Zero-Knowledge-Verfahren

Inhaltsverzeichnis

1 Einleitung

1.1 Zielstellung des Versuches

Bei dem Begriff Zero-Knowledge[1]-Verfahren bzw. Zero-Knowledge-Beweis, mag der ein oder andere vielleicht an seine letzte Prüfung denken, in der er eindrucksvoll *Null Wissen* bewiesen hat. Zwar wird auch in den hier betrachteten Zero-Knowledge-Verfahren *Null Wissen* während des Beweises übermittelt, dennoch ist der verifizierende Teilnehmer anschließend überzeugt, dass der Beweiser ein bestimmtes Wissen besitzt.

Wie solch ein Beweis funktioniert und als kryptographisches Protokoll[2] angewendet werden kann, soll in diesem Versuch vermittelt werden. Dazu werden im folgenden Kapitel zunächst die nötigen Grundlagen zu Commitment-Verfahren und interaktiven Beweisen vorgestellt, wobei auch von bereits vorhandenem Wissen aus den vorherigen Versuchen — insbesondere dem Versuch "Zahlentheoretische Algorithmen" — ausgegangen wird. Anschließend können in Kapitel 3 das generelle Vorgehen und die besonderen Eigenschaften von Zero-Knowledge-Beweisen betrachtet werden, die konkrete Umsetzung dieser Technik wird dabei anhand von drei bekannten Protokollen erläutert (Kapitel 4). In Kapitel 5 und 6 geht es dann um die Besonderheiten bei der Komposition, also z.B. der parallelen Ausführung solcher Protokolle, welche auch die Nutzung von Zero-Knowledge-Verfahren für digitale Signaturen ermöglicht. Abschließend wird mit der Man-In-The-Middle-Problematik eine Angriffsmöglichkeit auf Zero-Knowledge-Verfahren und eine entsprechende Abwehrmaßnahme vorgestellt. Durch die Implementierung eines der Zero-Knowledge-Protokolle in den verschiedenen Einsatzszenarien, können das Vorgehen bei Zero-Knowledge-Beweisen und deren Möglichkeiten ausprobiert werden.

1.2 Motivation Zero-Knowledge-Verfahren

Bei Zero-Knowledge-Verfahren handelt es sich um spezielle interaktive Beweissysteme die es einem Teilnehmer — dem *Beweiser* — ermöglichen, einen anderen Teilnehmer — den *Verifizierer* — zu überzeugen, über ein bestimmtes geheimes Wissen zu verfügen, ohne dieses preisgeben zu müssen.

Als Goldwasser, Micali und Rackhoff das Konzept der Zero-Knowledge-Verfahren 1985 in [GMR85] vorstellten, löste dies eine stürmische Entwicklung in der kryptographischen und komplexitätstheoretischen Forschung aus. Denn Zero-Knowledge-Verfahren faszinieren durch ihre widersprüchliche Definition: Mit ihnen können einerseits überzeugende Beweise geführt werden, und andererseits verraten sie nichts außer der Gültigkeit einer Aus-

[1] engl: Zero = Null, Knowledge = Wissen

[2] Die Begriffe Verfahren, Beweis und Protokoll können nahezu als Synonyme verwendet werden. Zero-Knowledge-Verfahren sind spezielle interaktive Beweise und stehen meist für generelle Vorgehensweisen. Anwendungen oder konkrete Abläufe der Verfahren werden als Protokolle bezeichnet.

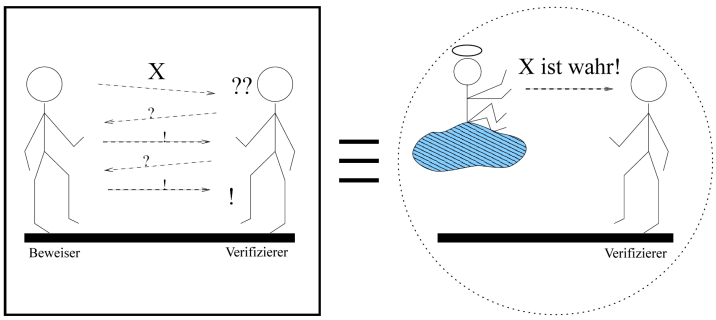

Abbildung 1: Illustration von Zero-Knowledge nach [Gol02]

sage. Aus der Sicht des Verifizierers sind die Antworten des Beweisers in einem Zero-Knowledge-Verfahren damit äquivalent zu einer 1-Bit-Antwort eines vertrauenswürdigen Orakels [Fei90].

Solche Beweissysteme eignen sich daher insbesondere für Identifikations- und Authentifikationsverfahren. Dabei führt ein sich identifizierender Teilnehmer einen Beweis zu einer bestimmten (mathematischen) Behauptung durch, der ihm nur gelingen kann, wenn er über das ihn identifizierende individuelle Geheimnis verfügt. Die verifizierende Instanz kann dadurch von den Identitäten der jeweiligen Teilnehmer überzeugt werden, ohne etwas über ihre Geheimnisse zu erfahren. Dies ist ein enormer Vorteil gegenüber üblichen Beweisverfahren, wie sie z.B. zur Authentifikation an Bankautomaten eingesetzt werden, denn dabei ist ein Nachweis von Wissen nur durch das Präsentieren des zugehörigen Geheimnisses möglich. Bisher muss ein Teilnehmer zur Authentifikation an Bankautomaten seine persönliche Geheimzahl (PIN) eingeben, so dass jeder, der diese Eingabe beobachtet, das Geheimnis des Teilnehmers erfährt. Zudem muss die PIN auch dem Banksystem bekannt sein, wo sie als Vergleichswert benötigt wird. Mithilfe von Zero-Knowledge-Verfahren können hingegen Authentifikationssysteme erstellt werden, in denen die PIN nie den „sicheren“ Chip verlassen muss und damit auch niemandem, außer dem Teilnehmer selbst, bekannt ist.

1.3 Einführendes Beispiel: Ali Babas Höhle

Das Grundprinzip von Zero-Knowledge-Verfahren wurde sehr anschaulich von Quisquater und Guillou in „How to explain zero-knowledge protocols to your children“ [QGAB89] anhand einer Geschichte von der mystischen Höhle Ali Babas beschrieben. Bevor wir uns den konkreten mathematischen Umsetzungen von Zero-Knowledge und dessen Grundlagen zuwenden, soll dieses Beispiel zunächst eine Idee über den generellen Ablauf von Zero-Knowledge-Verfahren geben.

Stellen wir uns dazu eine Höhle vor, wie sie in Abbildung 2 dargestellt ist. Diese Höhle verzweigt nach dem Eingang in zwei Richtungen, wobei beide Wege in einer Sackgasse enden. Allerdings sind die Gänge durch eine Geheimtür miteinander verbunden, die sich

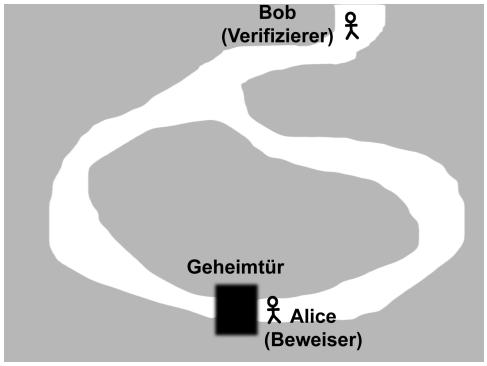

Abbildung 2: Ali Babas Höhle

durch ein bestimmtes Passwort öffnen lässt. Alice kennt dieses Passwort und will ihr Wissen Bob beweisen, ohne ihm aber das Passwort verraten zu müssen. Alice kann dazu folgendes Spiel mit Bob durchführen: Sie befiehlt Bob, vor dem Höhleneingang zu warten und geht nun selbst hinein, wobei sie sich willkürlich für einen der beiden Wege entscheidet. Bob geht kurze Zeit später ebenfalls in die Höhle, bleibt allerdings vor der Verzweigung stehen. Von da aus kann er Alice nicht sehen, Bob hat also keine Ahnung, in welchem Teil sie sich befindet. Anschließend wirft Bob eine Münze — bei Kopf muss Alice aus dem linken Gang herauskommen, bei Zahl aus dem Rechten. Das Ergebnis des Münzwurfes schreit Bob in die Höhle, so dass Alice hört, aus welchem Gang sie erscheinen soll. Da sie das Passwort kennt, ist das für sie kein Problem. Entweder sie befindet sich bereits in dem richtigen Gang oder sie verwendet ihr Passwort und gelangt durch die Geheimtür auf die andere Seite. In beiden Fällen erscheint sie also aus dem richtigen Gang. Nach der ersten Durchführung des Beweises wird Bob aber sicher nicht allzu beeindruckt sein, da schließlich eine 50% Chance für Alice bestand, im richtigen Gang gewesen zu sein. Also wiederholen sie das Spiel solange bis Bob überzeugt ist. Kennt Alice das Passwort, wird es ihr jedes Mal gelingen auf der von Bob gewünschten Seite aufzutauchen. Ohne das Passwort ist die Chance, immer aus dem korrekten Gang zu erscheinen, jedoch verschwindend gering, da Alice dazu jedes Mal die Wahl von Bob richtig geraten haben muss — bei 10 Versuchen ist die Wahrscheinlichkeit dafür nur $\frac{1}{2^{10}}$, d.h. $\frac{1}{1024}$. Mit solch einer Beweisform kann ein Beweiser, der das Passwort kennt, einen Verifizierer immer von seinem Wissen überzeugen, während die Erfolgschance eines betrügenden Beweisers bei mehreren Durchläufen vernachlässigbar gering ist. Denn bereits beim ersten Fehlversuch kann der Beweis abgebrochen werden.

Nehmen wir nun an, dass Bob mit einer Videokamera alles aufnimmt, was er sieht. In jedem Durchgang wird damit aufgezeichnet, wie Alice in der Höhle verschwindet, was Bob ihr zuruft und aus welchem Gang Alice erscheint. Will Bob mit dieser Aufnahme nun eine dritte Person Carol überzeugen, dass Alice tatsächlich das Passwort der Geheimtür kennt, wird ihm das aber nicht gelingen. Denn Bob könnte ein solches Videoband auch mit Malice — der bösen Zwillingsschwester von Alice, welche keine Ahnung von dem Passwort hat — erstellen. Dazu müssen sie sich nur vor jeder Beweisrunde absprechen, so dass Malice

jedesmal gleich zu Beginn den entsprechenden Gang wählen kann. Sie würde dann auch ohne die Kenntnis des Passwortes, in jeder Runde aus dem von Bob gewünschten Gang erscheinen. Selbst wenn sich Bob und Malice nicht absprechen, könnte Bob ein Video erstellen in dem Malice scheinbar das Passwort kennt. Dazu führen sie den Beweis wie gewohnt durch, d.h. Malice wählt zufällig einen der Gänge und Bob entscheidet ebenfalls zufällig aus welchem sie zu erscheinen hat. Haben beide die selbe Wahl getroffen, kommt Malice freudestrahlend aus dem gewünschten Gang und eine korrekte Beweisrunde ist aufgezeichnet. Stimmte der Wunsch von Bob allerdings nicht mit der Entscheidung von Malice überein, löschen sie diesen Durchgang einfach vom Videoband. Mit beiden Methoden kann Bob ein Video anfertigen, das von einer echten Aufzeichnung eines Beweises von Alice Wissen nicht unterschieden werden kann.

Die Möglichkeit solche scheinbar korrekten Videos zu erstellen, zeigt, dass es sich bei dem Beweis um einen Zero-Knowledge-Beweis handeln muss. Denn wenn ein gefälschtes Video ohne Kenntnis des Geheimnisses erstellt werden kann, kann auch aus der Durchführung eines echten Beweises keine Information über das Geheimnis erlangt werden.

2 Grundlagen

Damit wir die Eigenschaften solch eines Beweises auch formal fassen und beweisen werden können, benötigt es noch einiges an Definitionen und Grundlagen auf denen Zero-Knowledge-Verfahren basieren. Grundlegendes Wissen zur Komplexitätstheorie wurde bereits im Versuch “Zahlentheoretische Algorithmen” vermittelt und wird daher vorausgesetzt.

2.1 Commitment-Verfahren

Commitment-Verfahren bieten bei einer Kommunikation mehrerer Teilnehmer die Möglichkeit, sich verbindlich auf einen Wert festlegen zu können, ohne diesen sofort preiszugeben.

Solche Verfahren laufen stets in 2 Phasen ab – der Festlegungsphase (engl: *commit stage*) und der Öffnungsphase (engl: *opening stage*). In der Festlegungsphase legt sich der Sender S auf einen Wert m fest, so dass er nachher nicht mehr verändert werden kann und ein Empfänger R keine Möglichkeit hat, zu erkennen auf welchen Wert sich S festgelegt hat. Die so entstandene Nachricht wird *Festlegung* oder *Commitment* genannt und an R übermittelt. In der Öffnungsphase wird die festgelegte Nachricht m dem Empfänger bekannt gegeben, wobei durch das Commitment-Verfahren garantiert ist, dass S auch nur die ursprünglich festgelegte Nachricht m bekannt geben kann [CDG88]. Folgende Eigenschaften muss jedes Commitment-Verfahren besitzen:

- **Eindeutigkeit** (engl: *binding property*) Der Sender kann den Wert nach der Festlegungsphase nicht mehr ändern.

- **Geheimhaltung** (engl: *hiding property*) Der Empfänger erfährt vor der Öffnungsphase nichts über den festgelegten Wert.

Commitment-Verfahren können daher als Analogon zu versiegelten, undurchsichtigen Briefumschlägen aufgefasst werden. Denn wird eine Nachricht in einem Umschlag versiegelt, kann sie anschließend nicht mehr verändert werden und der Empfänger kann die Nachricht nicht lesen, solange der Umschlag verschlossen ist.

In der digitalen Welt lassen sich Commitment-Verfahren durch kollisionsresistente Einwegfunktionen realisieren. Der Sender berechnet dazu mit Hilfe einer öffentlich bekannten Einwegfunktion f die Festlegung auf eine Nachricht m durch $c = f(m)$ und übermittelt c an R. In der Öffnungsphase sendet S die Nachricht m an den Empfänger, welcher nun mittels der Funktion f überprüft, ob $f(m)$ dem erhaltenen Commitment c entspricht. Aufgrund der Kollisionsresistenz von f ist es in polynomialer Zeit nicht möglich, ein $m' \neq m$ mit $f(m') = f(m)$ zu finden, was die Eindeutigkeits-Eigenschaft des Commitment-Verfahrens sicherstellt. Die Geheimhaltung wird durch die Einwegfunktion f an sich garantiert, denn da es sich bei einer Eingwegfunktion um eine schwer zu invertierende Funktion handelt, ist ein Empfänger nicht in der Lage, $f^{-1}(c)$ effizient zu berechnen.

Commitment-Verfahren bilden die Grundlage der meisten Zero-Knowledge-Verfahren, und auch in dem Höhlen-Beispiel legte sich Alice zu Beginn des Beweises auf einen Wert fest. Sie wählte einen der beiden Gänge, ohne das Bob erkennen konnte, für welchen sie sich entschieden hat. Alice konnte ihre Wahl auch nicht mehr ändern, da Bob ihr anschließend bis zur Verzweigung folgte.

Für solch eine Festlegung auf nur ein Bit, versagen allerdings normale Commitment-Verfahren, da auf der Menge $\{0, 1\}$ keine kollisionsresistenten Einwegfunktionen existieren. Dieses Problem wurde durch Bit-Commitment-Verfahren[3] gelöst, bei denen man sich nun neben dem Bit zusätzlich auf eine hinreichend große Zufallszahl festzulegen hat [BCC88]. Bit-Commitment-Verfahren haben in Verbindung mit Zero-Knowledge eine wichtige Bedeutung, denn unter der Annahme, dass solche Bit-Commitment-Verfahren existieren, konnte in [GMW86] bewiesen werden, dass es Zero-Knowledge-Beweise für alle Probleme in $\mathcal{NP}$ gibt.

