ETH Zürich
Departement Informatik

Vorlesungsunterlagen

Kryptographische Protokolle

Martin Hirt und Ueli Maurer

Frühjahrssemester 2009

Vorwort

Kryptographische Protokolle, die oft ziemlich paradox erscheinende Eigenschaften haben, gehören zu den Rosinen der Kryptographie. Diese Vorlesung baut auf der Vorlesung *Kryptographie* auf und gibt einen Einblick in einige moderne Forschungsbereiche der Kryptographie. Im Vordergrund steht die Freude an grundlegenden und theoretischen Fragestellungen, aber die vielen Anwendungen werden natürlich auch diskutiert. Es werden drei Schwerpunktthemen behandelt:

- Interaktive Beweise und zero-knowledge Protokolle
- Sichere Multi-Party Berechnungen
- Digitale Abstimmungen (E-Voting)

Die ersten beiden Themen sind zentrale Paradigmen der modernen Kryptographie, sowohl in der Forschung wie auch immer mehr in den Anwendungen. Das dritte Thema ist eine konkrete Anwendung des zweiten.

Zero-knowledge Protokolle erlauben, einen Beweis einer Tatsache (oder des Wissens bestimmter geheimer Information S) zu liefern, ohne jegliche Information über den Beweis (respektive über S) wegzugeben. Sichere Multi-Party Berechnung erlaubt die Kooperation mehrerer sich nicht notwendigerweise vertrauenden Entitäten, ohne dass der Input einer Entität in die Kooperation den restlichen Entitäten bekannt wird und ohne dass Betrüger unter den Entitäten die Berechnung verfälschen können. Darunter fallen z.B. Protokolle für sichere und geheime digitale Abstimmungen ohne vertrauenswürdige Auswertungsinstanz.

Zürich, im Februar 2009 Martin Hirt und Ueli Maurer

Inhaltsverzeichnis

Kapitel 1

Interaktive Beweise und zero-knowledge Protokolle

1.1 Einleitung und Motivation

1.1.1 Konventionelle und interaktive Beweise

Ein konventioneller Beweis einer Aussage (z.B. einer mathematischen Vermutung) ist eine Folge von elementaren, einzeln nachvollziehbaren, aufeinander aufbauenden Schritten, wobei der letzte Schritt zur zu beweisenden Aussage führt. Dabei wird selbstverständlich, aber nicht explizit angenommen, dass der Beweis effizient verifizierbar ist, d.h. dass MathematikerInnen jeden Schritt in vernünftiger Zeit nachvollziehen können.

Normalerweise betrachtet man Beweise, die als Text mit Formeln ausgestaltet sind und durch Menschen verifiziert werden. Aussagen und Beweise können aber auch im Sinne einer Logik und eines Beweiskalküls präzise formalisiert werden, so dass ein Beweis im Prinzip durch einen Computer automatisch verifiziert werden kann, ausgehend von den Axiomen der entsprechenden Theorie (z.B. Gruppentheorie oder Zahlentheorie).

Im Gegensatz zu einem konventionellen Beweis erfolgt ein *interaktiver Beweis* durch Kommunikation zwischen dem sogenannten Beweiser P und dem sogenannten Verifizierer V, die man sich als Personen oder Computerprogramme denken kann. Oft werden wir die Beweiserin Peggy und den Verifizierer Vic nennen.

Gegeben ist eine Aussage oder Behauptung, die Peggy gegenüber Vic beweisen möchte. Beispiele von Aussagen sind, dass die in einem bestimmten Chiffrat enthaltene Nachricht eine bestimmte Form besitzt, dass eine bestimmte mathematische Vermutung wahr ist, oder die Aussage, dass Peggy den geheimen Schlüssel zu einem bestimmten Public Key kennt. Der Beweis der letzten Aussage kann z.B. zu Identifikationszwecken verwendet werden.

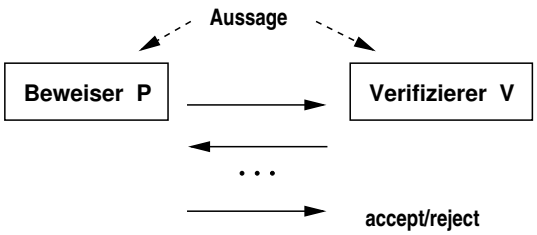


Abb. 1.1: Interaktiver Beweis für eine Aussage.

Ein interaktiver Beweis (siehe Abb. 1.1) für eine Aussage ist ein Protokoll zwischen Peggy und Vic, das spezifiziert, welche Nachrichten die beiden in welcher Reihenfolge einander senden müssen. Beide können (geheime) zufällige Bits wählen, falls dies nötig ist. Am Ende des Protokolls macht Vic eine binäre Entscheidung darüber, ob er den Beweis akzeptiert oder nicht.

Peggy und Vic können nicht gezwungen werden, sich an das Protokoll zu halten. Selbst in diesem Fall soll der interaktive Beweis funktionieren. Insbesondere soll Peggy keine falsche Behauptung beweisen können, und eine weitere Anforderung kann sein, dass Vic nicht in der Lage sein soll, durch falsches Verhalten Information zu bekommen, die ihm nicht zusteht.

Interaktive Beweise wurden von Goldwasser, Micali und Rackoff 1985 eingeführt [GMR85, GMR89]. Sie werden als eine der wichtigsten Entdeckungen resp. Entwicklungen der theoretischen Informatik der letzten Jahre angesehen. Die Theorie ist wie die Komplexitätstheorie stark formalisiert, aber wir werden hier versuchen, ohne unnötigen Formalismus auszukommen, ohne die Präzision zu vernachlässigen.

