das Signaturschema nach [FS86] ansehen.

Fiat-Shamir-Signaturschema: Dieses Schema ist der parallelen Version des Zero Knowledge Beweises recht ähnlich, jedoch wird kein Beweis durchgeführt, sondern eine Signatur erzeugt. Das Setup verläuft dabei gleich wie in Kapitel 2.4.4. Die Operationen von V übernimmt jedoch eine Funktion f.

1. P wählt zufällig $r_1, ..., r_t \in [0, n)$ und berechnet $x_i = r_i^2 \pmod n$.
2. P berechnet $f(m, x_1, ..., x_t)$ und benutzt die ersten kt Bits als e_{ij} Werte $(1 \leq i \leq t, 1 \leq j \leq k)$. m stellt dabei die zu signierende Nachricht dar.
3. P berechnet
$$y_i = r_i \prod_{j=1} s_j^{e_{ij}} \pmod n$$
für $i = 1, ..., t$ und sendet I, m, die e_{ij} Werte und alle y_i an V.

1. V berechnet $v_j = f(I, j)$ für $j = 1, ..., k$.
2. V berechnet dann
$$z_i = y_i^2 \prod_{j=1}^{k} v_j^{e_{ij}} \pmod n$$
für $i = 1, ..., t$.
3. V prüft, ob die ersten kt Bits von $f(m, z_1, ..., z_t)$ gleich e_{ij} sind.

V akzeptiert jede Signatur, die tatsächlich von P kommt, denn

$$z_i = y_i^2 \prod_{j=1}^{k} v_j^{e_{ij}} = r_i^2 \prod_{j=1}^{k} (s_j^2 v_j)^{e_{ij}} = r_i^2 = x_i \pmod n$$

und daher auch $f(m, z_1, ..., z_t) = f(m, x_1, ..., x_t)$.

Lassen wir nun V bei der Durchführung des Protokolls etwas schwindeln. V weiß, dass P sich immer mit der parallelen Protokollversion von Fiat und Shamir authentisiert, da diese in nur einer Runde abgeschlossen ist. Außerdem erstellt P auch seine Signaturen mit dem Signaturschema von Fiat und Shamir. V will für gewisse Dokumente digitale Signaturen von P erhalten. Allerdings würde P sie niemals unterschreiben, wenn er sie zu Gesicht bekommt. V hat sich also folgendes einfallen lassen.

Er beginnt eine normale Kommunikation mit P und will, dass sich dieser authentisiert. P tut dies bereitwillig, in dem Glauben, dass V ja ohnehin keine nützlichen Informationen bei diesem Vorgang erhält.

1. P berechnet $(x_1, ..., x_t)$ und schickt diese an V.

2. V benutzt zur Erstellung seiner e_{ij} die gleiche Funktion f, die P zum Erstellen der Signaturen verwendet. Er wählt also keine zufälligen Werte, sondern macht sie von den empfangenen x_i und von m abhängig, indem er $(e_1, ..., e_t) = f(m, x_1, ..., x_t)$ berechnet. Er kann so die Nachricht m in das Protokoll einbauen, ohne dass P das bemerken würde. Dieser Schritt erinnert sehr stark an das vorige Signaturschema. Schließlich schickt V protokollkonform die Bits zurück an P.

3. Dieser berechnet gutgläubig die y_i Werte für seinen Beweis und schickt sie an V zurück.

4. V freut sich, denn mit I, m, seinen e_{ij} Werten und den empfangenen y_i hält er eine gültige Signatur von P für die Nachricht m in der Hand. Diese kann er nun jedem zeigen, der sie nach Überprüfung auch als von P unterschrieben anerkennen wird.

Sakurai und Itoh zeigten also, dass, obwohl in der Tat keine nützliche Information in der parallelen Version übermittelt wird, es möglich ist, durch Verknüpfen von 2 eigentlich unabhängigen Protokollen gültige Unterschriften zu erzeugen.