2.2 Interaktive Beweissysteme

Bei dem Begriff „Beweis“ denken sicherlich die meisten an mathematische Beweise, in denen neue Aussagen logisch aus Axiomen oder bereits bewiesenen Aussagen abgeleitet werden. Solch ein traditioneller Beweis ist immer ein starres Objekt und kann, nachdem er entwickelt wurde, von jedem, der sich von der Korrektheit der bewiesenen Aussage überzeugen will, verifiziert werden. Betrachtet man den Beweisbegriff aber außerhalb der mathematischen

[3] Ausgehend von [BCC88] werden Bit-Commitments manchmal auch als *Blobs* bezeichnet

Welt, z.B. bei Gerichtsverhandlungen, so fällt auf, dass in diesem Fall die Interaktion zwischen mehreren Parteien eine große Rolle spielt.

Die Idee, solche Interaktionen auch in mathematischen Beweisen anzuwenden, hatten Goldwasser, Micali und Rackhoff, die 1985 die interaktiven Beweissysteme [GMR85] vorstellten sowie Babai, der unabhängig die sogenannten Arthur-Merlin-Beweissysteme [BM88] entwickelte. Dennoch war das Konzept zu diesem Zeitpunkt nicht gänzlich neu, denn bereits 1535 gab es den ersten schriftlich erwähnten interaktiven Beweis [BJW99]. In diesem Jahr entdeckte der italienische Mathematiker Niccolo Tartaglia eine Formel, mit der sich Polynome dritten Grades, d.h. Gleichungen der Form

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

lösen lassen. Tartaglia besaß allerdings keinen akademischen Grad und wollte, vermutlich aus Angst vor anderen etablierten Mathematiker, seine Lösungsformel nicht preisgeben. Um dennoch beweisen zu können, dass er die Formel tatsächlich kennt, führte Tartaglia einen Wettstreit mit dem italienischen Rechenmeister Antonio Maria Forio. Forio wollte Tartaglias Behauptung überprüfen, indem er ihm 30 Gleichungen dritter Ordnung sandte. Tartaglia konnte dank seiner Formel sämtliche Gleichungen lösen und schickte die Ergebnisse zurück an Forio. Dieser konnte nun leicht überprüfen, ob die Lösungen korrekt waren und wurde so von Tartaglias Wissen überzeugt, ohne dabei die Lösungsformel erfahren zu haben.

2.2.1 Definition nach Goldwasser, Micali und Rackhoff

Ist heutzutage in der Kryptographie von interaktiven Beweisen die Rede, sind in der Regel die Konzepte und Definitionen von [GMR85] bzw. der revidierten Fassung [GMR89] gemeint. Die Arthur-Merlin-Beweissysteme von Babai wurden stärker mit Blickrichtung auf komplexitätstheoretische Ergebnisse entwickelt und können zudem als Spezialfall der interaktiven Beweissysteme angesehen werden.

Interaktive Beweissysteme ergeben sich stets aus einem Dialog zwischen einem *Beweiser* $\mathcal{P}$ (von engl. Prover) und einem *Verifizierer* $\mathcal{V}$ (von engl. Verifier). Ziel des Beweisers ist es, den Verifizierer von der Gültigkeit einer bestimmten Behauptung x zu überzeugen. Der Verifizierer hat damit die Aufgabe, am Ende der Interaktion zu entscheiden, ob er den Beweis akzeptiert oder nicht. Der Dialog wird dazu als eine Art Frage-Antwort-Spiel geführt, wobei der Verifizierer als Fragesteller auftritt und der Beweiser die Antworten liefert. Beweiser und Verifizierer können dazu Berechnungen durchführen und Zufallswerte bestimmen. Am Ende der Interaktion entscheidet der Verifizierer, ob er den Beweis akzeptiert, d.h. von der Korrektheit der Behauptung des Beweisers überzeugt ist.

Der Beweis kann in solch einer interaktiven Form allerdings nie zweifelsfrei geführt werden, d.h. $\mathcal{V}$ kann nicht mit absoluter Gewissheit überzeugt werden, dass eine bestimmte Aussage wahr ist, sondern „nur“ mit z.B. 99.9999% Gewissheit. Wichtig für das Beweissystem ist dabei zum einen, dass solch eine überwältigende Wahrscheinlichkeit immer erreicht werden kann, wenn sich Beweiser und Verifizier ehrlich verhalten. Zum anderen soll ein betrügender Beweiser $\mathcal{P}^*$, der versucht, $\mathcal{V}$ von einer falschen Behauptung zu überzeugen, mit

einer ebenso hohen Wahrscheinlichkeit scheitern. Diese fundamentalen Eigenschaften, die jedes interaktive Beweissystem besitzen muss, werden als *Vollständigkeit* und *Korrektheit* bezeichnet:

Definition 1 (Vollständigkeit / Completeness)
Ein interaktives Beweissystem $(\mathcal{P}, \mathcal{V})$ [4] *heißt vollständig, wenn das Protokoll bei ehrlichem Beweiser* $\mathcal{P}$ *und ehrlichem Verifizierer* $\mathcal{V}$ *mit überwältigender Wahrscheinlichkeit zum Erfolg führt, d.h. der Beweiser kann den Verifizierer von der Gültigkeit seiner Aussage überzeugen.*

Definition 2 (Korrektheit / Soundness)
Ein interaktives Beweissystem $(\mathcal{P}, \mathcal{V})$ *heißt korrekt, wenn das Protokoll bei einem betrügenden Beweiser* $\mathcal{P}^*$ *und ehrlichem Verifizierer* $\mathcal{V}$ *nur mit vernachlässigbarer Wahrscheinlichkeit zum Erfolg führt, d.h. ein Beweiser, der das Geheimnis nicht kennt, kann den Verifizierer nicht von der Gültigkeit seiner Aussage überzeugen.*

Formal lassen sich interaktive Beweissysteme durch zwei probabilistische Turingmaschinen darstellen, die miteinander kommunizieren. Der Verifizierer wird dabei stets als polynomial zeitbeschränkte Turingmaschine betrachtet, während der Beweiser keinen komplexitätstheoretischen Beschränkungen unterliegen muss [GMR85]. Die Forderung der polynomialen Rechenzeitbeschränkung des Verifizierers ist notwendig, da er bei unbeschränkter Rechenzeit das Geheimnis des Beweisers ermitteln könnte, bzw. in der Lage wäre, für die Aussage von $\mathcal{P}$ selbst einen Beweis zu finden. Aufgrund dieser Konstruktion kann für jedes

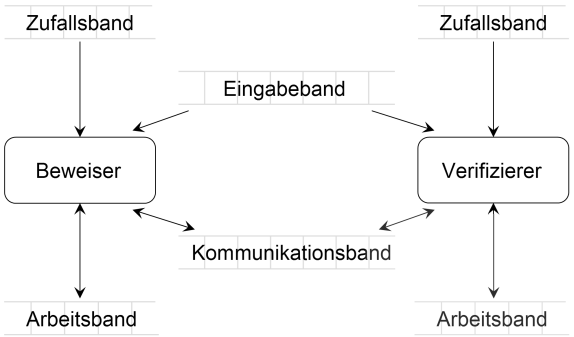

Abbildung 3: Interaktives Beweissystem

Problem in $\mathcal{NP}$ ein interaktiver Beweis angegeben werden. Ein Beweiser ist dank seiner unbeschränkten Rechenzeit in der Lage, einen vollständigen Beweis zu erstellen, welcher anschließend von einem Verifizierer in Polynomialzeit auf Korrektheit überprüft werden kann.

[4] Ein Beweissystem bzw. Protokoll $(\mathcal{P}, \mathcal{V})$ gibt an, wie sich „ehrliche“ Teilnehmer $\mathcal{P}$ und $\mathcal{V}$ verhalten. Für den Fall, dass ein „unehrlicher“ Teilnehmer an dem Protokoll beteiligt ist, schreibt man $\mathcal{P}^*$ bzw. $\mathcal{V}^*$. Dabei können sich $\mathcal{P}^*$ bzw. $\mathcal{V}^*$ ganz oder teilweise wie $\mathcal{P}$ oder $\mathcal{V}$ verhalten.

Die bekannteste und auch für diesen Versuch relevante Beweisform, ist der sogenannte **Wissensbeweis** (engl. *proof of knowledge*), mit dem ein bestimmtes Wissen zu einem öffentlichen Wert x nachgewiesen werden kann. Dazu ist eine Relation R gegeben und die Behauptung die es zu beweisen gilt, ist, dass der Beweiser zu einem bestimmten Wert x – der ihm und dem Verifizierer bekannt ist – einen Wert w kennt, so dass $(x, w) \in R$. Dieser Wert w ist das Geheimnis des Beweisers und wird oft auch Zeuge (engl. *witness*) oder Beweisstück für x genannt. Vollständigkeit und Korrektheit für einen Wissensbeweis lassen sich wie folgt formulieren:

- Vollständigkeit

$$\forall(x, w) \in R \quad \text{Prob}[(\mathcal{P}(w), \mathcal{V})(x) = accept] = 1$$

- Korrektheit

$$\forall \mathcal{P}^* : \forall(x, w') \notin R \quad \text{Prob}[(\mathcal{P}^*(w'), \mathcal{V})(x) = accept] < \frac{1}{|x|^n}$$

Die Vollständigkeit darf hierbei nicht mit der Sicherheit des Verifizierers verwechselt werden. Diese kann, wie bereits erwähnt, nie den Wert 1 erreichen, sondern sich nur beliebig nahe dieser Grenze nähern. Die Annäherung wird dabei durch den Term $\frac{1}{|x|^n}$ bestimmt, der auch Fehler(term) genannt wird und in Abhängigkeit der Rundenzahl n abnimmt.

3 Zero-Knowledge-Verfahren

3.1 Definitionen

Zero-Knowledge-Verfahren sind interaktive Beweise, die neben den Eigenschaften Vollständigkeit und Korrektheit zusätzlich die Zero-Knowledge-Anforderung erfüllen. Das bedeutet, dass ein Verifizierer durch den Beweisablauf kein zusätzliches Wissen bzw. Information erhält, außer ob die Behauptung des Beweisers richtig ist.

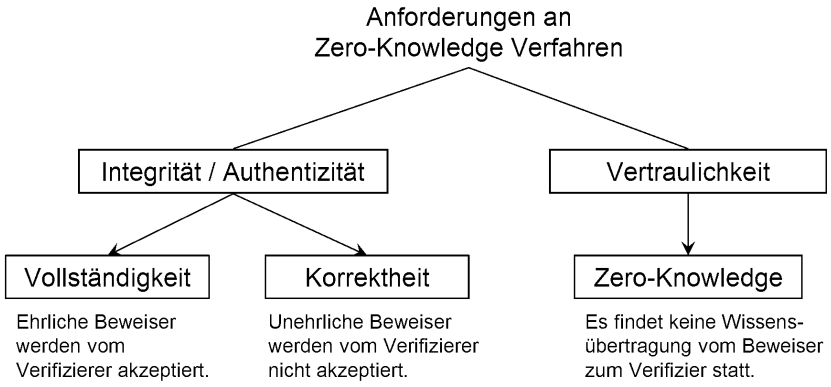

Abbildung 4: Anforderungen an Zero-Knowledge-Verfahren

Unter *zusätzlichem Wissen* ist dabei alles gemeint, was einem Verifizierer nach einer Interaktion mit $\mathcal{P}$ hilft, Berechnungen besser bzw. effizienter als vor dem Beweis durchzuführen.

Betrachten wir als Veranschaulichung eine Kommunikation zwischen einem Beweiser Alice und einem Verifizierer Bob. In diesem Dialog spricht nur Alice, während Bob zuhört oder Fragen stellt. Es ist klar, dass Alice durch Bobs Fragen kein Wissen aus der Konversation erhält. Auf der anderen Seite kann Bob Wissen erlangen – das hängt davon ab, was Alice ihm erzählt. Nehmen wir an, Bob stellt Alice Fragen zu zwei öffentlich bekannten Graphen. Fragt Bob, ob beide Graphen die gleiche Anzahl Knoten haben, so erfährt er durch Alice Antwort kein neues Wissen, da er die Antwort auch leicht hätte selbst bestimmen können. Will Bob aber wissen, ob beide Graphen isomorph sind, d.h. durch ein Umbenennen der Knoten ineinander überführt werden können, bringt ihm eine Antwort von Alice neues Wissen. Denn bisher ist kein effizientes Verfahren bekannt, mit dem Bob diese Antwort – also eine Unterscheidung von Isomorphismus und Nicht-Isomorphismus – selbst berechnen kann. Dass Alice in der Lage ist, diese eigentlich nicht durchführbaren Berechnungen vorzunehmen, liegt daran, dass sie als Beweiser über Zeugen bzw. Geheimnisse verfügt, die es ihr ermöglichen, solch schwierige Berechnungen effizient durchzuführen.

Um auch formal zeigen zu können, dass in einem Protokoll kein Wissen übertragen wird und es damit die Zero-Knowledge-Eigenschaft besitzt, beweist man, dass ein Verifizierer seine Sicht des Beweises – also alle ausgetauschten Daten – selbst erzeugt haben könnte. Dies entspricht der Erstellung eines Videos aus dem einführenden Beispiel in 1.3. Verallgemeinert beschreibt man für diesen Nachweis einen *Simulator* $\mathcal{M}$, der nur mit Kenntnis der öffentlichen Information x versucht einen korrekten Beweisablauf zu erzeugen, so dass er nicht von einem Originalbeweis unterschieden werden kann. Die Idee die hinter diesem Simulations-Konzept steht, ist folgende:

Wenn man keine geheime Information in die Simulation eines Beweistranskriptes hineinstecken muss und es nicht von der Aufzeichnung einer tatsächlichen stattgefundenen Kommunikation unterschieden werden kann, kann man auch keine Information aus solch einem Transkript herausholen.

Definition 3 (Transkript)
Ein Transkript view(x) beschreibt die Folge der während der Protokollausführung ausgetauschten Daten und hängt nur von der öffentlichen Information x und den von Beweiser und Verifizierer erzeugten Zufallsbits ab.

Definition 4 (Simulator)
Ein Simulator $\mathcal{M}$ ist ein in polynomialer Zeit arbeitender Algorithmus, der nur die zu beweisende Aussage x als Eingabe hat und bei der Protokollausführung den Beweiser simuliert. Das durch $\mathcal{M}$ erzeugte Transkript wird $\mathcal{M}(x)$ bezeichnet.

Gelingt es also, solch einen Simulator für einen interaktiven Beweis $(\mathcal{P}, \mathcal{V})$ anzugeben, besitzt $(\mathcal{P}, \mathcal{V})$ die Zero-Knowledge-Eigenschaft.

Definition 5 (Zero-Knowledge-Eigenschaft)
Ein interaktiver Beweis $(\mathcal{P}, \mathcal{V})$ besitzt die Zero-Knowledge-Eigenschaft, wenn es einen in polynomialer Zeit laufenden Simulator gibt, der ein Kommunikationstranskript erzeugt, das von einer tatsächlich stattgefundenen Kommunikation nicht unterschieden werden kann.

Das bedeutet, die simulierten und echten Transkripte besitzen eine ununterscheidbare Wahrscheinlichkeitsverteilung. Je nach dem Grad der Ununterscheidbarkeit unterteilt man Zero-Knowledge-Protokolle in 3 Sicherheitsstufen:

- **Perfekt Zero-Knowledge**
  Steht für *informationstheoretisch* sichere Zero-Knowledge-Verfahren. Die realen Kommunikationstranskripte und die simulierten Transkripte haben exakt die gleiche Verteilung.
- **Statistisch Zero-Knowledge**
  Der Unterschied zwischen den Wahrscheinlichkeitsverteilungen ist vernachlässigbar gering.
- **Berechenbar Zero-Knowledge**
  Hierbei ist nicht mehr gefordert, dass beide Verteilungen identisch sind, sondern nur, dass niemand die Transkripte in polynomialer Zeit unterscheiden kann. Dies entspricht dem Gedanken *komplexitätstheoretischer* Kryptographie.