1.1.2 Motivation interaktiver Beweise

Es gibt mindestens die folgenden drei praktischen und theoretischen Motivationen für die Betrachtung interaktiver Beweise:

- **Anwendungen.** Interaktive Beweise haben verschiedene wichtige Anwendungen:
  - **Identifikationsprotokolle** (siehe 1.3.1).
  - als Basis für die Konstruktion von **digitalen Signaturverfahren** (siehe 1.3.2).
  - im Entwurf sicherer **Multi-Party Berechnungen**. Eine Entität kann z.B. beweisen, dass sie eine gewisse Berechnung auf ihren geheimen Daten korrekt durchgeführt hat und eine korrekte Verschlüsselung des Resultats publiziert hat, ohne dabei das Resultat zu zeigen.
- **Reduktion der transferierten Information.** Bei einem konventionellen Beweis erhält Vic Information, die er vorher nicht besass, nämlich mindestens den Beweis der Aussage selbst. Insbesondere könnte er den Beweis an andere Personen weitergeben, ohne dass Peggy dies verhindern kann. Im Gegensatz dazu können interaktive Beweise so durchgeführt werden, dass Vic absolut keine Information erhält, die er nicht schon vor dem Beweisprotokoll besass (solche Protokolle werden *zero-knowledge* genannt) oder zumindest keine Information, die ihm etwas nützt.[1]
- **Vielfalt der beweisbaren Aussagen.** Gewisse Aussagen kann man konventionell gar nicht beweisen, weil der Beweis viel zu lang wäre (z.B. länger als die gesamten bis heute hergestellten Datenspeicher). Interaktiv kann der Beweis aber oft effizient erfolgen. Diese Motivation ist primär theoretisch, da man in der Regel annehmen muss, dass Peggy sehr grosse Rechenressourcen besitzt. Die Sichtweise ist, dass eine allmächtige Instanz einen beschränkten Verifizierer von der Wahrheit einer bestimmten Aussage (z.B. die noch unbewiesene Aussage, dass es unendlich viele Primzahlpaare gibt, oder $\mathbf{P} = \mathbf{NP}$) überzeugen möchte, die dieser ohne Hilfe nie beweisen könnte.

Viele Entscheidungsprobleme (siehe Bsp. 1.1) sind asymmetrisch in folgendem Sinn. Falls die Antwort „ja“ ist, kann man einen kurzen und einfach verifizierbaren Beweis dafür angeben, falls die Antwort aber „nein“ ist, so gibt es keinen kurzen und effizient verifizierbaren Beweis. Eine allmächtige Instanz, welche das Problem beliebig schnell lösen kann (z.B. mittels Durchprobieren aller Möglichkeiten), kann diese Überzeugung trotzdem nicht an einen berechenmässig beschränkten Verifizierer weitergeben, da dieser im wesentlichen die gleiche Berechnung nachvollziehen müsste, was berechenmässig zu aufwendig ist.[2]

Beispiel 1.1. Wir betrachten das folgende Entscheidungsproblem: Gibt es für einen gegebenen gerichteten Graphen $\mathcal{G}$ einen *Hamiltonschen Kreis*, d.h. einen geschlossenen Pfad, der jeden Knoten genau einmal besucht? Ist die Antwort ja, so ist jeder Hamiltonsche Kreis ein einfach verifizierbarer Beweis. Für die Nichtexistenz eines Hamiltonschen Kreises in einem Graphen ist hingegen kein effizienter Beweis bekannt. Es

[1] Die Formalisierung der zero-knowledge Eigenschaft ist nicht offensichtlich, da unklar ist, wie man die Menge des in einem Protokoll transferierten Wissens misst.

[2] Natürlich könnte der Verifizierer dem Beweiser einfach glauben, aber dies ist ja nicht der Zweck eines Beweises.

gibt zwar einen effizienten interaktiven Beweis für die Nichtexistenz eines Hamiltonschen Kreises, aber die Konstruktion ist sehr kompliziert. Wir geben deshalb in Abschnitt 1.2.2 ein ähnliches Beispiel, in dem der interaktive Beweis einfach ist.

1.1.3 Typen von interaktiven Beweisen

Man unterscheidet zwischen zwei Typen von Beweisen:

- **Beweise von Aussagen**, im oben diskutierten Sinn. Beispiele von Aussagen sind:
  - Eine gegebene Zahl n hat genau 3 Primfaktoren. Ein möglicher Beweis für diese Aussage besteht darin, die Primfaktoren anzugeben und zu beweisen, dass diese Primzahlen sind.
  - Es gibt unendlich viele Primzahlpaare.
  - Zwei gegebene Graphen sind isomorph.
  - Ein gegebener Graph $\mathcal{G}$ enthält einen Hamiltonschen Kreis.
  - Es gibt unendlich viele Primzahlpaare.[3]
  - $\mathbf{P} = \mathbf{NP}$.
- **Beweise von Wissen (Proof of Knowledge).** Peggy beweist Vic, dass sie bestimmte Information (z.B. einen geheimen Schlüssel) kennt. Es wird davon ausgegangen, dass diese Information verifiziert werden kann. Ein einfacher Beweis besteht darin, dass Peggy die Information an Vic schickt.
  - Ich kenne den geheimen Schlüssel, der zu einem bestimmten Public Key gehört (z.B. den diskreten Logarithmus x des Public Keys $z = g^x$, wobei g ein Generator einer zyklischen Gruppe ist).
  - Ich weiss, ob $\mathbf{P} = \mathbf{NP}$ oder $\mathbf{P} \neq \mathbf{NP}$ gilt. Es geht hier nicht um die (triviale) Aussage, dass entweder $\mathbf{P} = \mathbf{NP}$ oder $\mathbf{P} \neq \mathbf{NP}$ gilt, sondern darum, ob Peggy einen Beweis für die eine oder andere Aussage kennt. Vic muss dabei nicht notwendigerweise erfahren, welche der Aussagen $\mathbf{P} = \mathbf{NP}$ oder $\mathbf{P} \neq \mathbf{NP}$ wahr ist.
  - h sei eine Einweg-Hashfunktion und y sei ein gegebener Wert, z.B. ein notariell mit Zeitstempel beglaubigter Hashwert einer Erfindung. Peggy möchte einen Teil s der Beschreibung der Erfindung (aber nicht die ganze Erfindung) preisgeben, d.h. sie möchte beweisen, dass sie u und v kennt mit $h(u||s||v) = y$.