Dieses Beispiel soll verdeutlichen, dass es von extremer Wichtigkeit ist, immer an alle Möglichkeiten zu denken. Zu einem späteren Zeitpunkt wurde eine parallele Version des Protokolls entwickelt, die ihre Zero Knowledge Eigenschaft beibehält (allerdings wieder auf Kosten der Komplexität) und daher auch keine Fälschung von Unterschriften mehr ermöglicht. Wenn serielle Protokolle in parallele umgewandelt werden sollen, reicht es nicht die Schritte einfach alle auf einmal durchzuführen, sondern es muss auch dafür gesorgt werden, dass die Sicherheit (also die Zero Knowledge Eigenschaft) erhalten bleibt.

3.2 Concurrent Zero Knowledge

Protokolle der Vergangenheit beruhen (meist) auf der Annahme, dass 2 Parteien in einer, mehr oder weniger abgeschotteten, Umgebung miteinander kommunizieren. Die Realität sieht leider anders aus. Wenn 2 Parteien über das Internet kommunizieren, kann nicht garantiert werden, über welche Wege die Kommunikation läuft oder wer alles mitlauscht. In den meisten Fällen kann nicht einmal gewährleistet werden, dass man überhaupt mit dem gewünschten Partner spricht. In diesen Fällen ist die Anwendung von Zero Knowledge äußert sinnvoll. Es macht nichts, wenn man nicht mit dem beabsichtigten Partner redet, da keinerlei Information preisgegeben wird.

Jedoch wurden auch Zero Knowledge Protokolle nicht mit einem Blick auf die Möglichkeiten des Internet entwickelt. In ihrer Arbeit [DNS98] zeigten Dwork, Naor und Sahai, dass viele Protokolle nicht ohne weiteres für eine Ausführung im Internet geeignet sind, und die Zero Knowledge Eigenschaft verlieren.

Im Internet ist es möglich, dass ein Prover das gleiche Protokoll mit mehreren Verifiern gleichzeitig ausführt. Das stellt an sich noch kein Problem dar. Jede Instanz für sich wird, wenn sich alle an das Protokoll halten, ordentlich ablaufen und die Zero Knowledge Eigenschaft wird auch gewährleistet sein. Gehen wir aber von einem mächtigen Angreifer aus, der die Protokollinstanzen der Verifier unter Kontrolle hat, so kann dieser die Nachrichten der einen Instanz in einer anderen Nutzen und auf diesem Wege das Protokoll mißbrauchen. Er kann auch eine Instanz zu jedem beliebigen Zeitpunkt (immer wenn er an der Reihe ist, da er den Prover nicht kontrollieren kann) anhalten und mit anderen Instanzen fortfahren, um so vielleicht etwas zu lernen, bevor er mit der ersten Instanz weitermacht. Dwork, Naor und Sahai zeigten, dass es für ein solches Setup nicht mehr möglich ist, einen Simulator zu präsentieren. Dies muss noch nicht bedeuten, dass ein Angreifer dadurch wirklich etwas lernen kann, aber wir haben bereits in Kapitel 3.1 gesehen, dass es keine gute Idee ist, sich auf Vermutungen zu stützen.

Deshalb haben sie auch einen Weg ersonnen, wie man selbst in einem solchen Setup die Zero Knowledge Eigenschaft wieder herstellen kann. Sie führen dazu den Parameter der Zeit in die Protokolle ein. Dabei werden die Systemuhren der beiden Protokollpartner in das Protokoll miteinbezogen. Die Idee ist nun für den Ablauf des Protokolls Verzögerungen und Timeouts einzubauen und die einzelnen Protokollschritte um einige Checks zu erweitern. P generiert z. B. eine Nachricht m, auf die V antworten muss. V muss mit m eine Berechnung durchführen, die zumindest α lang dauert. Deshalb akzeptiert P keine Nachrichten, bevor nicht zumindest die Zeit α verstrichen ist. Damit sich V aber nicht ewig Zeit lassen kann, bricht P das Protokoll nach β Sekunden ab, wenn er in dieser Zeit keine Antwort von V erhält.