Mit dieser Simulierbarkeit wird also gezeigt, dass ein Beweistranskript keine Information über das Geheimnis enthält – nicht einmal, ob es überhaupt verwendet wurde.

Wie solch ein Simulator konstruiert werden kann, wird im Ende des nächsten Abschnittes erklärt. Zunächst steht der generelle Ablauf eines Zero-Knowledge-Beweises im Vordergrund.

3.2 Genereller Beweisablauf

Da es sich bei Zero-Knowledge-Beweisen um spezielle interaktive Beweise handelt, kommen hier die im vorigen Kapitel beschriebenen Elemente zum Einsatz. Zusammengefasst gehören zu solch einem interaktiven Beweis ein Beweiser, ein Verifizierer, sowie ein Geheimnis, das es nachzuweisen gilt:

Beweiser $\mathcal{P}$
$\mathcal{P}$ will einen Verifizierer überzeugen, dass er eine bestimmte Information oder Wissen besitzt. Dabei will $\mathcal{P}$ aber nichts über sein Geheimnis verraten.

Verifizierer $\mathcal{V}$
$\mathcal{V}$ stellt dem Beweiser $\mathcal{P}$ eine Reihe von Fragen, die ihm helfen zu entscheiden, ob $\mathcal{P}$ tatsächlich das weiß, was er behauptet zu wissen.

Geheimnis w
Das Geheimnis ist eine Information und kann z.B. ein Passwort, ein geheimer Schlüssel eines Public Key Kryptosystems oder die Lösung zu einem mathematischen Problem sein.

Der Beweisablauf zwischen $\mathcal{P}$ und $\mathcal{V}$ besteht aus mehreren *Beweisrunden*, wobei jede Runde in der Regel in drei Schritten abläuft. Im ersten Schritt wählt $\mathcal{P}$ einen Zufallswert aus einer zuvor festgelegten Menge und sendet ein Commitment (a) dieses Wertes an $\mathcal{V}$. Im nächsten Schritt wählt $\mathcal{V}$ aus einer Menge möglicher Fragen eine – die sogenannte Challenge (c) – aus und stellt sie $\mathcal{P}$. Der Verifizierer $\mathcal{V}$ wird im dritten Schritt genau dann akzeptieren, wenn $\mathcal{P}$ die richtige Antwort (z) auf diese Frage geben kann. Mithilfe seines Geheimnisses gelingt es $\mathcal{P}$ stets die korrekten Antworten zu liefern, so dass $\mathcal{V}$ jede Beweisrunde (a, c, z) und damit auch den gesamten Beweis akzeptieren wird. Das Einbringen der Zufallsgrößen in Schritt eins und zwei dient der Unvorhersagbarkeit der Antworten und unterscheidet die einzelnen Beweisrunden voneinander.

Schritt 1:	$\mathcal{P} \rightarrow \mathcal{V}$	Commitment a
Schritt 2:	$\mathcal{P} \leftarrow \mathcal{V}$	Challenge c
Schritt 3:	$\mathcal{P} \rightarrow \mathcal{V}$	Response z

Abbildung 5: Ablauf einer Beweisrunde

Ein betrügender Beweiser $\mathcal{P}^*$, der einen Verifizierer von einer falschen Behauptung überzeugen will, kann in solch einem Beweis genau dann korrekte Antworten liefern, wenn er im Voraus richtig errät, welche Challenge ihm $\mathcal{V}$ senden will. Dies gelingt ihm bei einer 1-bit Challenge mit der Wahrscheinlichkeit $\frac{1}{2}$, so dass sich der Verifizierer auch nur zu 50% sicher sein kann, dass der Beweiser das Geheimnis tatsächlich kennt. Durch Wiederholungen der Protokollrunden sinkt jedoch die Wahrscheinlichkeit, dass ein betrügender Beweiser die Challenges in jedem Durchlauf richtig geraten hat. In Abhängigkeit der Rundenzahl n des Beweises lässt sich somit die Sicherheit bis auf einen gewünschten Wert $1 - 2^{-n}$ steigern. Zudem wird das komplette Protokoll bei nur einer falschen Antwort des Beweisers abgebrochen, d.h. der Verifizierer ist bereits bei einer falschen Antwort von $\mathcal{P}^*$ überzeugt, dass seine Behauptung falsch ist und er das Geheimnis nicht kennt.

3.3 Black-Box Simulator vs. Honest-Verifier Simulator

Um zu zeigen, dass ein solcher Beweis die Zero-Knowledge-Eigenschaft besitzt, muss ein Simulator angegeben werden, der Transkripte, d.h. Folgen von Beweisrunden (a, c, z) erzeugt, die von Transkripten einer tatsächlich stattgefundenen Kommunikation nicht unterschieden werden können. Der Simulator spielt dabei einen Beweis nach dem selben Prinzip wie ein betrügender Beweiser durch und nimmt nur die Runden in das Transkript auf, in denen er die Challenge richtig geraten hat.

Wenn man annehmen kann, dass sich der Verifizierer $\mathcal{V}$ immer ehrlich verhält, wird er seine Challenges stets zufällig wählen, so dass die Challenges genausogut vom Simulator $\mathcal{M}_{\mathcal{V}}$ selbst erzeugt werden können. Der Simulator ist damit in der Lage, ausschließlich

korrekte Beweisrunden zu erzeugen, denn durch die Kenntnis der Challenge kann er seine Festlegungen a so wählen, dass er im 3. Schritt die für (a, c) richtige Antwort z liefern kann. Mit der Konstruktion eines solchen Simulators $\mathcal{M}_{\mathcal{V}}$ wird die sogenannte *Honest-Verifier Zero-Knowledge* Eigenschaft nachgewiesen.

Ein betrügender Verifizierer $\mathcal{V}^*$, der versucht etwas über das Geheimnis zu erfahren, könnte dagegen seine Fragen nach einer bestimmten Strategie wählen und sie z.B. in Abhängigkeit der Festlegungen von $\mathcal{P}$ im ersten Schritt bilden. Die Wahrscheinlichkeitsverteilung von Transkripten mit Beteiligung von $\mathcal{V}^*$ wäre dann nicht mehr ununterscheidbar von einem Honest-Verifier Simulator $\mathcal{M}_{\mathcal{V}}$ zu simulieren. Im allgemeinen Fall simuliert $\mathcal{M}$ daher nur den Beweiser und kommuniziert mit dem gewünschten Verifizierer. Selbst wenn $\mathcal{V}^*$ versucht zu betrügen, kann so ein gültiges Transkript erstellt werden. Es wird dazu ein globaler Simulator $\mathcal{M}$ konstruiert, der für jeden Verifizierer $\mathcal{V}^*$ eine eigene Simulation erzeugt, in der es $\mathcal{V}^*$ als Black Box in den Prozess mit einbezieht – daher auch der Name *Black-Box Simulation*. Neben einer normalen Kommunikation mit $\mathcal{V}^*$ hat $\mathcal{M}$ zudem die Möglichkeit $\mathcal{V}^*$ zurückzusetzen, falls er nicht die Challenge liefert, die $\mathcal{M}$ erwartet hat. Zurücksetzen bedeutet, dass die letzte fehlgeschlagene Runde aus dem „Gedächtnis“ des Verifizierers gelöscht wird, d.h. er wird in den Zustand vor dieser Beweisrunde zurückversetzt, so dass die falsche Antwort des Simulators keinen Einfluss auf die Strategie von $\mathcal{V}^*$ hat. Das simulierte Transkript setzt sich dann ausschließlich aus den Runden zusammen, in denen $\mathcal{M}$ die Challenge von $\mathcal{V}^*$ korrekt geraten hat.

Abbildung 6 zeigt die beiden Simulator-Typen sowie den generellen Ablauf eines Beweises.

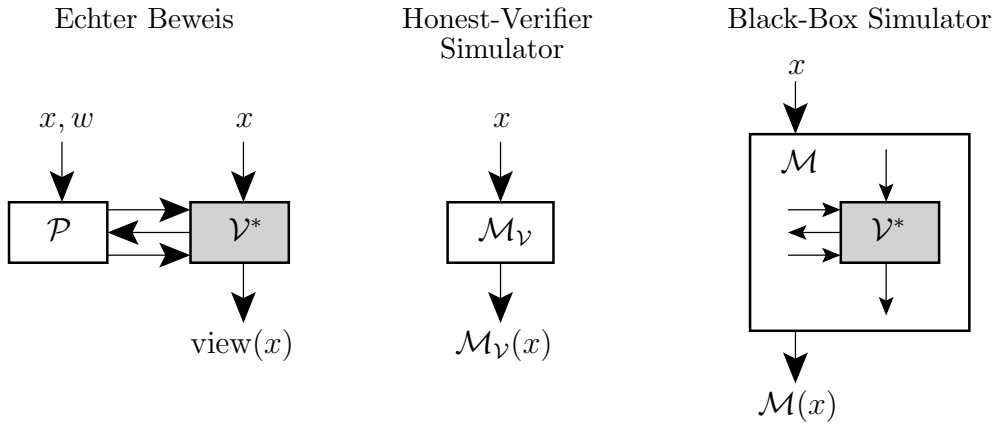

Abbildung 6: Zero-Knowledge-Simulatoren im Vergleich[5]

[5]Darstellung nach[Pfi99]

3.4 Zusammenfassung

Ist ein interaktiver Beweis $(\mathcal{P}, \mathcal{V})$ vollständig und korrekt und konnte zudem mittels eines Simulators die Zero-Knowledge-Eigenschaft nachgewiesen werden, besitzt $(\mathcal{P}, \mathcal{V})$ zusammengefasst folgende Besonderheiten:

Ein Verifizierer lernt nichts durch das Protokoll.
$\mathcal{V}$ erhält durch die Interaktion mit $\mathcal{P}$ kein zusätzliches Wissen, das er nicht selbst aus der öffentlichen Informationen x ableiten könnte. Das ist das zentrale Konzept von Zero-Knowledge, d.h. während des Beweises wird Null Wissen übertragen.

Ein Beweiser, der das Geheimnis nicht kennt, kann einen Verifizierer nicht betrügen.
Wenn $\mathcal{P}^*$ das Geheimnis nicht kennt, kann er $\mathcal{V}$ nur mit vernachlässigbarer Wahrscheinlichkeit betrügen. Die Erfolgschance von $\mathcal{P}^*$, kann dazu so gering wie nötig gehalten werden, indem eine entsprechend hohe Anzahl von Beweisrunden durchgeführt wird.

Ein Verifizierer kann einen Beweiser nicht betrügen.
$\mathcal{V}^*$ erhält selbst dann keine zusätzlichen Informationen aus dem Beweis, wenn er versucht zu betrügen und sich dabei nicht ans Protokoll hält. Die einzige Möglichkeit, die $\mathcal{V}^*$ hat, ist sich selbst zu überzeugen, dass $\mathcal{P}$ das Geheimnis kennt.

Ein Verifizierer kann sich gegenüber Dritten nicht als Beweiser ausgeben.
Da keine Information – außer der Gültigkeit der Behauptung – von $\mathcal{P}$ zu $\mathcal{V}$ übertragen wird, kann sich $\mathcal{V}$ später nicht mit dem Beweistranskript als $\mathcal{P}$ ausgeben. Das Transkript kann sogar nicht einmal von $\mathcal{V}$ verwendet werden um Dritte zu überzeugen, dass $\mathcal{P}$ sein Geheimnis kennt, da der Beweis ebenso gut simuliert sein könnte.

Es besteht allerdings die Gefahr von Man-In-The-Middle-Angriffen, bei denen $\mathcal{V}$ alle Antworten von $\mathcal{P}$ an eine dritte Person weiterleitet und sich ihm gegenüber so als $\mathcal{P}$ ausgeben kann. Dies muss aufgrund der Interaktionen natürlich in Echtzeit geschehen. Man-In-The-Middle-Angriffe sowie eine Möglichkeit zur Verhinderung, werden in Kapitel 7 genauer behandelt.

4 Bekannte Zero-Knowledge-Protokolle

Als ’85 das Konzept der Zero-Knowledge-Verfahren entdeckt wurde, gab es zunächst nur sehr wenige konkrete Beispiele und Protokolle, so dass man befürchtete, dass – ähnlich wie in den Public-Key-Kryptosystemen – nur wenige mathematische Probleme für die Umsetzung dieser Technik in Frage kommen. Nur ein Jahr später konnte allerdings schon das Gegenteil bewiesen werden. Unter der Annahme, dass Bit-Commitment-Verfahren existieren, gelang es Goldreich, Micali und Widgerson einen Zero-Knowledge-Beweis für das

$\mathcal{NP}$-vollständige Problem der 3-Färbbarkeit von Graphen zu entwickeln und damit war bewiesen, dass Zero-Knowledge-Verfahren auf Grundlage aller Probleme in $\mathcal{NP}$ möglich sind [GMW86].

Für drei dieser Probleme werden nun die konkreten Umsetzungen der Zero-Knowledge-Technik betrachtet und die dafür notwendigen Eigenschaften bewiesen. Das Beweissystem für Graphisomorphismus soll dabei zunächst einen Einstieg in die mathematische Umsetzung der Zero-Knowledge-Technik liefern. Im Anschluss werden dann Verfahren, die auf den bekannteren Problemen der quadratischen Reste und diskreten Logarithmen basieren, etwas genauer betrachtet.

4.1 Zero-Knowledge-Beweissystem für Graphisomorphismus

Eines der klassischsten und anschaulichsten Beispiele von Zero-Knowledge-Protokollen ist der '86 von Goldreich, Micali und Widgerson [GMW86] entwickelte Beweis für Graphisomorphie. Dieser basiert auf der Annahme, dass es praktisch unmöglich ist, einen Isomorphismus zwischen zwei hinreichend großen Graphen G_1 und G_2 zu finden. Ein Graph wird dabei durch die Menge seiner Kanten E und Knoten V als $G = (V, E)$ beschrieben.

Definition 6 (Isomorphismus)
*Zwei Graphen $G_1(V_1, E_1)$ und $G_2(V_2, E_2)$ heißen **isomorph**, wenn es eine bijektive Abbildung $\phi : V_1 \to V_2$ gibt, so dass für alle $(v_0, v_1) \in V_1$ gilt*

$$\{v_0, v_1\} \in E_1 \Leftrightarrow \{\phi(v_0), \phi(v_1)\} \in E_2$$

*ϕ heißt dann **Isomorphismus** zwischen G_1 und G_2.*
Die Isomorphiebeziehung ist reflexiv, symmetrisch und transitiv.

Anschaulich bedeutet Isomorphie, dass ein Graph G_1 durch bloßes Umbenennen seiner Knoten in den Graphen G_2 – bzw. umgekehrt – überführt werden kann.

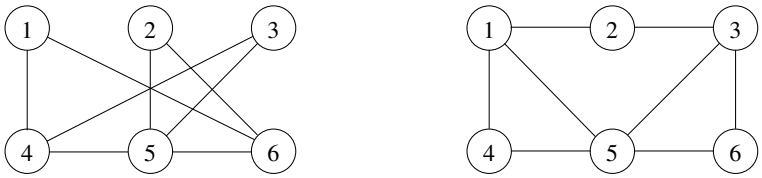

Abbildung 7: Zwei isomorphe Graphen

Die **Graphisomorphie-Annahme** beruht nun darauf, dass bisher kein Algorithmus gefunden wurde, der für große Graphen effizient Isomorphie bzw. Nicht-Isomorphie bestimmen kann. Es bleibt zwar die Möglichkeit, alle Permutation eines Graphen zu testen, aber

bereits bei einer Größe von 12 Knoten müssen dazu $12! = 479001600$ mögliche Umformungen überprüft werden.