Oft erfolgt ein Beweis einer Aussage als Beweis von Wissen: Peggy beweist, dass sie einen Beweis für die Aussage kennt, und damit natürlich auch, dass die Aussage wahr ist. Für Aussagen ohne kurzen und effizient verifizierbaren Beweis kann dieser Ansatz aber nicht verwendet werden (siehe Beispiel 1.1).

1.1.4 Anforderungen an ein Beweissystem

Die Grundanforderungen an ein Beweissystem (konventionell oder interaktiv) sind:

- **Vollständigkeit** (completeness)**:** Wenn eine Aussage wahr ist (resp. im Fall eines Beweises von Wissen, wenn Peggy die fragliche Information kennt), dann wird der Beweis akzeptiert.[4]
- **Widerspruchsfreiheit** (soundness)**:** Wenn eine Aussage falsch ist (resp. Peggy die fragliche Information nicht kennt), dann gibt es keine Strategie für Peggy, Vic zu überzeugen. Diese Anforderung kann auch abgeschwächt werden, indem man nur verlangt, dass jeder Beweis für eine falsche Aussage mit höchstens vernachlässigbarer, d.h. äussert kleiner Wahrscheinlichkeit akzeptiert wird.

[3] Es ist nocht bekannt, ob diese Aussage wahr oder falsch ist.

[4] Diese Anforderung kann auch abgeschwächt werden, indem man nur verlangt, dass der Beweis für eine wahre Aussage mit überwältigender (d.h. äusserst nahe bei 1) Wahrscheinlichkeit akzeptiert wird. (Diese Abschwächung wird aber nicht nötig sein, d.h. in allen bekannten Beispielen ist die Wahrscheinlichkeit gleich 1.)

Im Folgenden diskutieren wir einige wichtige Kriterien und mögliche Anforderungen an Beweisverfahren:

- **Effizienz** in Bezug auf
  - den Berechnungsaufwand für Peggy
  - den Berechnungsaufwand für Vic
  - den Kommunikationsaufwand
  - die Anzahl Runden[5] eines Protokolls.
- **Allgemeinheit:** Welche Typen von Aussagen können mit dem Verfahren bewiesen werden? Gewisse Verfahren sind sehr allgemein und erlauben jede mögliche Aussage zu beweisen. Andere Verfahren sind auf Aussagen eines ganz bestimmten Typs beschränkt (z.B. ob ein Graph einen Hamiltonschen Kreis besitzt).
- **Informationsverlust:** Wieviel Information erhält der Verifizierer durch das Protokoll? (Z.B. keine Information, oder zumindest keine nützliche Information.)
- **Typ der Sicherheit:** Welche **kryptographischen Annahmen** werden gemacht? (Z.B. Schwierigkeit des Faktorisierens grosser Zahlen.) Die Sicherheit kann für Peggy (oder Vic) **informationstheoretisch** sein, d.h. die andere Partei kann selbst mit unendlichen Computerressourcen nicht betrügen resp. Information erhalten.

1.1.5 Kapitelübersicht

Im Rest dieses Kapitels behandeln wir zunächst einige motivierende Beispiele von interaktiven Beweisen (Abschnitt 1.2). In Abschnitt 1.3 diskutieren wir Anwendungen interaktiver Beweise und in Abschnitt 1.4 führen wir sogenannte Bit-Commitments ein. Die notwendige Formalisierung interaktiver Beweise wird in Abschnitt 1.5 diskutiert, um in Abschnitt 1.6 zero-knowledge Beweise formal definieren zu können. In Abschnitt 1.7 führen wir Beweise von Wissen ein. Hier muss formalisiert werden, was es bedeutet, dass ein Computer etwas "weiss". In Abschnitt 1.8 behandeln wir ein Protokoll, mit dem jede grundsätzlich verifizierbare Aussage auch in zero-knowledge bewiesen werden kann. In Abschnitt 1.9 diskutieren wir ein allgemeines und abstraktes Framework für bestimmte Typen von interaktiven Beweisen.