Durch diese zusätzlichen Überprüfungen im Protokoll ist es möglich Zero

Knowledge Protokolle zu entwickeln, die auch den Möglichkeiten eines Angreifers im Internet trotzen. Es ist zwar weiterhin möglich Protokolle mehrfach laufen zu lassen, einige anzuhalten und zu einem späteren Zeitpunkt wieder aufzunehmen, allerdings wird in einem solchen Fall das Protokoll von P abgebrochen werden, da die Schranken nicht eingehalten wurden.

Dwork, Naor und Sahai entwickelten mit dieser Methode mehrere Protokolle, für die sie auch einen Simulator präsentieren konnten, also die Zero Knowledge Eigenschaft bewiesen haben. Auf diese Protokolle wird in dieser Arbeit nicht mehr näher eingegangen, sie können aber in [DNS98] nachgelesen werden.

3.3 Resettable Zero Knowledge

In [CGGM00] führten Canetti, Goldreich, Goldwasser und Micali eine neue Definition von Zero Knowledge ein, die als die bisher strengste gilt. Ein normaler Zero Knowledge Beweis braucht echte Zufallszahlen (sowohl P als auch V müssen Münzen werfen), um ordnungsgemäß ablaufen zu können. Die Anforderungen der realen Welt, wie z. B. Smart Cards, können dies allerdings nicht gewährleisten. Resettable Zero Knowledge geht soweit, dass es einem Angreifer sogar möglich ist, P zurückzusetzen (resetten) und ihn zu zwingen die selben Zufallszahlen erneut zu verwenden.

Man könnte sich nun fragen, ob eine solch strenge Definition überhaupt noch Sinn macht, schließlich wird es keinen solchen Angreifer geben. Smart Cards jedoch können zu jeder Zeit einfach zurückgesetzt werden, indem man ihnen einfach die Stromversorgung kappt.

Jedes der „normalen“ Zero Knowledge Protokolle kann dieser neuen Definition nicht standhalten. Wird P gezwungen die gleichen Zufallszahlen erneut zu verwenden, ist es in den meisten Fällen möglich, das Geheimnis in nur wenigen Runden herauszufinden. Damit ist natürlich die Zero Knowledge Eigenschaft völlig verletzt.

Wie kann man durch Resetten von P den diskreten Logarithmus herausfinden? Um dies zu verdeutlichen nehmen wir erneut unser Beispiel aus Kapitel 2.4.1 her und verändern den Ablauf ein wenig:

1. P wählt zufällig eine Zahl e und schickt $B = b^e$.
2. V wirft keine Münze sondern, schickt einfach Kopf an P.
3. P muss e bekanntgeben. V prüft ob $B = b^e$.
4. V resettet P. Das Protokoll beginnt von vorne, jedoch muss P die gleichen Zufallszahlen verwenden.
5. P schickt das gleiche e wie vorher und schickt $B = b^e$.
6. V schickt Zahl an P.

7. P muss $x+e$ bekanntgeben. V prüft noch, ob $y \cdot B = b^{x+e}$ und beendet das Protokoll.

V freut sich jetzt, denn er kennt sowohl e, als auch $x+e$. So fällt es ihm nicht schwer x zu berechnen. Fortan kann sich V fröhlich als P ausgeben, da er jetzt dessen Geheimnis kennt.

Resettable Zero Knowledge soll nun sogar einen Angreifer scheitern lassen, dem erlaubt wird, P mehrfach zurückzusetzen, wobei P jedesmal die gleiche Konfiguration und Zufallszahlen verwenden muss. Weiters ist es V auch erlaubt, mehrere Instanzen eines Protokolls mit P laufen zu lassen, und Nachrichten der einen Instanz als Input einer anderen Instanz zu verwenden, ohne dass er einen Nutzen davon hätte. Jedes Resettable Zero Knowledge Protokoll ist daher auch gleichzeitig Concurrent Zero Knowledge. Daraus ergibt sich auch, dass Concurrent Zero Knowledge eine weitaus weniger strenge Definition als Resettable Zero Knowledge hat. In ihrer Arbeit zeigten sie weiters, dass sich jedes Concurrent Zero Knowledge Protokoll zu einem Resettable Zero Knowledge Protokoll (über Umwege) erweitern lässt.