Solch ein Isomorphismus ϕ zwischen zwei Graphen bildet in dem Zero-Knowledge-Beweis das Geheimnis des Beweisers $\mathcal{P}$:

Bevor der Beweis beginnen kann, muss $\mathcal{P}$ zunächst das Geheimnis und die zugehörige öffentliche Information erzeugen. Dazu wählt er einen möglichst großen Graphen G_1 und eine zufällige Permutation ϕ, mit der er den Graphen G_2 bildet. Das Paar (G_1, G_2) wird dem Verifizierer $\mathcal{V}$ zugänglich gemacht, den Isomorphismus ϕ hält $\mathcal{P}$ geheim. Ziel des Beweisers ist es dann, den Verifizierer zu überzeugen, den Isomorphismus zwischen G_1 und G_2 zu kennen.

Beweiser und Verifizierer nutzen dazu folgendes Protokoll:

Beweiser $\mathcal{P}$		**Verifizierer $\mathcal{V}$**
geg.: $(G_1, G_2), \phi$		geg.: (G_1, G_2)
(1) wählt zufällige Permutation π und berechnet $H = \pi(G_2)$	$\xrightarrow{H}$	
		(2) wählt $c \in_R \{1, 2\}$
	$\xleftarrow{c}$	
(3) $\sigma = \begin{cases} \pi \cdot \phi & \text{wenn } c = 1 \\ \pi & \text{wenn } c = 2 \end{cases}$	$\xrightarrow{\sigma}$	
		(4) überprüft ob $\sigma(G_c) \stackrel{?}{=} H$

Abbildung 8: Zero-Knowledge-Protokoll für Graphisomorphismus

(1) Der Beweiser $\mathcal{P}$ wählt im ersten Schritt eine weitere Permutation π und wendet diese auf G_2 an. Den zu G_2 isomorphen Ergebnisgraphen $H = \pi(G_2)$ schickt er an $\mathcal{V}$. Sind G_1 und G_2 tatsächlich isomorph, so ist der neue Graph H auch isomorph zu G_1.

(2) Nachdem der Verifizierer H empfangen hat, hat er nun die Wahl sich den Isomorphismus zwischen H und G_1 oder H und G_2 zeigen zu lassen. Dazu wählt er zufällig $c \in \{1, 2\}$ und sendet diese Challenge zurück an $\mathcal{P}$.

(3) $\mathcal{P}$ soll nun den Isomorphismus zwischen G_c und H anhand einer Permutation σ nachweisen. Wenn c den Wert 2 hat, sendet er einfach $\sigma = \pi$ zurück. Sonst muss $\mathcal{P}$ den Isomorphismus zwischen G_1 und H nachweisen und bildet dazu mit Hilfe seines Geheimnisses die Permutation $\sigma = \pi \cdot \phi$.

(4) $\mathcal{V}$ überprüft nun, ob $\sigma(G_c)$ tatsächlich ein Isomorphismus von H ist. In diesem Fall akzeptiert $\mathcal{V}$ die Beweisrunde.

Solch eine Beweisrunde wird n mal wiederholt, bis der Verifizierer ausreichend überzeugt wurde, dass der Beweiser tatsächlich den Isomorphismus ϕ kennt. Um zu zeigen, dass dieses Protokoll auch wirklich ein perfekter Zero-Knowledge-Beweis ist, müssen nun die in Kapitel 3 beschriebenen Eigenschaften nachgewiesen werden.

Nachweis der ***Vollständigkeit*** - Halten sich $\mathcal{P}$ und $\mathcal{V}$ an das Protokoll und kennt $\mathcal{P}$ den Isomorphismus ϕ zwischen G_1 und G_2, wird es ihm immer gelingen, $\mathcal{V}$ zu überzeugen: Denn sind G_1 und G_2 isomorphe Graphen, muss auch der aus G_2 erzeugte Graph H isomorph zu G_1 sein. Mithilfe der Permutation ϕ kann $\mathcal{P}$ daher für jede Challenge immer den gewünschten Isomorphismus liefern.

Nachweis der ***Korrektheit*** - Ein betrügender Beweiser $\mathcal{P}^*$, der ohne Kenntnis des Geheimnisses ϕ einen Verifizierer überzeugen will, den Isomorphismus zwischen G_1 und G_2 zu kennen, soll mit überwältigender Wahrscheinlichkeit scheitern: Ohne Kenntnis des Isomorphismus zwischen G_1 und G_2 ist $\mathcal{P}^*$ immer nur in der Lage, auf eine der zwei Challenge $c \in \{1, 2\}$ zu antworten. $\mathcal{P}^*$ kann dazu am Anfang jeder Runde raten, welchen Isomorphismus $\mathcal{V}$ sehen will und den Graphen H entsprechend aus G_1 oder G_2 bilden. Lag er mit seiner Wahl richtig, kann er den Verifizierer überzeugen, indem er π sendet, hat er jedoch falsch getippt, wird ihn der Verifizierer als Schwindler erkennen. Bei n-facher Ausführung des Beweises sinkt damit die Betrugswahrscheinlichkeit von $\mathcal{P}^*$ auf 2^{-n}, d.h. er kann einen Verifizierer nur mit vernachlässigbarer Wahrscheinlichkeit täuschen.

Nachweis der ***perfekten Zero-Knowledge-Eigenschaft*** - Dieser Nachweis geschieht wie in 3.3 beschrieben mit Hilfe eines Simulators $\mathcal{M}$, der die Kommunikation zwischen $\mathcal{P}$ und $\mathcal{V}$ nachstellt. Der Simulator geht dabei genauso vor wie der Betrüger $\mathcal{P}^*$, und rät zu Beginn jeder Runde eine Challenge $c^{'}$. Ist $c^{'} = 1$ bildet er den Graphen $H = \pi G_1$, für $c^{'} = 2$ bestimmt er $H = \pi G_2$. Den erzeugten Graphen H sendet $\mathcal{M}$ an den Verifizierer und erhält dessen Challenge c. Stimmt c mit der geraten Challenge $c^{'}$ überein, sendet er die Permutation π und der Verifizierer wird akzeptieren. Hat er jedoch eine falsche Challenge geraten, kann er keine gültige Antwort liefern und bricht die Beweisrunde ab. Der Simulator zeichnet nur die korrekten Beweisrunden auf, so dass ein solches Transkript von einem echten Beweistranskript nicht zu unterscheiden ist.

4.2 Zero-Knowledge-Beweissystem für quadratische Reste

Nach dem klassischsten Protokoll folgt nun das bekannteste Zero-Knowledge-Verfahren, dass '86 von Fiat und Shamir [FS87] entwickelt wurde und daher auch unter dem Namen *Fiat-Shamir-Protokoll* bekannt ist. Das Verfahren basiert auf der Annahme, dass es praktisch unmöglich ist Quadratwurzeln in $\mathbb{Z}_n^*$ ohne Kenntnis der Primfaktoren von n effizient zu berechnen[6]. Es konnte gezeigt werden, dass dieses Problem genauso schwer zu lösen ist, wie die Berechnung der Primfaktoren p, q für ein gegebenes n.

[6]Die Ähnlichkeit mit dem RSA-Verfahren ist nicht zufällig, denn Shamir war einer der drei Entdecker dieses Kryptosystems.

Im Fiat-Shamir-Verfahren ist das Geheimnis w des Beweisers $\mathcal{P}$ solch eine Quadratwurzel aus $\mathbb{Z}_n^*$, der öffentliche Schlüssel bzw. das Identitätsmerkmal von $\mathcal{P}$ ist der zugehörige quadratische Rest $x = w^2 \bmod n$, mit dem später die Kenntnis von w verifiziert werden kann. Dazu muss aber zunächst der Modulus n feststehen, dieser kann vom Beweiser erzeugt oder von einer Trusted Third Party den beiden Teilnehmern $\mathcal{P}, \mathcal{V}$ geliefert werden. Wichtig ist dabei nur, dass n das Produkt zweier verschiedener Primzahlen ist, wobei die Faktoren p, q nicht öffentlich zugänglich sind. Das eigentliche Geheimnis w wählt jeder Beweiser dann zufällig in $\mathbb{Z}_n^*$ und veröffentlicht den quadratischen Rest x und n als seinen öffentlichen Schlüssel.

Schlüsselgenerierung

Sei $n = p \cdot q$ das Produkt zweier zufälliger, verschiedener Primzahlen.

Der Beweiser $\mathcal{P}$ wählt $w \in_R \mathbb{Z}_n^*$ und berechnet $x = w^2 \bmod n$. Die Werte (n, x) werden als öffentlicher Schlüssel veröffentlicht, w bildet das Geheimnis von $\mathcal{P}$.

Protokoll

Beweiser		**Verifizierer**
geg.: $(n, x)w$		geg.: (n, x)
wählt s zufällig in $\mathbb{Z}_n^*$		
berechnet $a = s^2 \bmod n$		
	$\xrightarrow{a}$	
		wählt $c \in_R \{0, 1\}$
	$\xleftarrow{c}$	
berechnet $z = s \cdot w^c \bmod n$		
	$\xrightarrow{z}$	
		überprüft ob $z^2 \stackrel{?}{=} a \cdot x^c \bmod n$

Abbildung 9: Zero-Knowledge-Beweissystem für quadratische Reste

Will $\mathcal{P}$ nun einem Verifizierer beweisen, dass der von ihm vorgelegte öffentliche Schlüssel x tatsächlich ein quadratischer Rest in $\mathbb{Z}_n^*$ ist bzw. er den zugehörigen geheimen Schlüssel w kennt, nutzt er das in Abbildung 9 dargestellte Protokoll. Eine Beweisrunde dieses Protokolls besteht dabei wie üblich aus drei Beweisschritten: Im ersten Schritt wählt der Beweiser $\mathcal{P}$ zufällig $s \in \mathbb{Z}_n^*$ und berechnet das Commitment a dieses Wertes durch $a = s^2 \bmod n$. Das Commitment wird an den Verifizierer $\mathcal{V}$ gesendet, welcher nun eine Challenge $c \in \{0, 1\}$ an den Beweiser liefert. Der Beweiser $\mathcal{P}$ hat im nächsten Schritt die Aufgabe, die Antwort $z = s \cdot w^c \bmod n$ zu berechnen. Hat $\mathcal{V}$ die Challenge $c = 0$ gewählt, muss $\mathcal{P}$ dafür nur die Festlegung aus Schritt eins öffnen, d.h. den Wert $z = s$ senden. Für $c = 1$ benötigt der Beweiser allerdings sein Geheimnis und berechnet $z = s \cdot w \bmod n$. Der Verifizierer überprüft die Korrektheit der gegebenen Antwort, indem er die Gleichung $z^2 \stackrel{?}{=} a \cdot x^c \bmod n$ testet. Ist die Gleichung gültig, akzeptiert $\mathcal{V}$ die Beweisrunde.

AUFGABE 1
Wieso sendet der Verifizierer nicht ständig die Challenge $c = 1$, wenn nur in dem Fall der Beweiser sein Geheimnis w benötigt?

Satz 1 *Das Fiat-Shamir Verfahren ist vollständig, korrekt und besitzt die perfekte Zero-Knowledge-Eigenschaft.*

Vollständigkeit: Halten sich Beweiser $\mathcal{P}$ und Verifizierer $\mathcal{V}$ an das Protokoll, wird $\mathcal{V}$ stets mit überwältigender Wahrscheinlichkeit den Beweis akzeptieren, denn in $\mathbb{Z}_n^*$ gilt:

$$z^2 = (s \cdot w^c)^2 = s^2 \cdot (w^2)^c = a \cdot x^c \pmod{n}$$

Korrektheit: Zunächst ist festzustellen, dass ein betrügender Beweiser $\mathcal{P}^*$ in jeder Runde in der Lage ist, auf eine der zwei Challenges $c = 0$ oder $c = 1$ korrekt zu antworten. Dazu legt er sich zu Beginn jeder Beweisrunde auf eine Challenge c fest und bildet seinen Ausgangswert a in Abhängigkeit von c: Hat $\mathcal{P}^*$ das Bit $c = 0$ geraten, sendet er in der ersten Runde $a = s^2$ und benötigt als Antwort auf die erhoffte Challenge ausschließlich die von ihm gewählte Quadratwurzel s. Für den Fall der Challenge $c = 1$ müsste normalerweise das Geheimnis w in die Berechnung der Antwort einfließen. Will $\mathcal{P}^*$ auch ohne Kenntnis von w eine gültige Beweisrunde für $c = 1$ erzeugen, muss er daher seinen Ausgangswert entsprechend „präparieren“ und bestimmt a folgendermaßen: $a = s^2 \cdot x^{-1}$. Als Antwort z genügt dann erneut der zufällig gewählte Wert s. $\mathcal{V}$ wird diese Runde akzeptieren, da die Gleichung $z^2 = a \cdot x^c \bmod n$ erfüllt ist:

$$z^2 \quad = \quad a \cdot x^c \quad = \quad s^2 x^{-1} \cdot x \quad = \quad s^2 \pmod{n}$$

In beiden Fällen muss sich $\mathcal{P}^*$ aber vor jeder Beweisrunde bereits auf eine Challenge festlegen und nur, wenn diese auch tatsächlich mit der Challenge des Verifizierers übereinstimmt, kann er eine gültige Antwort geben. D.h. er ist pro Runde nur mit der Wahrscheinlichkeit von $\frac{1}{2}$ in der Lage, den Verifizierer zu überzeugen. Will $\mathcal{P}^*$ seine Betrugschancen erhöhen, müsste er mit nicht vernachlässigbarer Wahrscheinlichkeit selbst dann die korrekte Antwort geben können, wenn er eine falsche Challenge geraten hat. Damit wäre er allerdings in der Lage, die Quadratwurzel von x – also das Geheimnis w – selbst zu berechnen. Denn könnte $\mathcal{P}^*$ in einer Runde beide Challenges korrekt beantworten, so muss er ein z_0 mit $z_0^2 = a$ und auch ein z_1 mit $z_1^2 = a \cdot x$ kennen. Wegen

$$\frac{z_1^2}{z_0^2} = \frac{a \cdot x}{a} = x \bmod n$$

liefert ihm $\frac{z_1}{z_0}$ eine gültige Quadratwurzel von x.

Dies widerspricht aber der Annahme, dass es keine effiziente Berechnung von Quadratwurzeln in $\mathbb{Z}_n^*$ gibt. Einem betrügender Beweiser $\mathcal{P}^*$ bleibt damit nur die oben beschriebene Möglichkeit, die Challenges vor jeder Beweisrunde zu raten. Die Betrugswahrscheinlicheit kann daher mit ausreichendem n auf einen exponentiell kleinen Wert 2^{-n} reduziert werden. Die Korrektheit ist somit bewiesen.

Zero-Knowledge-Eigenschaft: Intuitiv ist schnell klar, dass dieser Beweis keine zusätzlichen Informationen über das Geheimnis w offenbart. Die Festlegungen a der ersten Runde sind zufällige quadratische Reste ohne Bezug zu x oder w. In Abhängigkeit der Challenge ist die Antwort z entweder die Quadratwurzel s, die zufällig in $\mathbb{Z}_n^*$ gewählt wurde, oder für den Fall $c = 1$ das Produkt $s \cdot w \bmod n$. Das Geheimnis ist in dem Fall durch die Zufallszahl s maskiert und kann nur mit dem selben Aufwand extrahiert werden, mit dem auch w direkt aus x zu berechnen ist.