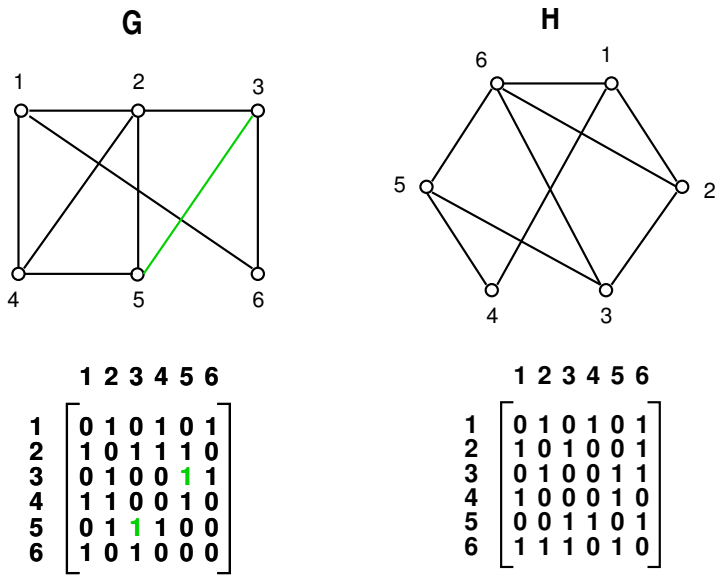
1.2 Einfache Beispiele von interaktiven Beweisen

In Abschnitt 1.1.2 wurden drei Anwendungen interaktiver Beweise erwähnt, für die wir in diesem Abschnitt einige Beispiele geben, noch bevor das Konzept eines interaktiven Beweises oder die zero-knowledge Eigenschaft formal definiert wird. In Abschnitt 1.2.1 geben wir einen Beweis für die Aussage, dass zwei Graphen isomorph sind (und dass Peggy den Isomorphismus kennt), und in Abschnitt 1.2.2 wird ein Protokoll für den Beweis der Nichtisomorphie zweier Graphen diskutiert, für die es vermutlich keinen kurzen konventionellen Beweis gibt. Die Beispiele in den Abschnitten 1.2.3, 1.2.4 und 1.2.5 sind Protokolle, die in der Praxis als Identifikationsprotokolle und für andere Anwendungen eingesetzt werden. Das letzte Beispiel in Abschnitt 1.2.6 verwendet Bit-Commitments, eine neue kryptographische Primitive, die in Abschnitt 1.4 formalisiert wird.

1.2.1 Graphenisomorphismus (GI)

Wir betrachten das *Graphenisomorphieproblem* (GI) respektive im nächsten Abschnitt das *Graphennichtisomorphieproblem* (GNI). Gegeben sind zwei Graphen $\mathcal{G}$ und $\mathcal{H}$ und die Frage ist, ob die Graphen isomorph sind (oder nicht).

[5] In einer "Runde" kann jeder Protokollteilnehmer jedem anderen eine Meldung schicken. Gelegentlich wird auch ein ganzer Durchlauf des Protokolls als Runde bezeichnet.



	1	2	3	4	5	6
1	0	1	0	1	0	1
2	1	0	1	1	1	0
3	0	1	0	0	1	1
4	1	1	0	0	1	0
5	0	1	1	1	0	0
6	1	0	1	0	0	0

	1	2	3	4	5	6
1	0	1	0	1	0	1
2	1	0	1	0	0	1
3	0	1	0	0	1	1
4	1	0	0	0	1	0
5	0	0	1	1	0	1
6	1	1	1	0	1	0

Abb. 1.2: Darstellung von Graphen durch die Adjazenzmatrix. Die Graphen $\mathcal{G}$ und $\mathcal{H}$ sind isomorph.

Graphen sind mathematische Objekte (und nicht Zeichnungen). Ein Graph mit n Knoten kann auf verschiedene Arten repräsentiert werden. Die erste gebräuchliche Art ist, einen Graphen durch die Menge der Punkte und die Menge der Kanten zu beschreiben, wobei eine (gerichtete) Kante ein (geordnetes) Paar von Knoten ist.

Für uns ist die Beschreibung mittels der *Adjazenzmatrix* besser geeignet, wobei man sich die Knoten des Graphen von 1 bis n nummeriert denkt. Die Adjazenzmatrix (siehe Abb. 1.2) ist eine binäre $n \times n$ Matrix. Eine 1 an der Position (i, j) (i-te Zeile, j-te Kolonne) bedeutet, dass eine Kante von Knoten i nach Knoten j führt. Die Adjazenzmatrix eines ungerichteten Graphen ist symmetrisch.

Im Folgenden setzen wir oft den Graphen und seine Adjazenzmatrix gleich. Zwei Graphen $\mathcal{G}$ und $\mathcal{H}$ (siehe Abb. 1.2) sind isomorph, wenn es eine Permutation der Knotenbeschriftung von $\mathcal{G}$ gibt, so dass die Adjazenzmatrix gleich derjenigen von $\mathcal{H}$ wird. Ein Isomorphismus σ entspricht einer Permutationsmatrix, d.h. einer $n \times n$ Matrix mit einer 1 in jeder Zeile und einer 1 in jeder Kolonne, und den restlichen Einträgen gleich 0. $\mathcal{G}$ und $\mathcal{H}$ sind isomorph (mit Isomorphismus σ) wenn

$$\mathcal{H} = \sigma\mathcal{G}\sigma^{-1}.$$

Es ist kein polynomialer Algorithmus bekannt, um zu entscheiden, ob zwei Graphen isomorph sind. Allerdings handelt es sich um ein etwas exotisches Beispiel eines schwierigen Entscheidungsproblems, weil für fast alle Paare von Graphen die Isomorphiefrage einfach zu entscheiden ist. Es ist sogar kompliziert, Graphenpaare zu konstruieren, für welche die Isomorphie nicht einfach zu entscheiden ist. Im Folgenden nehmen wir aber an, es mit solchen Graphenpaaren zu tun zu haben.

In Bezug auf einen Beweis verhält sich das GI-Problem wie das Problem der Hamiltonschen Kreise. Falls die Graphen isomorph sind, gibt es einen einfachen Beweis, nämlich die entsprechende Permutation, falls sie aber nicht isomorph sind, so gibt es im allgemeinen Fall (vermutlich) keinen effizienten und kurzen Beweis.