In [CGGM00] wurden auch mehrere Protokolle vorgestellt, die der neuen Definition genügen. Sie sind allerdings nicht mehr einfach zu verstehen und erfordern bereits einiges an Hintergrundwissen.

Kapitel 4

Zusammenfassung und Ausblick

Obwohl Zero Knowledge an sich ein erst relativ junges Gebiet ist, wurde das Prinzip in den letzten 20 Jahren um vieles verfeinert und erweitert. Nach der Definition wurde bald gezeigt, dass alle Sprachen in $\mathcal{NP}$ Zero Knowledge Beweise haben. Es folgte eine Fülle von Zero Knowledge Protokollen, die auf verschiedenen kryptographischen Schwierigkeiten beruhen (z. B. RSA, DLP, ...) aber im Wesentlichen die gleiche Sicherheit bieten. Es wurde auch der Wunsch laut die Anzahl der Runden, die ein Zero Knowledge Protokoll laufen muss, zu verkürzen. So wurden zum Teil parallelisierte Versionen von Protokollen vorgestellt, oder neue Protokolle mit fixer Rundenzahl entworfen.

In der letzten Zeit verlegte man die 2 Parteien Protokolle in ein großes Netzwerk wie das Internet und erkannte, dass zusätzliche Sicherheitsmechanismen vonnöten sind. Man gab einem Angreifer immer mehr (theoretische) Macht und entwickelte Protokolle, die selbst einem mächtigen Angreifer standhalten. Concurrent Zero Knowledge hält Angreifer ab, die mehrere Instanzen eines Protokolls in beliebiger Reihenfolge ablaufen lassen können, und dabei auch noch Nachrichten von den Ergebnissen einer anderen Protokollausführung abhängig machen können.

Die Krönung im Bezug auf Sicherheit stellt bis heute Resettable Zero Knowledge dar. In solchen Protokollen ist es einem Angreifer sogar möglich, den Prover jederzeit zurückzusetzen und ihn zu zwingen die gleichen Zufallszahlen wie vorher zu benutzen. Bisherige Zero Knowledge Protokolle halten einem solchen Angreifer nicht mehr stand. Eine neue Generation von Zero Knowledge Protokollen ist gefragt. Auf diesem Gebiet erscheinen derzeit sicher die meisten Veröffentlichungen.

Ob Resettable Zero Knowledge den Höhepunkt darstellt, oder ob noch strengere Definitionen erfunden werden, wird die Zukunft zeigen. Es könnte aber leicht sein, dass schon morgen ein findiger Kryptologe auf eine gravie-

rende Schwäche von Resettable Zero Knowledge stößt und eine neue Definition sich durchsetzen wird.

Schließlich hängt auch Zero Knowledge an einem dünnen Ast, denn es ist nach wie vor nicht bewiesen, dass die Probleme, die der modernen Kryptographie und ihren Anwendungen zu Grunde liegen, schwierig sind. Gelingt es jemandem ein $\mathcal{NP}$-vollständiges Problem zu lösen, also zu zeigen, dass $\mathcal{P} = \mathcal{NP}$, dann wäre es vorläufig auch mit Zero Knowledge vorbei.

Literaturverzeichnis

[BCC88] Brassard, Gilles, David Chaum und Claude Crêpeau: *Minimum disclosure proofs of knowledge.* J. Comput. Syst. Sci., 37(2):156–189, 1988.

[BM88] Babai, László und Shlomo Moran: *Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity class.* J. Comput. Syst. Sci., 36(2):254–276, 1988.

[CGGM00] Canetti, Ran, Oded Goldreich, Shafi Goldwasser und Silvio Micali: *Resettable zero-knowledge (extended abstract).* In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, Seiten 235–244. ACM Press, 2000.