Um die Zero-Knowledge-Eigenschaft auch formal beweisen zu können, muss ein Algorithmus angegeben werden, der die Kommunikation zwischen $\mathcal{P}$ und $\mathcal{V}$ ohne die Kenntnis von w mit solch einer Wahrscheinlichkeitsverteilung simulieren kann, dass sie nicht von einer echten Kommunikation unterschieden werden kann. Kann man davon ausgehen, dass sich alle Verifizierer stets ehrlich verhalten und damit ihre Challenges tatsächlich zufällig wählen, genügt es, einen Simulator für die *Honest-Verifier Zero-Knowledge*-Eigenschaft anzugeben. Der Simulator $\mathcal{M}_\mathcal{V}$ kann in dem Fall die Funktion des Verifizierers übernehmen und selbst die zufälligen Challenges erzeugen:

Algorithmus für Simulator $\mathcal{M}_\mathcal{V}$:

Input: n, x

for i:=1 to k **do**

Wähle zufällig $c \in \{0, 1\}$

Wähle zufällig $z \in \mathbb{Z}_n^*$

Berechne $a = z^2 \cdot x^{-c} \bmod n$

Veröffentliche (a, c, z)

end

Kann aber nicht ausgeschlossen werden, dass sich ein Verifizierer $\mathcal{V}^*$ betrügerisch verhält, genügt diese Simulation nicht. Erzeugt ein Verifizierer die Challenges nach einer bestimmten Vorschrift – z.B. in Abhängigkeit der Festlegung a – wäre die tatsächliche Kommunikation nicht mehr ununterscheidbar durch $\mathcal{M}_\mathcal{V}$ zu simulieren, da dieser den Wert a erst im letzten Schritt berechnet. Transkripte mit einer Wahrscheinlichkeitsverteilung entsprechend der Strategie eines betrügenden Verifizierers $\mathcal{V}^*$, lassen sich durch einen *Black-Box-Simulator* erzeugen. Für das Fiat-Shamir-Verfahren ergibt sich dazu folgender Algorithmus:

Algorithmus für Simulator $\mathcal{M}$:

Input: n, x

for i:=1 to k **do**

repeat

Wähle zufällig $c^{'} \in \{0,1\}$

Wähle zufällig $z \in \mathbb{Z}_n^*$ und berechne $a = z^2 \cdot x^{-c'} \bmod n$

Rufe $\mathcal{V}^*$ mit Eingabe a auf und erhalte Challenge $c \in \{0,1\}$

if $c = c^{'}$ sende z an $\mathcal{V}^*$ und veröffentliche (a, c, z)

else setze $\mathcal{V}^*$ zurück

until $c = c^{'}$

end

Das Fiat-Shamir-Verfahren gilt aufgrund seiner Effizienz für praktische Anwendungen als wichtigstes Zero-Knowledge-Verfahren und eignet sich im Gegensatz zu Public-Key Verfahren wie z.B. RSA auch für Implementierungen auf SmartCards. Damit kam es für den Einsatz in Pay-TV Systemen in Frage und wird seit der Patentierung '91 unter dem Namen „videocrypt“ [vid91] auf Decoderkarten verwendet.

4.3 Zero-Knowledge-Beweissystem für diskrete Logarithmen

Eine weitere beliebte Einwegfunktion in der Kryptographie ist die diskrete Exponentialfunktion, d.h die Berechnung von $b = a^x \bmod n$. Diese kann in allen Gruppen in polynomialer Zeit durchgeführt werden, für die Umkehrfunktion, also das Berechnen des diskreten Logarithmus x mit $a^x = b \bmod n$, gibt es in bestimmten Gruppen jedoch keine effizienten Algorithmen. Dazu gehören auch multiplikative Gruppen mit primer Ordnung. Die Komplexität des Findens eines diskreten Logarithmus in solch einer Gruppe ist vergleichbar mit dem Faktorisieren einer ähnlich großen Zahl n [Sch98]. Auf Basis dieser Annahme existieren mittlerweile eine Vielzahl von Zero-Knowledge-Verfahren, die ersten Protokolle zum Nachweis der Kenntnis eines diskreten Logarithmus wurden bereits '87 von Chaum et.al. in [CEGP87, CEG87] präsentiert.

Eine Möglichkeit, die für solche Verfahren notwendige multiplikative Gruppe mit primer Ordnung zu erzeugen, beginnt mit der Wahl zweier Primzahlen p, q wobei q ein Teiler von $(p-1)$ zu sein hat. Ein Generator wird daraufhin mit folgendem Algorithmus ermittelt [MVO96]:

Algorithmus zum Finden eines Generators G_q:

Input: die zyklische Gruppe $\mathbb{Z}_p^*$, $p-1 = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$

Output: ein Generator g für die zyklische Gruppe $\mathbb{Z}_p^*$

1. Wähle zufällig ein Element a aus $\mathbb{Z}_p^*$

2. **for** $i := 1$ to k **do**

   Berechne $b := a^{\frac{p-1}{p_i}}$

   Falls $b = 1$ gehe zu Schritt 1.

3. Return $g = a$

g heißt dabei Generator der eindeutigen Untergruppe G_q und wird gemeinsam mit den Primzahlen p und q veröffentlicht. Diese Werte werden in der Regel von einer Trusted Third Party geliefert und können von einer ganzen Gruppe von Benutzern verwendet werden.

Jeder Benutzer kann dann sein eigenes Geheimnis w aus $\mathbb{Z}_q$ wählen und das zugehörige Untergruppenelement $x = g^w \bmod p$ veröffentlichen.

Mithilfe des Zero-Knowledge-Protokolls kann anschließend ein Beweiser einen Verifizierer überzeugen, dass er den diskreten Logarithmus w von x kennt, ohne etwas über w verraten zu müssen:

Satz 2 *Das Zero-Knowledge-Protokoll zum Nachweis der Kenntnis eines diskreten Logarithmus ist vollständig, korrekt und besitzt die perfekte Zero-Knowledge-Eigenschaft.*

Vollständigkeit: Halten sich Beweiser $\mathcal{P}$ und Verifizierer $\mathcal{V}$ an das Protokoll und kennt $\mathcal{P}$ tatsächlich das Geheimnis w, so wird es ihm immer gelingen, $\mathcal{V}$ zu überzeugen. Denn in $\mathbb{Z}_p^*$ gilt:

$$g^z = g^{s+cw \bmod q} = g^s \cdot (g^w)^c = a \cdot x^c \pmod p$$

Korrektheit: Auch hier kann ein betrügender Beweiser $\mathcal{P}^*$ in jeder Runde auf eine der zwei Challenges $c = 0$ oder $c = 1$ korrekt antworten, indem er sich zu Beginn jeder Beweisrunde auf eine Challenge c festlegt und seinen Ausgangswert a abhängig von c bildet. Hat $\mathcal{P}^*$ das Bit $c = 0$ geraten, sendet er die Festlegung $a = g^s$ und benötigt als Antwort ausschließlich den von ihm gewählten Logarithmus s. Für den Fall der Challenge $c = 1$, wählt $\mathcal{P}^*$ zunächst wie gewohnt $s \in_R \mathbb{Z}_q$, berechnet aber $a = g^s \cdot x^{-1}$. Als Antwort genügt dann erneut der ihm bekannte Logarithmus s. Der Verifizierer wird diese Runde akzeptieren, da die Gleichung $g^z = a \cdot x \pmod p$ erfüllt ist:

$$g^z = a \cdot x = g^s x^{-1} \cdot x = g^s \pmod p$$

In beiden Fällen musste sich $\mathcal{P}^*$ aber vor jeder Beweisrunde bereits auf eine Challenge festlegen und nur, wenn diese auch tatsächlich mit der Challenge des Verifizierers über-

Schlüsselgenerierung

gegebene Parameter: Primzahlen p, q mit $q|(p-1)$, Generator g der Untergruppe G_q

Beweiser wählt w zufällig in $\mathbb{Z}_q$ und erzeugt sein Schlüsselpaar – bestehend aus einem öffentlichen Schlüssel x und einem geheimen Schlüssel w – durch:

$$(x = g^w \bmod p, \quad w)$$

Protokoll

Beweiser $\mathcal{P}$		**Verifizierer $\mathcal{V}$**
geg.: (p, q, g) x, w		geg.: (p, q, g) x
wählt s zufällig in $\mathbb{Z}_q$		
berechnet $a = g^s \bmod p$		
	$\xrightarrow{a}$	
		wählt $c \in_R \{0, 1\}$
	$\xleftarrow{c}$	
berechnet $z = s + c \cdot w \bmod q$		
	$\xrightarrow{z}$	
		überprüft ob $g^z \stackrel{?}{=} a \cdot x^c \bmod p$

Abbildung 10: Zero-Knowledge-Beweissystem für diskrete Logarithmen

einstimmte, konnte er eine gültige Antwort geben. Die Wahrscheinlichkeit, dass er in jeder Iteration das richtige Bit rät und somit einen Verifizierer überzeugen kann, beträgt damit nur 2^{-n}.

Will $\mathcal{P}^*$ seine Betrugschancen erhöhen, müsste er mit nicht vernachlässigbarer Wahrscheinlichkeit selbst dann die korrekte Antwort geben können, wenn er eine falsche Challenge geraten hat. Damit wäre er allerdings in der Lage, den diskreten Logarithmus von x selbst zu berechnen. Ist $\mathcal{P}^*$ fälschlicherweise von $c = 0$ ausgegangen, müsste er nun eine Antwort $\bar{z}$ präsentieren, so dass $g^{\bar{z}} = a \cdot x \bmod p$ gilt. Kann er solch ein $\bar{z}$ berechnen, liefert ihm das den diskreten Logarithmus von x:

$$\begin{aligned} & g^{\bar{z}} && = && a \cdot x = g^s \cdot x && \pmod p \\ \Rightarrow \quad & g^{\bar{z}-s \bmod q} && = && x = g^w && \pmod p \\ \Rightarrow \quad & \bar{z} - s && = && w && \pmod q \end{aligned}$$

Hat $\mathcal{P}^*$ die Challenge $c = 1$ angenommen, jedoch $c = 0$ erhalten, benötigt er nun ein $\bar{z}$ das folgende Gleichung erfüllt: $g^{\bar{z}} = a \bmod p$. Auch dieses $\bar{z}$ führt einen zum Geheimnis w:

$$\begin{aligned} & g^{\bar{z}} && = && a = g^s \cdot x^{-1} && \pmod p \\ \Rightarrow \quad & g^{s-\bar{z} \bmod q} && = && x = g^w && \pmod p \\ \Rightarrow \quad & s - \bar{z} && = && w && \pmod q \end{aligned}$$

Das Problem des Berechnens von $\bar{z}$ ist damit äquivalent zum Berechnen des diskreten Logarithmus in G_q, d.h. die Betrugswahrscheinlichkeit von 2^{-n} kann nicht gesteigert werden.

Zero-Knowledge-Eigenschaft: Ähnlich wie im Beweissystem für quadratische Reste ist hier schnell deutlich, dass während des Protokolls keine zusätzlichen Informationen über das Geheimnis w offenbart werden. Die Festlegungen a sind zufällige und gleichverteilte Werte der durch g erzeugten Untergruppe G_q ohne Bezug zu x oder w. In Abhängigkeit der Challenge ist die Antwort z entweder der Logarithmus s, der zufällig in $\mathbb{Z}_q$ gewählt wurde, oder für den Fall $c = 1$ die Summe $s + w \bmod q$. Das Geheimnis ist in dem Fall durch die Zufallszahl s maskiert und könnte nur durch das Berechnen des diskreten Logarithmus von a extrahiert werden.

Für den formalen Nachweis der Zero-Knowledge-Eigenschaft lässt sich folgender in polynomialer Zeit endender Simulator angeben:

Algorithmus für Simulator $\mathcal{M}$:

Input: $(p, q, g), x$

for i:=1 to k **do**

repeat

Wähle zufällig $c^{'} \in \{0, 1\}$

Wähle zufällig $z \in \mathbb{Z}_q$ und berechne $a = g^z \cdot x^{-c^{'}} \bmod p$

Rufe $\mathcal{V}^*$ mit Eingabe a auf und erhalte Challenge $c \in \{0, 1\}$

if $c = c^{'}$ sende z an $\mathcal{V}^*$ und veröffentliche (a, c, z)

else setze $\mathcal{V}^*$ zurück

until $c = c^{'}$

end

Eine sehr bekannte Variante des Zero-Knowledge-Beweissystems für diskrete Logarithmen ist das *Schnorr-Protokoll* [Sch89]. Dabei wird vom Verifizierer statt der 1-Bit Challenge eine Challenge $c \in \mathbb{Z}_q$ gewählt. Dies beschleunigt das Verfahren, da nun die Wahrscheinlichkeit, dass ein betrügender Beweiser die korrekte Challenge rät, pro Runde mit $\frac{1}{q}$ deutlich geringer ist. Um einen betrügenden Beweiser auszuschließen, sind somit weniger Beweisrunden nötig, allerdings führt diese Veränderung auch zum Verlust der generellen Zero-Knowledge-Eigenschaft. Denn ein Simulator ist nun ebenfalls nicht mehr in der Lage, in polynomialer Zeit ununterscheidbare Transkripte zu erzeugen, dies gelingt nur, wenn er die Challenge selbst wählen darf. Das Schnorr-Protokoll besitzt also nur die Honest-Verifier Zero-Knowledge-Eigenschaft. Es ist dennoch aufgrund der Effizienz ein sehr populäres Verfahren, und findet insbesondere auf SmartCards Anwendung.

Sollen Zero-Knowledge-Verfahren beschleunigt werden, haben die dazu notwendigen Änderungen meist den Verlust der generellen Zero-Knowledge-Eigenschaft zur Folge. Verallgemeinert stellen diese Änderungen einen Teil der Probleme dar, die nun im folgenden Kapitel bei der Komposition von Zero-Knowledge-Protokollen betrachtet werden.

AUFGABE 2

Die Parameter für ein Zero-Knowledge-Beweissystem zum Nachweis der Kenntnis eines diskreten Logarithmus sind mit ($p = 137$, $q = 17$, $g = 74$) gegeben.

(a) Ein Beweiser $\mathcal{P}$ wählt als seinen geheimen Schlüssel $w = 14$, und berechnet den öffentlichen Schlüssel $x = g^w \bmod p = 56$.

Für den anschließenden Beweis wählt $\mathcal{P}$ zu Beginn einer Beweisrunde $s = 10$ und sendet $a = g^s \bmod p = 72$ an den Verifizierer.

Welche Antwort müsste der Beweiser senden, wenn $\mathcal{V}$ die Challenge $c = 0$ wählt, welche bei $c = 1$?

(b) Wie könnte eine gültige Simulationsrunde für einen Beweis zu dem öffentlichen Schlüssel $x = 56$ aussehen, wenn erwartet wird, dass $\mathcal{V}$ die Challenge $c = 1$ sendet?

(c) Wieviele Beweisrunden sind nötig, damit ein Verifizierer mindestens zu $99,999$ % von der Gültigkeit der Behauptung eines Beweisers überzeugt ist ?

AUFGABE 3

(a) Implementieren Sie die für einen Zero-Knowledge-Beweis der Kenntnis eines diskreten Logarithmus notwendigen Methoden `sendCommitment()`, `sendResponse()` in `Prover.java` und `verify()` in `Verifier.java`.

(b) Vervollständigen Sie die Methode `simulate()` in `Simulator.java`, um für einen Beweis aus Aufgabe 3.a ununterscheidbare Transkripte zu erzeugen.

Zum Testen des Beweises sowie der Simulation kann `ZK_Beweis.java` genutzt werden.