Im Folgenden beschreiben wir einen interaktiven Beweis für Graphenisomorphismus, bei dem Peggy keine Information über den Isomorphismus weggibt (siehe Abb. 1.3). Zwei Graphen $\mathcal{G}$ und $\mathcal{H}$ sind bekannt, und Peggy kennt einen Isomorphismus σ. Zuerst wählt Peggy eine zufällige Permutation π, permutiert $\mathcal{G}$ mit π zum Graphen $\mathcal{T} = \pi\mathcal{G}\pi^{-1}$, und sendet $\mathcal{T}$ an Vic. Vic wählt ein zufälliges Challenge-Bit c und sendet es an Peggy zurück, die nun je nach Bit den Isomorphismus von $\mathcal{T}$ zu $\mathcal{G}$ (im Fall $c = 0$) oder zu $\mathcal{H}$ (im Fall $c = 1$) öffnen muss. Vic prüft die Korrektheit dieses Isomorphismus. Dieses Protokoll wird s mal wiederholt (z.B. $s = 50$), damit die Betrugswahrscheinlichkeit (WSK, dass Vic akzeptiert, obwohl Peggy σ nicht kennt)

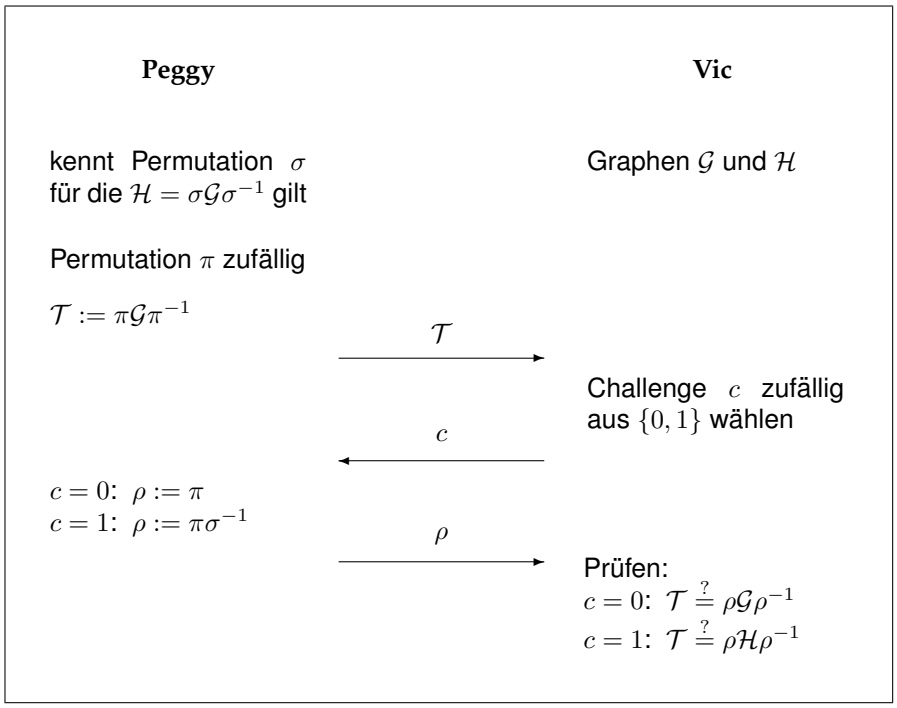


Abb. 1.3: Eine Runde des interaktiven Beweises für Graphenisomorphismus.

genügend klein wird.

Wenn Peggy σ kennt, dann akzeptiert Vic mit Wahrscheinlichkeit 1 (Vollständigkeit). Um die Sicherheit zu analysieren, müssen wir sowohl Peggy's wie auch Vic's Sicht betrachten. Wir müssen also einerseits untersuchen, ob Peggy ohne Kenntnis von σ Vic nicht zum Akzeptieren bringen kann (Widerspruchsfreiheit), und andererseits wollen wir uns überzeugen, dass Peggy keine Information an Vic weggibt (zero-knowledge).

- **Sicherheit für Vic.** Warum soll Vic überzeugt sein, dass (1) Peggy einen Isomorphismus σ kennt, respektive dass (2) die Graphen isomorph sind? Diese beiden Fälle unterscheiden sich subtil: (1) impliziert zwar (2), aber (2) könnte wahr sein, ohne dass (1) stimmt. Wir können wie folgt argumentieren: Jemand, der sowohl den Challenge $c = 0$ als auch den Challenge $c = 1$ beantworten kann, der kann aus den beiden Antworten σ direkt berechnen. Wenn σ nicht existiert (d.h. die Graphen nicht isomorph sind, dann ist die Betrugswahrscheinlichkeit für s Runden höchstens 2^{-s}.

  Diese Begründung, dass jemand, der σ nicht kennt, höchstens mit Wahrscheinlichkeit 2^{-s} betrügen kann, ist subtiler (und stimmt so nucht). Tatsächlich ist die Erfolgswahrscheinlichkeit eines Betrügers leicht höher als 2^{-s}, da er bei einem für ihn ungünstigen Challenge immer noch eine (sehr kleine) Chance hat, die Antwort richtig zu erraten. Dies illustriert, dass präzise Definitionen und eine exakte Analyse notwendig sind (siehe Abschnitt 1.7).