[Cha90] Chaum, D.: *Zero-knowledge undeniable signatures (extended abstract).* In: Damgård, Ivan B. (Herausgeber): *Advances in Cryptology - EuroCrypt '90*, Seiten 458–464, Berlin, 1990. Springer-Verlag. Lecture Notes in Computer Science Volume 473.

[DH76] Diffie, Whitfield und Martin E. Hellman: *New Directions in Cryptography.* IEEE Transactions on Information Theory, IT-22(6):644–654, 1976.

[DNS98] Dwork, Cynthia, Moni Naor und Amit Sahai: *Concurrent zero-knowledge.* In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, Seiten 409–418. ACM Press, 1998.

[Eck04] Ecker, Jürgen: *Automatentheorie und Kryptologie.* Fachhochschule Hagenberg, Skriptum zur Vorlesung, Seiten 75–77, 2002–2004.

[FFS87] Feige, U., A. Fiat und A. Shamir: *Zero knowledge proofs of identity.* In: *Proceedings of the nineteenth annual ACM conference on Theory of computing*, Seiten 210–217. ACM Press, 1987.

[FS86] Fiat, A. und A. Shamir: *How to prove yourself: practical solutions to identification and signature problems.* In: Odlyzko, A. M. (Herausgeber): *Advances in Cryptology - Crypto '86*, Seiten 186–194, Berlin, 1986. Springer-Verlag. Lecture Notes in Computer Science Volume 263.

[GMR85] Goldwasser, S, S Micali und C Rackoff: *The knowledge complexity of interactive proof-systems.* In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, Seiten 291–304. ACM Press, 1985.

[GMW86] Goldreich, Oded, Silvio Micali und Avi Wigderson: *Proofs that yield nothing but their validity, and a methodology of cryptographic protocol design.* In: *Proceedings of the 27th FOCS*, Seiten 174–187, 1986.

[GMW91] Goldreich, Oded, Silvio Micali und Avi Wigderson: *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems.* J. ACM, 38(3):690–728, 1991.

[Gol01] Goldreich, Oded: *Foundations of Cryptography, Volume 1: Basic Tools.* Cambridge University Press, 2001.

[GQ89] Guillou, L. C. und J. J. Quisquater: *A "paradoxical" identity-based signature scheme resulting from zero-knowledge.* In: Goldwasser, Shafi (Herausgeber): *Advances in Cryptology - Crypto '88*, Seiten 216–231, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science Volume 403.

[OO89] Okamoto, T. und K. Ohta: *Disposable zero-knowledge authentications and their applications to untraceable electronic cash.* In: Brassard, Gilles (Herausgeber): *Advances in Cryptology - Crypto '89*, Seiten 481–497, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science Volume 435.

[QGAB89] Quisquater, J. J., L. C. Guillou, M. Annick und T. A. Berson: *How to explain zero-knowledge protocols to your children.* In: Brassard, Gilles (Herausgeber): *Advances in Cryptology - Crypto '89*, Seiten 628–631, Berlin, 1989. Springer-Verlag. Lecture Notes in Computer Science Volume 435.

[RSA78] Rivest, Ronald R., Adi Shamir und Leonard M. Adleman: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM, 21(2):120–126, 1978.

[SI92] Sakurai, K. und T. Itoh: *On the discrepancy between serial and parallel of zero-knowledge protocols.* In: Brickell, Ernest F. (Herausgeber): *Advances in Cryptology - Crypto '92*, Seiten 246–259, Berlin, 1992. Springer-Verlag. Lecture Notes in Computer Science Volume 740.

[Tur37] Turing, Alan: *On Computable Numbers, with and Application to the Entscheidungsproblem.* Proceedings of the London Mathematical Society, 42(2):230–265, 1936–1937.

[Vol91] Volkmann, Lutz: *Graphen und Digraphen. Eine Einführung in die Graphentheorie.* Springer Verlag Wien, 1991.

[Woe03] Woeginger, Gerhard J.: *Exact algorithms for NP-hard problems: a survey.* Seiten 185–207, 2003.