5 Komposition von Zero-Knowledge-Verfahren

Ein Hauptanwendungsgebiet von Zero-Knowledge-Verfahren ist die Verwendung als Subprotokoll in umfangreicheren kryptographischen Verfahren. Eine grundlegende Frage für die Anwendbarkeit ist daher, ob die Komposition – also die beliebige Zusammensetzung von mehreren Zero-Knowledge-Protokollen zu einem größeren Gesamtprotokoll – wieder ein Verfahren ergibt, welches die Zero-Knowledge-Eigenschaft besitzt. Im Gegensatz zu einer einfachen Ausführung muss in solch einer Zusammensetzung ein stärkeres Angreifermodell berücksichtigt werden, da ein bzw. mehrere betrügende Verifizierer nun einzelne Nachrichten aus einem Protokollablauf in einer anderen Protokollinstanz einsetzen können, um so etwas über das Geheimnis zu erfahren. Es ist also zu untersuchen, ob die Zero-Knowledge-Eigenschaft selbst mehreren koordiniert angreifenden Verifizierern standhalten kann.

Als Komposition ergeben sich dabei zwei Möglichkeiten: parallele und konkurrente Komposition. Wird ein Protokoll mehrfach zeitgleich gestartet, spricht man von **paralleler Komposition**, ein asynchroner Ablauf mehrerer Instanzen wird **konkurrente Komposition** bezeichnet.

5.1 Parallele Komposition

Durch die sequentielle Wiederholung der Beweisrunden innerhalb eines Zero-Knowledge-Beweises, kann die Erfolgschance eines betrügenden Beweisers auf einen exponentiell kleinen Wert begrenzt werden. Um das Verfahren zu beschleunigen, wäre es jedoch wünschenswert, die Wiederholungen parallel durchführen zu können, um so die Anzahl der Interaktionen auf ein Minimum zu reduzieren. In solch einer parallelen Ausführung sendet der Beweiser mehrere Festlegungen $(a_1, \ldots a_k)$ gleichzeitig an den Verifizierer, dieser übermittelt daraufhin die entsprechende Anzahl an Challenges $(c_1, \ldots c_k)$ und der Beweiser antwortet im nächsten Schritt auf alle Herausforderungen mit $(z_1, \ldots z_k)$.

Parallele Komposition bedeutet aber auch, dass ein Protokoll zur gleichen Zeit mehrfach abläuft. Dies geschieht z.B. in Multi-Party Protokollen, in denen ein Beweiser eine bestimmte Behauptung mehreren Verifizierern gleichzeitig beweisen will. Auch in solch einem Szenario soll natürlich das Geheimnis des Beweisers durch die Zero-Knowledge-Eigenschaft geschützt bleiben.

Intuitiv scheint solch eine parallele Ausführung wieder ein Zero-Knowledge-Verfahren zu ergeben. Es ist schwer vorzustellen, wie in einem parallelen Verfahren ein Beweiser plötzlich falsche Behauptungen beweisen kann, oder ein betrügender Verifizierer neue Informationen über das Geheimnis erfährt. Und tatsächlich ist diese Intuition korrekt, so lange sich die Verifizierer ehrlich verhalten und ihre Challenges unabhängig von $\mathcal{P}$'s Nachrichten wählen. Können aber betrügende Verifizierer nicht ausgeschlossen werden, ist die Zero-Knowledge-Eigenschaft nicht mehr gewährleistet. Die Ursache liegt in der Simulations-Forderung, die nicht mehr erfüllt werden kann, denn um eine gültige Beweisrunde zu simulieren, müssen nun sämtliche Challenges $c_1 \ldots c_k$ korrekt geraten werden. Da dies jedoch nur mit der Wahrscheinlichkeit von 2^{-k} gelingt, ist es in polynomialer Zeit nicht mehr möglich, ein gültiges Transkript zu erzeugen [FS90].

Damit scheitern parallele Versionen aus rein technischen Gründen an der Zero-Knowledge-Definition und können somit in Umgebungen mit potentiell betrügenden Verifizierer generell nicht eingesetzt werden. Honest-Verifier Zero-Knowledge-Verfahren bleiben hingegen auch bei paralleler Komposition simulierbar sicher und werden daher als Subprotokolle bevorzugt eingesetzt.

5.2 Konkurrente Komposition

Die Betrachtung von konkurrenter Komposition ist für Anwendungen der Zero-Knowledge-Verfahren in offenen Netzwerken wie dem Internet notwendig, da in solch einer Umgebung mehrere Instanzen eines Protokolls ohne zeitliche Kopplung durchgeführt werden können. Solch eine asynchrone Ausführung stellt an sich kein Problem dar, halten sich alle Teilnehmer an das Protokoll wird jede Instanz korrekt ablaufen und die Zero-Knowledge-Eigenschaft ist gewährleistet. Das Risiko besteht in einer Zusammenarbeit mehrerer Verifizierer bzw. einem mächtigen Angreifer, der eine Gruppe von Verifizierern kontrollieren kann. Dadurch können einzelne Nachrichten einer Protokollinstanz in einer anderen Kommunikation eingesetzt werden, um so die nicht polynomial zeitbeschränkte Rechenkraft des Beweisers auszunutzen. Die Verifizierer können dazu jedes Protokoll so lange anhalten, bis sie in einem anderen Protokoll eventuell nutzbare Nachrichten erhalten.

Dwork, Noar und Sahai zeigten in ihrer Arbeit [DNS98], dass die meisten Zero-Knowledge-Verfahren solch einer koordinierten Durchführung nicht standhalten und ihre Zero-Knowledge-Eigenschaft verlieren. Um diese Eigenschaft wieder herzustellen und damit die Anwendung von Zero-Knowledge-Verfahren auch im Internet zu ermöglichen, entwarfen sie eine Lösung die den Parameter Zeit in die Protokolle einbringt. Diese Idee basiert auf genauen Systemuhren der jeweiligen Protokollpartner und schränkt durch Verzögerungen α und Zeitschranken β die Angriffsmöglichkeiten der Verifizierer ein. Hat $\mathcal{P}$ eine Nachricht an einen Verifizierer gesendet, so darf dieser frühestens nach α Sekunden antworten, benötigt er jedoch mehr als β Sekunden bricht $\mathcal{P}$ das Protokoll ab. Durch ein enges Zeitfenster ist es dann einem betrügenden Verifizierer nicht mehr möglich, Nachrichten aus einem Protokollablauf in einer anderen Instanz zu verwenden, ohne das $\mathcal{P}$ mindestens eines der Protokolle beendet. Dwork, Soar und Sahai entwickelten mit dieser Methode mehrere Protokolle, für die sie einen Simulator angeben und damit die Zero-Knowledge-Eigenschaft beweisen konnten.

6 Nicht-interaktive Zero-Knowledge-Verfahren

Die bisher vorgestellten Zero-Knowledge-Protokolle basieren alle auf der Interaktion der Teilnehmer sowie der Unvorhersagbarkeit der Challenges vom Verifizierer. Mit mehrfacher Ausführung der Protokolle konnte dabei die Sicherheit auf den gewünschten Wert angenähert werden. Die Abhängigkeit der Sicherheit von den Interaktionen ist allerdings ein großer Nachteil der Zero-Knowledge-Verfahren: Zum einen ist die Kommunikation im Verhältnis zu den notwendigen Rechenoperationen sehr zeitaufwändig und schränkt daher insbesondere die Verwendung als Subprotokoll ein. Zum anderen stellt es hohe Anforderungen an die Ausführungsumgebung sowie die Teilnehmer, denn Beweiser und Verifizierer müssen während der gesamten Durchführung des Beweises miteinander verbunden sein. Die Möglichkeit, Zero-Knowledge-Verfahren mit schwächeren Anforderungen zu entwerfen, würde daher die Anwendbarkeit solcher Systeme stark erhöhen.

Zur Motivation von nicht-interaktiven Zero-Knowledge-Beweisen wurde in [BFM88] folgendes Szenario beschrieben, das mit den bisherigen Verfahren nicht zu lösen ist:

Seien A und B zwei Mathematiker, wobei sich A auf der Suche nach Inspiration auf einer langen Weltreise befindet. Dabei führt A seine mathematischen Untersuchungen fort und sollte, sobald er einen Beweis für ein neues Theorem entdeckt hat, in der Lage sein, B eine Postkarte zu schreiben und ihm darin die Gültigkeit seiner Behauptung mittels eines Zero-Knowledge-Verfahrens zu beweisen.

Nicht-interaktive Zero-Knowledge-Verfahren lösen diese Probleme, indem sie die notwendigen Interaktionen auf eine Runde reduzieren, d.h. es findet nur noch eine monodirektionale Kommunikation vom Beweiser zum Verifizierer statt. Die gemeinsame Annahme all solcher Verfahren besteht darin, dass allen Beteiligten ein gemeinsamer String σ zur Verfügung steht und die Challenges des Verifizierers ersetzt [BFM88, BSMP91].

Die bekannteste Möglichkeit zur Erzeugung solch eines Wertes σ ist die Verwendung einer Hashfunktion, welche die Rolle des Verifizierers einnimmt und über den Hashwert die notwendigen Challenges liefert. Dieses Vorgehen ist unter dem Namen *Fiat-Shamir Heuristik* bekannt und wurde in [FS87], mit dem Ziel, das Fiat-Shamir Identifikationsverfahren in ein nicht-interaktives Signaturschema zu überführen, entwickelt.

6.1 Fiat-Shamir Heuristik

Mithilfe der Fiat-Shamir Heuristik lassen sich alle interaktiven 3-Runden Zero-Knowledge-Verfahren in ein nicht-interaktives Beweissystem umwandeln. Die Grundidee ist dabei zunächst, den interaktiven Zero-Knowledge-Beweis in seine parallele Ausführung umzuwandeln und anschließend die Rolle des Verifizierers durch eine Hashfunktion zu ersetzten. Da durch diese Ersetzung der Verifizierer keinen Einfluss mehr auf den Beweis hat, ist keine Unterscheidung von Honest-Verifier und Black-Box Zero-Knowledge mehr nötig, so dass die Zero-Knowledge-Eigenschaft trotz der parallelen Ausführung der Protokolle erhalten bleibt.

In einem durch diese Heuristik entstandenen nicht-interaktiven Zero-Knowledge-Verfahren, legt sich der Beweiser zu Beginn nun auf k Zufallswerte fest – je mehr, desto sicherer ist der Beweis – und verwendet die Festlegungen $a_1, \ldots, a_k$ anschließend als Input der Hashfunktion. Die ersten k Bits des entstandenen Hashwertes liefern die einzelnen Challenges $c_1, \ldots, c_k$. Für jede Festlegung wird dann, entsprechend der zugehörigen Challenge, wie üblich der Antwortwert berechnet. Als Beweis können nun sämtlichen Festlegungen $a_1 \ldots a_k$ und Antworten $z_1 \ldots z_k$ veröffentlicht werden. Will ein Verifizierer die Korrektheit des Beweises überprüfen, ermittelt er zunächst mit Hilfe der Hashfunktion die zugrunde liegenden Challenges. Sind alle Antworten des Beweisers korrekt, akzeptiert er den Beweis.

Die Sicherheit beruht bei diesem Verfahren auf der Schwierigkeit, die Ergebnisse einer Hashfunktion vorherzusagen. Da die Festlegungen des Beweiser als Input der Hashfunktion

verwendet werden, muss sich ein betrügender Beweiser $\mathcal{P}^*$, der das Geheimnis nicht kennt, wie bereits in der interaktiven Version auf die Ausgangswerte festlegen, ohne die einzelnen Challenges zu kennen. Damit ist auch hier seine Chance, eine 1-bit Challenge korrekt zu beantworten jeweils $\frac{1}{2}$, die Wahrscheinlichkeit, einen kompletten Beweis zu fälschen, liegt entsprechend bei 2^{-k}. Während bei interaktiven Verfahren ca. 10 Iterationen ausreichten, um einen Betrug auszuschließen, müssen es hier allerdings deutlich mehr sein. Da der Beweis offline stattfindet, ist er Brute-Force Angriffen ausgesetzt, d.h. $\mathcal{P}^*$ kann solange eine Menge von Ausgangswerten wählen, bis die Hashfunktion genau die Challenges erzeugt, die er beantworten kann. Um diese Möglichkeit praktisch auszuschließen, wird eine Rundenanzahl k von 64 oder 128 vorgeschlagen.

Um die Zero-Knowledge-Eigenschaft für solch ein Verfahren beweisen zu können, wird das '93 von Bellare und Rogaway eingeführte *Random Oracle Model* [BR93] verwendet. Dabei werden Hashfunktionen zu echten Zufallsfunktionen idealisiert, so dass die Hashwerte formal von einem *Random Oracle* geliefert werden. Das Random Oracle verhält sich allerdings auch deterministisch und liefert bei gleichen Aufrufen identische Zufallswerte. Seit seiner Einführung wurde das Random Oracle Model erfolgreich genutzt, um formal die Sicherheit vieler praktischer Protokolle zu analysieren.[7] Für den Nachweis der Zero-Knowledge-Eigenschaft eines nicht-interaktiven Verfahrens im Random Oracle Model wird dem Simulator $\mathcal{M}$ zusätzlich eine starke Eigenschaft zugesprochen. Denn im Gegensatz zu Beweiser und Verifizierer ist $\mathcal{M}$ in der Lage, das Random Oracle selbst zu bestimmen. Dabei kann er sogenannte *trapdoor* Informationen einbauen, mit denen er später die Ausgaben des Oracles manipulieren und einen ehrlichen Beweiser simulieren kann. Für die Zero-Knowledge-Eigenschaft wird dann verlangt, dass ein Simulator $\mathcal{M}$ für ein gegebenes x und selbstgewähltes σ' ein Beweistranskript erzeugt, das von einer tatsächlichen Kommunikation basierend auf (x, σ) nicht unterschieden werden kann.

Die Zero-Knowledge-Eigenschaft dieser Verfahren ist daher allerdings nur im Random Oracle Model beweisbar, jedoch nicht in der „realen“ Welt. Das würde in bestimmten Anwendungen auch der Intuition widersprechen, denn z.B. beim Einsatz solcher Beweise als digitale Signaturen sollte ein Simulator nicht in der Lage sein, ununterscheidbare Signaturen des Beweisers zu erzeugen. Aufgrund der nicht-Simulierbarkeit einer konkret stattgefunden Kommunikation ist damit zwar die formale Zero-Knowledge-Eigenschaft nicht gewährleistet, allerdings erfährt der Verifizierer nichts, was ihm ermöglicht das Geheimnis des Beweisers effizienter zu berechnen als vor dem Beweis. Die Verfahren sind nach dem Wegfall der Zero-Knowledge-Eigenschaft also immer noch sicher im klassischen Sinne, d.h. ein komplexitätstheoretisch beschränkter Angreifer hat nur vernachlässigbare Erfolgsaussichten.

[7] Es gibt allerdings auch Kritiker dieses Models, da bereits gezeigt werden konnte, dass es Kryptosysteme gibt, die zwar beweisbar sicher im Random Oracle Model sind, jedoch für jede Instanziierung des Random Oracles durch Hashfunktionen ihre Sicherheit verlieren [CGH98].

6.2 Digitale Signaturen

Nicht-interaktive Zero-Knowledge-Verfahren sparen nicht nur kostbare Interaktionsrunden, sondern erschließen auch ganz neue Anwendungsgebiete. So wurde in [FS87] die Verwendung von Zero-Knowledge-Verfahren als digitale Signaturen[8] vorgestellt und unter anderem in [BG89] weiterentwickelt. Eine Möglichkeit, Zero-Knowledge-Identifikationsverfahren in ein Signaturschema zu überführen, ist die Anwendung der eben beschriebenen Fiat-Shamir Heuristik. Dabei wird die Hashfunktion für die Erzeugung der Challenges nun zusätzlich auf die zu signierende Nachricht m angewendet, d.h. der Hashwert ist abhängig von der Nachricht und den Festlegungen des Beweisers. So können im Nachhinein weder die Festlegungen noch die Nachricht geändert werden, ohne die Signatur zu zerstören.