  Es ist wichtig, dass Vic den Challenge zufällig wählt. Wenn Peggy nämlich c wüsste, bevor sie $\mathcal{T}$ senden muss, dann könnte sie sich bei der Wahl von $\mathcal{T}$ darauf vorbereiten. Sie könnte ρ zufällig wählen und im Fall $c = 0$ (resp. $c = 1$) $\mathcal{T} = \rho\mathcal{G}\rho^{-1}$ (resp. $\mathcal{T} = \rho\mathcal{H}\rho^{-1}$) setzen und als erste Nachricht des Protokolls senden.

- **Sicherheit für Peggy.** Warum gibt Peggy keine Information weg? Informell beschrieben deshalb, weil Vic sich die gleiche Kommunikation (d.h. Tripel $\langle\mathcal{T}, c, \rho\rangle$, die im Protokoll von Abb. 1.3 mit Peggy auftreten) selbst erzeugen könnte, und zwar mit der identischen Wahrscheinlichkeitsverteilung wie in einer Ausführung des Protokolls mit Peggy. Dass dies möglich ist, folgt aus der Tatsache, dass sich Vic auf jeden der möglichen Challenges $c = 0$ oder $c = 1$ vorbereiten kann (aber nicht auf beide). Er

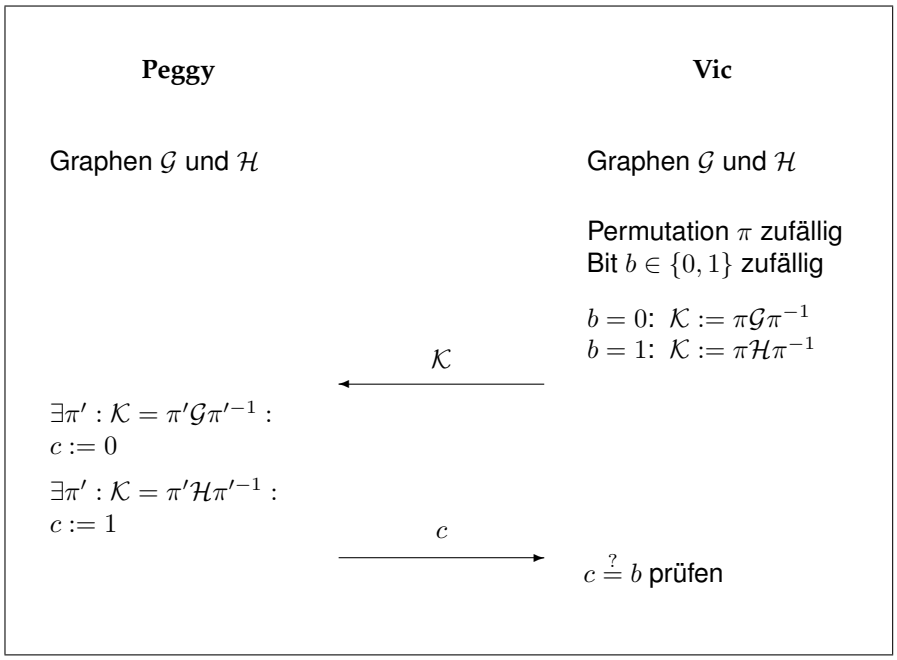


Abb. 1.4: Eine Runde des interaktiven Beweises für Graphennichtisomorphismus.

kann also c zufällig wählen und dann passende aber sonst zufällige $\mathcal{T}$ und ρ berechnen. Wenn er s solche Tripel erzeugt, so ist die Wahrscheinlichkeitsverteilung dieser Tripel identisch mit derjenigen der gesamten Kommunikation im echten Protokoll. Deshalb kann die echte Kommunikation keine Information für Vic enthalten. Dies ist eine informelle Definition von "zero-knowledge".

Allerdings hat diese Argumentation das Problem, dass die Verteilung natürlich unterschiedlich wäre, wenn Vic die Challenges nicht gemäss Protokoll (d.h. zufällig), sondern z.B. abhängig von der vorherigen Kommunikation des Protokolls wählt. Um zu beweisen, dass das Protokoll zero-knowledge ist, muss man zeigen, dass selbst ein betrügerischer Vic die gesamte Kommunikation, welche er in einem Protokoll mit Peggy sehen würde, simulieren könnte (siehe Abschnitt 1.6).

1.2.2 Graphennichtisomorphismus (GNI)

Gegeben seien zwei Graphen $\mathcal{G}$ und $\mathcal{H}$, für welche die Isomorphie nicht einfach entschieden werden kann. Peggy möchte Vic beweisen, dass $\mathcal{G}$ und $\mathcal{H}$ nicht isomorph sind. Wie schon erwähnt, scheint dies nicht-interaktiv nicht effizient möglich zu sein.

Das interaktive Protokoll, bestehend aus einer s-fachen Wiederholung (z.B. $s = 50$) des folgenden Teilprotokolls (siehe Abb. 1.4), löst dieses Problem. Die Verifikation ist für Vic effizient möglich. Dieses Protokoll ist ein Beweis einer Aussage, kann aber nicht als Beweis von Wissen interpretiert werden: Zuerst, wählt Vic zufällig ein Bit b und eine zufällige Permutation π auf n Werten ($n!$ Möglichkeiten), und berechnet einen Graphen $\mathcal{K}$ wie folgt: Im Fall $b = 0$ als die entsprechende Permutation von $\mathcal{G}$ (d.h. $\mathcal{K} = \pi G\pi^{-1}$) und im Fall $b = 1$ als die entsprechende Permutation von $\mathcal{H}$ (d.h. $\mathcal{K} = \pi\mathcal{H}\pi^{-1}$). Vic sendet $\mathcal{K}$ an Peggy. Da $\mathcal{G}$ und $\mathcal{H}$ nicht isomorph sind, kann Peggy entscheiden, ob $b = 0$ oder $b = 1$. Peggy sendet ein Bit c an Vic (wobei $c = b$ sein soll). Vic verifiziert, ob $c = b$. Vic akzeptiert den Beweis, wenn Peggy in allen s Runden richtig antwortet.