Um die Signaturlänge zu verringern, hat es sich dabei durchgesetzt, die Signaturen aus den Challenges $(c_1, \ldots, c_k)$ und den Antworten $(z_1, \ldots, z_k)$ zu bilden. Die Gültigkeit einer Signatur überprüft der Verifizierer, indem er den Beweis „rückwärts“ rechnet, d.h. die zu den Antworten passenden Festlegungen $(a_1, \ldots, a_k)$ bestimmt. Anschließend kann er ebenfalls den Hashwert von $(m||a_1|| \ldots ||a_k)$ berechnen und mit den Challenges $(c_1, \ldots, c_k)$ der Signatur vergleichen.

Wird ein Zero-Knowledge-Beweis der Kenntnis eines diskreten Logarithmus mittels der Fiat-Shamir Heuristik in ein nicht-interaktives Signaturverfahren überführt, ergibt sich folgender Ablauf:

Schlüsselgenerierung:

gegebene Parameter: Primzahlen p, q mit $q|(p-1)$, Generator g der Untergruppe G_q, Einweg-Hashfunktion H

Beweiser wählt $w \in \mathbb{Z}_q$ und veröffentlicht den Schlüssel $x = g^w \bmod p$

Signieren einer Nachricht m:

(1) Beweiser wählt $s_1, \ldots, s_k$ zufällig in $\mathbb{Z}_q$ und berechnet $a_i = g^{s_i} \bmod p$ für $i = 1, \ldots, k$

(2) Beweiser berechnet $H(m||a_1|| \ldots ||a_k)$ und nutzt die ersten k Bits des Hashwertes als Challenges $c_1, \ldots, c_k$

(3) Beweiser berechnet Antworten $z_i = s_i + c_i \cdot w \bmod q$ für $i = 1, \ldots, k$

(4) Beweiser sendet Nachricht m und zugehörige Signatur $(c_1, \ldots, c_k, z_1, \ldots, z_k)$

Verifizieren der Signatur $(c_1, \ldots, c_k, z_1, \ldots, z_k)$ für Nachricht m:

(1) Verifizierer berechnet $b_i = g^{z_i} \cdot x^{-c_i} \bmod p$ für $i = 1, \ldots, k$

(2) Verifizierer ermittelt $H(m||b_1|| \ldots ||b_k)$

(3) Verifizierer akzeptiert, wenn die ersten k Bits von $\mathrm{H}(m||b_1|| \ldots ||b_k)$ identisch mit $c_1, \ldots, c_k$ der Signatur sind.

[8] Eine genauere Betrachtung digitaler Signaturen findet sich im Versuch “Digitale Signatursysteme”.

AUFGABE 4
Zeigen Sie, dass der Verifizierer eine korrekte Signatur immer akzeptieren wird.

AUFGABE 5
Implementieren Sie die für das Erzeugen und Verifizieren von Signaturen notwendigen Methoden `sign()` in `NIProver.java` und `verify()` in `NIVerifier.java`. Zum Testen der Methoden steht `ZK_Signatur.java` zur Verfügung.

Eine nützliche Eigenschaft nicht-interaktiver Zero-Knowledge-Beweise ist die Übertragbarkeit der Ergebnisse, d.h. mit einem Beweis können nun mehrere Verifizierer von der Gültigkeit einer Behauptung überzeugt werden. Dies ist aber gleichzeitig auch das Hauptproblem solcher Verfahren, denn nun ist ein Verifizierer in der Lage einen Beweis zu kopieren und sich gegenüber einer dritten Person als Beweiser auszugeben. Dies ist ein Spezialfall der Man-In-The-Middle-Angriffe, die nun im folgenden Kapitel betrachtet werden.

7 Man-In-The-Middle-Angriffe

Als Adi Shamir '89 auf einer Konferenz das von ihm und Amos Fiat entworfene Zero-Knowledge-Identifikationsverfahren [FS87] und deren Anwendung auf Kreditkarten vorstellte, beschrieb er die damit erreichbare Sicherheit unter anderem mit folgenden Worten:

> *„I could go to a Mafia-owned store a million successive times and they will still not be able to misrepresent themselves as me."*

Wie sich später herausstellte, war diese Aussage allerdings etwas zu optimistisch. Beth und Desmedt beschreiben in [BD91] einen Angriff, mit dem es der Mafia doch gelingt:

> Alice befindet sich in „Bob's-Dinner" – einem zu der Mafia gehörenden Restaurant – während Carol zur selben Zeit in dem Juwelier „Daves Emporium" einkaufen geht. Bob und Carol sind beides Mitglieder der Mafia und kommunizieren über eine geheime Funkverbindung miteinander, Alice und Dave hingegen wissen nichts von dem bevorstehenden Betrug. Wenn Alice ihr Essen bezahlen und dazu ihre Identität Bob beweisen will, signalisiert dieser Carol, dass der Betrug beginnen kann. Carol wählt nun teure Diamanten und bereitet sich ebenfalls auf ihren Bezahlvorgang und ihre Identifikation gegenüber Dave vor. Sobald Alice ihren Beweis startet, leitet Bob sämtliche Werte mittels der Funkverbindung an Carol weiter, die das gleiche Protokoll nun mit Dave beginnt. Alle Fragen von Dave werden dann zurück an Bob übermittelt, der sich die korrekten Antworten von Alice geben lässt und wieder an Carol sendet, die nun Daves Fragen korrekt beantwortet.

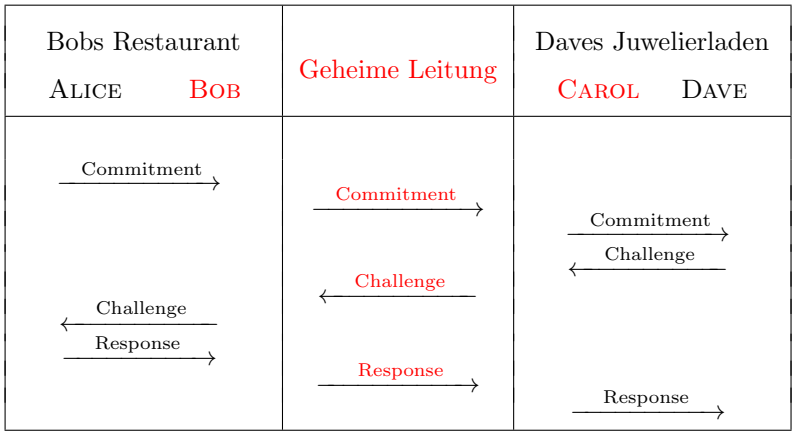

Abbildung 11: Man-In-The-Middle-Angriff nach [BD91]

> In Wirklichkeit beweist Alice ihre Identität also gegenüber Dave und nicht, wie sie annimmt, gegenüber Bob. Bob und Carol fungieren nur als Moderator und leiten alle Antworten und Fragen des Protokolls weiter. Ist der Beweis beendet hat sich Alice somit gegenüber Dave identifiziert und statt für einen Hamburger für ein paar sehr teure Diamanten bezahlt – mit denen Carol natürlich verschwindet.

Findet die Identifizierung nicht wie in dem Beispiel physisch über SmartCards, sondern logisch über ein Netzwerk statt, ist die Aufteilung auf zwei Angreifer nicht mehr nötig und Bob könnte den Betrug alleine durchführen.

Dieser Angriff wird als Mafia-Betrug oder auch als Man-In-The-Middle-Attack bezeichnet, da sich der (bzw. die) Angreifer zwischen zwei kommunizierenden Teilnehmern befindet und von da aus komplette Kontrolle über den Datenverkehr hat. Damit kann er beiden Teilnehmen die Identität des jeweils anderen vorspiegeln und so das an sich sichere Zero-Knowledge-Verfahren aushebeln. Ursache für diese Angriffsmöglichkeit ist also keine kryptographische Schwäche, sondern die offene und daher potentiell unsichere Umgebung, in der sich die Teilnehmer befinden.

Eine Möglichkeit, Man-In-the-Middle-Angriffe zu verhindern, ist daher die isolierte Ausführung der Beweise in geschützten Umgebungen, wie Faradayschen Käfigen, um so die Kommunikation der Angreifer zu unterbinden. Für Anwendungen in Netzwerken wurde die Einführung von Zeitschranken vorgeschlagen [BGD+91], d.h. jede Runde des Protokolls muss in einer bestimmten und recht kurz bemessenen Zeit durchgeführt werden, so dass für den Angreifer keine Zeit mehr bleibt, die einzelnen Fragen und Antworten weiterzuleiten. Beide Lösungsvorschläge versuchen allerdings nur den Angreifer in seinen Möglichkeiten einzuschränken, anstatt das Protokoll den erhöhten Sicherheitsanforderungen der Umgebung anzupassen. Denn das Problem für diese Schwäche liegt in der Unfähigkeit des Beweisers, die verschiedenen Verifizierer voneinander zu unterscheiden. Geht man aber

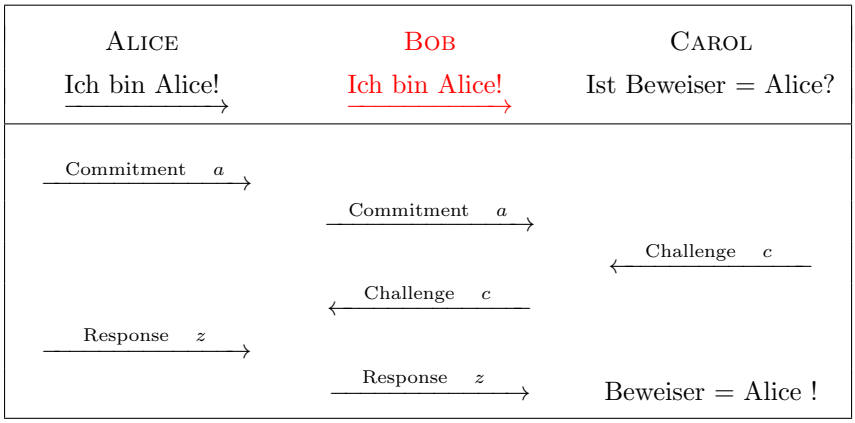

Abbildung 12: genereller Man-In-The-Middle-Angriff

davon aus, dass Verifizierer über Identitäten oder öffentliche Schlüssel verfügen, können diese Informationen als Adressierung des gewünschten Verifizierers in das Protokoll mit einbezogen werden.

Eine Technik, die diesen Ansatz umsetzt, wurde ’96 von Jakobsson, Saku und Impagliazzo in [JSI96] unter dem Namen *Designated Verifier Proofs* veröffentlicht. Ihre Motivation lag allerdings vor allem in der Verbesserung digitaler Signaturen, d.h. nicht-interaktiver Zero-Knowledge-Verfahren, da diese deutlich stärker unter der Angriffsmöglichkeit leiden. Denn während Man-In-The-Middle-Angriffe bei interaktiven Zero-Knowledge-Verfahren auf die Zeit der tatsächlich gewollten Kommunikation eingeschränkt sind, entziehen sich nicht-interaktive Beweise nach ihrer Ausführung der Kontrolle des Beweisers und können somit jeder Zeit ohne sein Mitwirken wiedergegeben werden.

Für den Fall des Wissenschaftlers A, der sich auf einer Weltreise befindet und nun Dank eines nicht-interaktiven Zero-Knowledge-Beweises, Wissenschaftler B davon überzeugen konnte, den Beweis für ein neues Theorem entdeckt zu haben, wäre das fatal. Denn während er sich nun auf der Heimreise befindet, könnte B den Zero-Knowledge-Beweis veröffentlichen und sich selber als Entdecker ausgeben.

Designated Verifier Proofs verhindern solch einen Missbrauch, indem der Beweis auf einen Empfänger (oder eine Gruppe von Empfängern) festgelegt wird, so dass nur dieser überzeugt werden kann und trotzdem nicht in der Lage ist, sich später als Beweiser oder Urheber der Signatur gegenüber Dritten auszugeben. Erreicht wird dies, indem das Zero-Knowledge-Protokoll so abgewandelt wird, dass der Beweiser Alice nun nicht mehr die Aussage „*Ich kenne das Geheimnis w*“, sondern die Aussage „*Ich kenne das Geheimnis w oder ich bin Bob*“ beweist. Für Bob ist dies natürlich überzeugend, versucht er aber den Beweis an einen anderen Verifizierer Carol weiterzuleiten, ist der zweite Teil des Aussage „*ich bin Bob*“ immer wahr, so dass Carol nicht überzeugt werden wird, dass Bob auch das Geheimnis w kennt.

Für die Umsetzung dieser Idee schlugen Cramer und Damgard in [CD97] die Verwendung des Konzeptes der OR-Protokolle [CDS94] vor. Zusätzlich wird, wie in allen Umsetzungen der Designated Verifier Proofs, das Vorhandensein von öffentlichen Schlüsseln, d.h. das Public Key Model, vorausgesetzt. Mithilfe des OR-Protokolls kann dann ein Beweiser zeigen, dass er für zwei gegebene Werte – hier die öffentlichen Schlüssel – x_A, x_B einen zugehörigen geheimen Schlüssel w kennt, so dass entweder $(x_A, w) \in R$ oder $(x_B, w) \in R$, allerdings ohne dass zu erkennen ist, welcher Fall zutrifft.

Das ist möglich, da Zero-Knowledge-Beweise so simuliert werden können, dass sie von echten Kommunikationstranskripten nicht unterscheidbar sind. In einem OR-Protokoll führt der Beweiser einen normalen Zero-Knowledge-Beweis für sein Geheimnis, d.h. für $(x_A, w) \in R$ durch und simuliert den anderen Fall $(x_B, w) \in R$. Als Grundlage eines solchen Schemas werden 3-Runden Honest-Verifier Zero-Knowledge-Protokolle eingesetzt. Dies hat den Vorteil, dass der Simulator die Rolle des ehrlichen Verifizierers übernehmen kann und die Challenge selbst wählt. Ein Beweiser $\mathcal{P}$ kann so mit Hilfe des Simulators $\mathcal{M}_\mathcal{V}$ und des öffentlichen Schlüssels x_B des Verifizierers korrekte Transkripte (b, d, y) für den Beweis der Kenntnis des geheimen Schlüssels w_B mit $(x_B, w_B) \in R$ erzeugen.

Schlüsselgenerierung

gegebene Parameter: Primzahlen p, q mit $q|(p-1)$, Generator g der Untergruppe G_q
Jeder Teilnehmer wählt w zufällig in $\mathbb{Z}_q$ und erzeugt sein Schlüsselpaar – bestehend aus einem öffentlichen Schlüssel x und einem geheimen Schlüssel w – durch:

$$(x = g^w \bmod p, \quad w)$$

Protokoll

Beweiser $\mathcal{P}$		**Verifizierer $\mathcal{V}$**
geg.: $(x_A, w_A), x_B$		geg.: $x_A, (x_B, w_B)$
wählt s zufällig in $\mathbb{Z}_q$ berechnet $a = g^s \bmod p$		
wählt $y \in \mathbb{Z}_q$ und $d \in \{0,1\}$ berechnet $b = g^y \cdot x_B^{-d} \bmod p$		
	$\xrightarrow{(a,b)}$	
		wählt $C \in \{0,1\}$
	$\xleftarrow{C}$	
berechnet $c = C \oplus d$ und $z = (s + c \cdot w_A) \bmod q$		
	$\xrightarrow{(z,d,y)}$	
		überprüft ob $g^z \stackrel{?}{=} a \cdot x_A^c \bmod p$ und $g^y \stackrel{?}{=} b \cdot x_B^d \bmod p$ mit $c = C \oplus d$

Abbildung 13: OR-Version des Zero-Knowledge-Protokolls für diskrete Logarithmen

Für einen Zero-Knowledge-Beweis der Kenntnis eines diskreten Logarithmus ergibt sich in Abbildung 13 dargestelltes Verfahren. Für die Erzeugung einer korrekten Transkriptes (b, d, y) des simulierten Beweises für $(x, w_B) \in R$ wählt der Beweiser im ersten Schritt eine Challenge d und eine Antwort y. Die Festlegung b, die an den Verifizierer zu senden ist, ergibt sich dann durch $b = g^y \cdot x_B^{-d} \bmod p$. Im ersten Schritt einer Beweisrunde sendet $\mathcal{P}$ nun neben der üblichen Festlegung a die Festlegung b aus dem simulierten Beweis. Der Verifizierer antwortet daraufhin wie gewohnt mit einer Challenge C, zwei werden nicht benötigt, da sich der Beweiser in der Simulation bereits auf eine Challenge d festgelegt hat. Diese hält er jedoch noch geheim und wählt die Challenge c für den „richtigen“ Beweis in Abhängigkeit von C und d, so dass $c \oplus d = C$ gilt. Im nächsten Beweisschritt kann der Beweiser nun die korrekten Antworten z und y liefern. Damit der Verifizierer auch in der Lage ist, diese Antworten zu überprüfen, erhält er zusätzlich die Challenge d, die zweite tatsächlich verwendete Challenge c kann er mit Hilfe von C selbst berechnen. Durch dieses Vorgehen sieht der Verifizierer am Ende zwei korrekt durchgeführte Beweise, ohne anhand des Protokolls erkennen zu können, welcher davon simuliert wurde. Da der Beweiser in dem konkreten Fall allerdings $(x_A, w) \in R \ \vee \ (x_B, w) \in R$ bewiesen hat, wird der Verifizierer – unter der Annahme, dass er seinen geheimen Schlüssel w_B nicht preisgegeben hat – davon überzeugt sein, dass der Beweiser das Geheimnis w zu x_A kennt. Würde der Verifizierer aber nun versuchen, den Beweis weiterzuleiten, kann er niemanden überzeugen, dieses w zu kennen, da er einen korrekten Beweis genauso gut durch sein Geheimnis w_B erzeugen kann.

Mithilfe dieses Schemas können also Zero-Knowledge-Beweise entworfen werden, die resistent gegen Man-In-The-Middle-Angriffe sind. Voraussetzung für diese Sicherheit ist aber, dass alle Teilnehmer unterschiedliche Schlüsselpaare besitzen, die korrekt entsprechend des Schlüsselgenerierungsalgorithmus erstellt wurden. Ansonsten könnte der Angreifer den öffentlichen Schlüssel des von ihm gewünschten Verifizierers einfach an den Beweiser weitergeben, bzw. seinen eigenen Schlüssel in Abhängigkeit davon bilden. Um diese Möglichkeiten auszuschließen, kann auf eine Trusted Third Party zurückgegriffen werden, bei der alle Teilnehmer ihre öffentlichen Schlüssel registrieren lassen und mittels eines Zero-Knowledge-Beweises die Kenntnis des zugehörigen geheimen Schlüssels nachweisen.

AUFGABE 6

(a) Vollziehen Sie mithilfe des Programmes `ZK_Attack.java` den Man-In-The-Middle-Angriff auf den in Aufgabe 3 implementierten Zero-Knowledge-Beweis.

(b) Implementieren Sie die Methoden `or_sendCommitment()`, `or_sendResponse()` und `or_verify()` in `Prover.java` und `Verifier.java` nach dem beschriebenen OR-Schema. Überprüfen Sie die Funktionsfähigkeit des erweiterten Beweises sowie dessen Verhalten bei einem Man-In-The-Middle-Angriff mithilfe von `ZK_Attack.java`.

AUFGABE 7

Würde der Man-In-the-Middle-Angriff auf solch ein OR-Protokoll gelingen, wenn Bob jeweils nur den Beweis von Alice Geheimnis – also die Werte (a, z) – weiterleitet bzw. in einem angepassten OR-Protokoll zusammen mit dem öffentlichen Schlüssel von Carol verwendet?

8 Zusammenfassung

Ist das Man-In-The-Middle-Problem behoben, bieten Zero-Knowledge-Verfahren eine ideale Möglichkeit zur Authentifikation von Nutzern, da sie beweisbar die Geheimhaltung der dazu notwendigen Geheimnisse leisten. Zero-Knowledge-Verfahren können dazu als reine Authentikationssysteme in Chipkarten oder Onlinesystemen verwendet werden oder auch als Bausteine in komplexeren Anwendungen wie z.B. Credential-Systemen [CL01], eVoting [CGS97] oder eCash [Bra93] dienen. Dabei werden sie meist zum Nachweis von protokollkonformem Verhalten der Teilnehmer bzw. des Systems eingesetzt. In Credential- und eCash-Anwendungen kann zudem mittels Zero-Knowledge-Protokollen der Besitz von Credentials bzw. digitalem Bargeld nachgewiesen werden, ohne dies vorzeigen zu müssen.

Zwar eignen sich Zero-Knowledge-Verfahren nicht für Verschlüsselungssysteme, sie können aber helfen, die bereits existierenden Public-Key Kryptosysteme zu verbessern. Denn mit nicht-interaktiven Zero-Knowledge-Beweisen ist es möglich, Public-Key Verschlüsselungssysteme so zu erweitern, dass sie ihre Sicherheit selbst bei der stärksten Form der Chosen-Ciphertext-Angriffe behalten. Mit der in Kapitel 6.2 beschriebenen Technik ist es zudem möglich, nicht-interaktive Zero-Knowledge-Beweise für digitale Signaturen zu nutzen. Dabei bieten sie eine gute Alternative gegenüber üblichen Signaturverfahren wie z.B. RSA, da sie schneller zu berechnen sind und kürzere Signaturlänge erfordern als vergleichbare Public-Key Verfahren.

Werden heutzutage Zero-Knowledge-Verfahren eingesetzt, geschieht dies sogar meist aus Gründen der Effizienz und nicht aufgrund der Zero-Knowledge-Eigenschaft. Da Zero-Knowledge-Verfahren einen Beweis in einen iterativen Prozess einfacherer Berechnungen untertei-

len, besitzen sie einen deutlich geringeren Berechnungsbedarf als Verfahren, die aus einem großen Komplex bestehen. In der Regel benötigen Zero-Knowledge-Verfahren nur 1/10 bis 1/100 des Rechenaufwandes von vergleichbaren Public-Key Systemen. Das Fiat-Shamir Verfahren führt z.B. bei einem Modulus n von 512 bit nur 11 bis 30 modulare Multiplikationen durch, während ein (nicht optimiertes) RSA-Verfahren 768 dieser Berechnungen benötigt [MVO96].

Durch diese Eigenschaft kommen Zero-Knowledge-Verfahren für den Einsatz auf Smart-Cards in Frage, für die sich bisher, aufgrund der geringen Rechenleistung, nur symmetrische Kryptoverfahren eigneten. Mit Zero-Knowledge-Verfahren lassen sich also Anwendungen unter Nutzung der Vorteile der asymmetrischen Verfahren realisieren, die zuvor aus Effizienzgründen nur den symmetrischen Systemen vorbehalten waren. Bereits 1991 bei der Patentanmeldung für das Verschlüsselungssystem „videocrypt“, das in Pay-TV Systemen zum Einsatz kommt, wurde diese Möglichkeit erkannt und durch die Einbeziehung des Fiat-Shamir Verfahrens genutzt. Die Decoder für solche Pay-TV Systeme enthalten keinerlei Geheimnisse und können frei im Handel verkauft werden. Erst durch das Einlegen einer Chipkarte – die sämtliche Informationen zur Programmentschlüsselung enthält – sind die gewünschten Sender benutzbar. Der Decoder unterscheidet dabei mittels des Fiat-Shamir Verfahrens zwischen echten und gefälschten Karten.

Zusammenfassend lässt sich feststellen, dass Zero-Knowledge-Verfahren insbesondere in Bereichen der Authentifikation eine starke Konkurrenz zu herkömmlichen Public-Key Verfahren darstellen. Zero-Knowledge-Verfahren besitzen ebenfalls die Vorteile der asymmetrischen Kryptosysteme, verfügen jedoch über deutlich effizientere Implementierungen und bieten die perfekte Geheimhaltung der zur Authentifikation nötigen Informationen. Ein Nachteil ist allerdings die teilweise hohe Anzahl an notwendigen Interaktionen, die zu einem starken Kommunikations- und Zeitaufwand führen kann. Mit nicht-interaktiven Zero-Knowledge-Verfahren oder Modifikationen wie dem Schnorr-Protokoll lässt sich zwar dieser Aufwand reduzieren, jedoch geht dabei aus formalen Gründen meist die Zero-Knowledge-Eigenschaft verloren. Die Verfahren sind aber dennoch klassisch sicher, und werden – wenn auf die Simulierbarkeit verzichtet werden kann – bevorzugt eingesetzt.

Man muss allerdings immer bedenken, dass Zero-Knowledge-Verfahren, wie alle asymmetrischen Verfahren, auf der unbewiesenen Annahme, dass $\mathcal{P} \neq \mathcal{NP}$ gilt, basieren. Bei einem Gegenbeweis wären diese Verfahren damit nutzlos.

Literatur

[BCC88] Brassard, Gilles ; Chaum, David ; Crepeau, Claude: Minimum disclosure proofs of knowledge. In: *Journal of Computer and System Sciences* 37 (1988), Nr. 2, S. 156–189

[BD91] Beth, Thomas ; Desmedt, Yvo: Identification Tokens - or: Solving the Chess Grandmaster Problem. In: *CRYPTO '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, Springer-Verlag, 1991, S. 169–177

[BFM88] Blum, Manuel ; Feldman, Paul ; Micali, Silvio: Non-interactive zero-knowledge and its applications. In: *STOC '88: Proceedings of the 20th annual ACM symposium on Theory of computing*, ACM Press, 1988, S. 103–112

[BG89] Bellare, Mihir ; Goldwasser, Shafi: New paradigms for digital signatures and message authentication based on non-interactive zero knowledge proofs. In: *CRYPTO '89: Proceedings on Advances in cryptology*, Springer-Verlag New York, Inc., 1989, S. 194–211

[BGD+91] Bengio, Samy ; Grassard, Gilles ; Desmedt, Yvo G. ; Goutier, Claude ; Quisquater, Jean-Jacques: Secure Implementation of Identification Systems. In: *Journal of Cryptology* 4 (1991), S. 175–183

[BJW99] Beutelspacher, A. ; J.Schwenk ; Wolfenstetter, K.-D.: *Moderne Verfahren der Kryptographie - von RSA zu Zero-Knowledge.* Vieweg-Verlag, 1999

[BM88] Babai, Lazlo ; Moran, Shlomo: Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity class. In: *Journal of Computer and System Sciences* 36 (1988), Nr. 2, S. 254–276

[BR93] Bellare, Mihir ; Rogaway, Phillip: Random oracles are practical: a paradigm for designing efficient protocols. In: *CCS '93: Proceedings of the 1st ACM conference on Computer and communications security*, ACM Press, 1993, S. 62–73

[Bra93] Brands, Stefan: Untraceable off-line cash in wallet with observers. In: *CRYPTO '93: Advances in Cryptology* Bd. 773, Springer-Verlag, 1993 (LNCS), S. 302–318

[BSMP91] Blum, Manuel ; Santis, Alfredo D. ; Micali, Silvio ; Persiano, Giuseppe: Noninteractive Zero-Knowledge. In: *SIAM Journal on Computing* 20 (1991), Nr. 6, S. 1084–1118

[CD97] Cramer, Ronald ; Damgård, Ivan: Fast and Secure Immunization Against Adaptive Man-in-the-Middle Impersonation. In: *EUROCRYPT '97: Advances in Cryptology* Bd. 1233, Springer-Verlag, 1997 (LNCS), S. 75–87

[CDG88] Chaum, David ; Damgård, Ivan ; Graaf, Jeroen van d.: Multiparty Computations Ensuring Privacy of Each Party's Input and Correctness of the Result. In: *CRYPTO '87: A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, Springer-Verlag, 1988, S. 87–119

[CDS94] Cramer, Ronald ; Damgård, Ivan ; Schoenmakers, Berry: Proof of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In: *LNCS* 839 (1994), S. 174–187

[CEG87] Chaum, David ; Evertse, Jan-Hendrik ; Graaf, Jeroen van d.: An Improved Protocol for Demonstrating Possession of Discrete Logarithms and Some Generalizations. In: *EUROCRYPT*, 1987, S. 127–141

[CEGP87] Chaum, David ; Evertse, Jan-Hendrik ; Graaf, Jeroen van d. ; Peralta, Rene: Demonstrating possession of a discrete logarithm without revealing it. In: *CRYPTO '86: Proceedings on Advances in cryptology*, Springer-Verlag, 1987, S. 200–212

[CGH98] Canetti, Ran ; Goldreich, Oded ; Halevi, Shai: The random oracle methodology, revisited (preliminary version). In: *STOC '98: Proceedings of the 30th annual ACM symposium on Theory of computing*, ACM Press, 1998, S. 209–218

[CGS97] Cramer, Ronald ; Gennaro, Rosario ; Schoenmakers, Berry: A Secure and Optimally Efficient Multi-Authority Election Scheme. In: *LNCS* 1233 (1997), S. 103–116

[CL01] Camenisch, Jan ; Lysyanskaya, Anna: An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In: *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, Springer-Verlag, 2001, S. 93–118

[DNS98] Dwork, Cynthia ; Naor, Moni ; Sahai, Amit: Concurrent zero-knowledge. In: *STOC '98: Proceedings of the 30th annual ACM symposium on the Theory of computing*, ACM Press, 1998, S. 409–418

[Fei90] Feige, Uriel: *Alternative Ways for Zero Knowledge Interactive Proofs.* Rehovot, Israel, Weizmann Institute of Science, Diss., 1990

[FS87] Fiat, Amos ; Shamir, Adi: How to prove yourself: practical solutions to identification and signature problems. In: *CRYPTO '86: Proceedings on Advances in cryptology*, Springer-Verlag, 1987, S. 186–194

[FS90] Feige, Urel ; Shamir, Adi: Witness indistinguishable and witness hiding protocols. In: *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, ACM Press, 1990, S. 416–426

[GMR85] Goldwasser, Shafi ; Micali, Silvio ; Rackoff, Charles: The knowledge complexity of interactive proof-systems. In: *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, ACM Press, 1985, S. 291–304

[GMR89] Goldwasser, Shafi ; Micali, Silvio ; Rackoff, Charles: The knowledge complexity of interactive proof systems. In: *SIAM Journal on Computing* 18 (1989), Nr. 1, S. 186–208

[GMW86] Goldreich, Oded ; Micali, Silvio ; Wigderson, Avi: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. In: *Journal of the ACM* 38 (1986), Nr. 3, S. 690–728

[Gol02] Goldreich, Oded: Zero-knowledge twenty years after its invention. In: *Electronic Colloquium on Computational Complexity* (2002)

[JSI96] Jakobsson, Markus ; Sako, Kazue ; Impagliazzo, Russell: Designated Verifier Proofs and Their Applications. In: *LNCS* 1070 (1996), S. 143–158

[MVO96] Menezes, Alfred J. ; Vanstone, Scott A. ; Oorschot, Paul C. V.: *Handbook of Applied Cryptography*. CRC Press, Inc., 1996

[Pfi99] Pfitzmann, Birgit: *Higher cryptographic protocols*. 1999. – Vorlesungsscript

[QGAB89] Quisquater, Jean-Jacques ; Guillou, Louis ; Annick, Marie ; Berson, Tom: How to explain zero-knowledge protocols to your children. In: *CRYPTO '89: Proceedings on Advances in cryptology*, Springer-Verlag New York, Inc., 1989, S. 628–631

[Sch89] Schnorr, Claus P.: Efficient identification and signatures for smart cards. In: *CRYPTO '89: Proceedings on Advances in cryptology*, Springer-Verlag New York, Inc., 1989, S. 239–252

[Sch98] Schneier, Bruce: *Applied cryptography. protocols, algorithms, and source code in C*. Wiley, 1998

[vid91] videocrypt: *European Patent Application 0 428 252 A2, A System for Controlling Access to Broadcast Transmissions*. 1991