Falls die Aussage stimmt, akzeptiert Vic mit Wahrscheinlichkeit 1. Falls die Aussage nicht stimmt, kann Peggy selbst mit unendlichen Computerressourcen nicht erreichen, dass Vic mit Wahrscheinlichkeit $> 1/2^s$ akzeptiert. Wenn nämlich die Graphen isomorph sind, dann ist $\mathcal{K}$ statistisch unabhängig von b, und Peggy hat keine bessere Strategie, als b zu erraten.

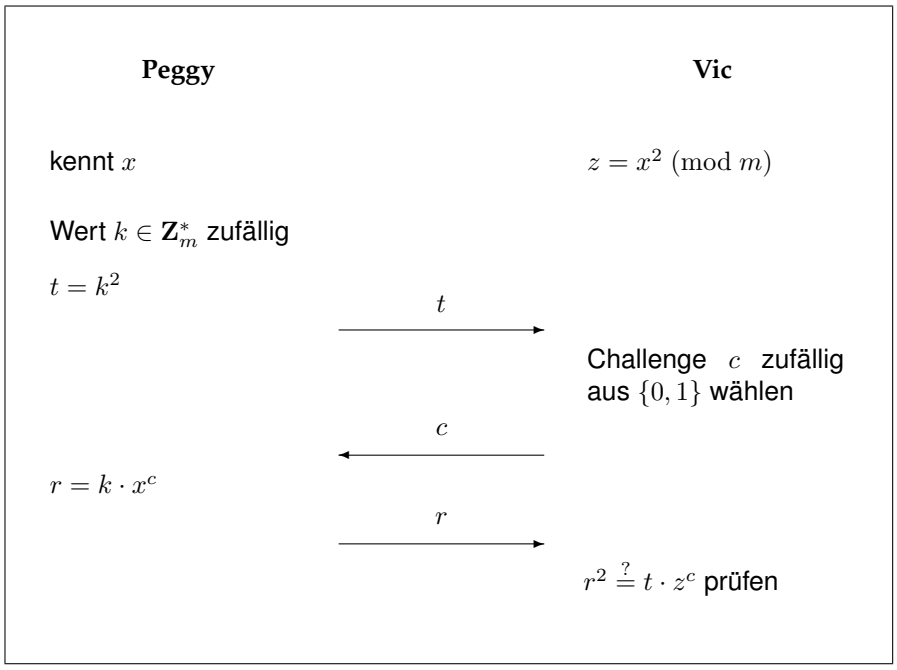


Abb. 1.5: Eine Runde des Fiat-Shamir Protokolls.

Dieses Protokoll ist aber nicht zero-knowledge. Angenommen, es gibt einen Graphen $\mathcal{K}$, von dem Vic nicht weiss, ob er zu $\mathcal{G}$ oder $\mathcal{H}$ isomorph ist, so kann er von Peggy die Antwort erfahren. Er erhält also Information, die er nicht selbst erzeugen kann. Ob diese Information ihm aber etwas nützt, ist natürlich eine ganz andere Frage.

Speziell an diesem Protokoll ist, dass Peggy das GI-Problem lösen können muss, um das Protokoll durchführen zu können. Für die Analyse der Frage, ob es für ein bestimmtes Problem einen interaktiven Beweis gibt oder nicht, nimmt man aber wie schon erwähnt an, der Beweiser sei rechenmässig nicht beschränkt. Selbstverständlich müssen die in der Praxis verwendeten Protokolle aber sowohl für Peggy als auch für Vic effizient sein.

1.2.3 Quadratwurzeln modulo m: das Fiat-Shamir Protokoll

Das in Abb 1.5 dargestellte Protokoll wurde 1986 von Fiat und Shamir [FS86] vorgeschlagen. Sei m ein RSA-Modulus[6]. Das Fiat-Shamir Protokoll erlaubt Peggy zu beweisen, dass sie mind. eine Quadratwurzel x (modulo m) einer gegebenen Zahl z kennt ($z = x^2 \pmod m$). Dies impliziert auch, dass z ein quadratischer Rest modulo m ist; das Protokoll ist demnach auch ein Beweis einer Aussage.

Quadratwurzeln ziehen modulo m ist schwierig, und zwar äquivalent zum Faktorisieren von m, wovon sich der Leser als Übungsaufgabe überzeugen kann. Fiat und Shamir schlugen die Verwendung des Protokolls als Identifikationsprotokoll vor. Peggy's Public Key ist z und ihr geheimer Schlüssel ist x, und sie kann sich authentisieren, indem sie ihre Kenntnis von x beweist. Diese Anwendung eines interaktiven Beweises wird in Abschnitt 1.3.1 in einem allgemeineren Kontext diskutiert.

Zuerst sendet Peggy einen Wert $t = k^2$, wobei sie k zufällig aus $\mathbf{Z}_m^*$ wählt. Anschliessend wählt Vic als Challenge ein zufälliges Bit c. Peggy muss $r = k$ senden, falls $c = 0$ und $r = k \cdot x$, falls $c = 1$. Vic prüft, ob $r^2 = t \cdot z^c$. Dieses Protoll wird s mal wiederholt (z.B. $s = 50$).

Die Analyse dieses Protokolls ist analog zur Analyse des Protokolls für Graphenisomorphismus. Wenn Peggy x kennt, dann akzeptiert Vic mit Wahrscheinlichkeit 1. Vic ist überzeugt, dass Peggy den Wert x kennt, weil jemand, der sowohl den Challenge $c = 0$ als auch den Challenge $c = 1$ beantworten kann, aus

[6] $m = ab$ mit a,b prim

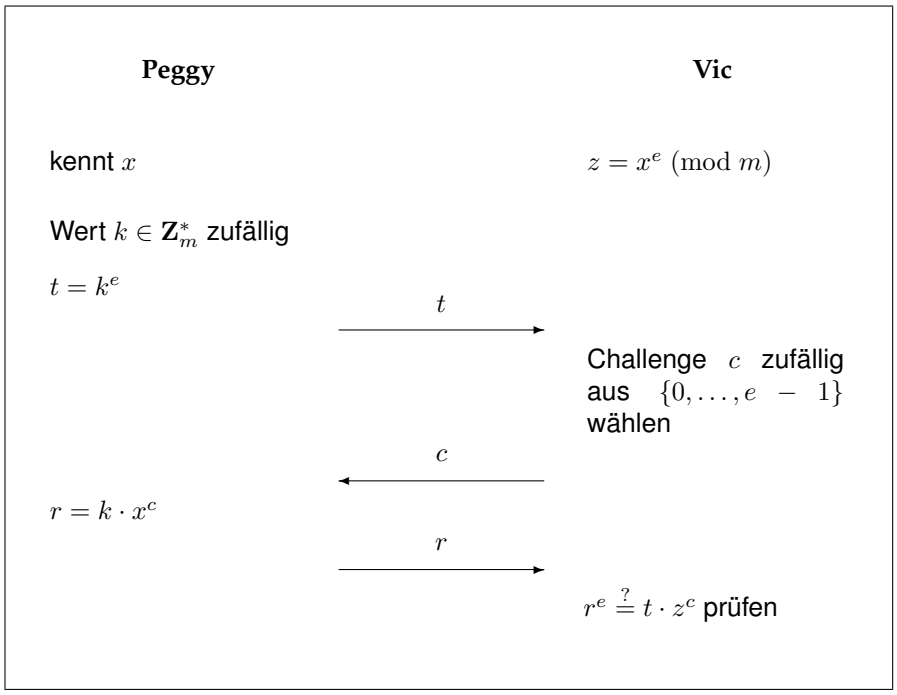


Abb. 1.6: Das Guillou-Quisquater Protokoll.

den beiden Antworten x direkt berechnen kann (x ist der Quotient der beiden Antworten). Deshalb kann jemand, der x nicht kennt, höchstens einen der beiden Challenges beantworten. Ein Betrüger wird also mit Wahrscheinlichkeit $1/2$ erwischt, und die Wahrscheinlichkeit, dass ein Betrüger s Runden des Protokolls unentdeckt übersteht, ist vernachlässigbar.[7]

Es ist wichtig, dass Vic den Challenge zufällig wählt. Wenn Peggy nämlich c wüsste, bevor sie t senden muss, dann könnte sie sich bei der Wahl von t darauf vorbereiten. Es ist dem Leser überlassen, dies zu verifizieren. Ebenso kann sich der Leser als Übung informell überlegen, warum dieses Protokoll zero-knowledge ist.

1.2.4 Höhere Wurzeln modulo m: das Guillou-Quisquater-Protokoll (GQ)

Das in Abb 1.6 dargestellte Protokoll kann als Variante des Fiat-Shamir Protokolls betrachtet werden, in dem der Challenge aus dem grösseren Bereich $\{0, \ldots, e-1\}$ gewählt werden kann. Somit ist die Betrugswahrscheinlichkeit pro Runde nur $1/e$ statt $1/2$ (siehe Analyse unten) und das Protokoll ist effizienter, da weniger Runden (z.B. bei grossem e nur eine) benötigt werden, um eine genügend kleine Betrugswahrscheinlichkeit zu erreichen.

Das Protokoll erlaubt Peggy zu beweisen, dass sie eine e-te Wurzel x (modulo m) einer gegebenen Zahl z kennt ($z = x^e \pmod m$), wobei e mit Vorteil eine Primzahl ist. Wie beim Fiat-Shamir Protokoll ist z Peggy's Public Key, ihr geheimer Schlüssel ist x, und sie kann sich identifizieren, indem sie ihre Kenntnis von x beweist. Im Gegensatz zum Fiat-Shamir Protokoll kann dieses Protokoll aber nicht als Beweis einer Aussage (dass z eine e-te Wurzel modulo m besitzt) interpretiert werden, da ja jedes z eine e-te Wurzel besitzt (sofern wie hier angenommen $\mathrm{ggT}(e, \varphi(m)) = 1$ gilt).

Zuerst sendet Peggy einen Wert $t = k^e$, wobei sie k zufällig aus $\mathbf{Z}_m^*$ wählt. Anschliessend wählt Vic als Challenge einen zufälligen Wert c aus $\{0, \ldots, e-1\}$. Peggy muss mit $r = k \cdot x^c$ antworten.

Wenn Peggy x kennt, dann akzeptiert Vic mit Wahrscheinlichkeit 1. Vic ist überzeugt, dass Peggy den Wert x kennt, weil jemand, der für zwei verschiedene Challenges c und c' korrekt antworten kann, x kennt.

[7] Die Aussage, dass die Betrugswahrscheinlichkeit nicht grösser als $1/2^s$ sein kann ist, wie schon früher erwähnt, nicht präzise.

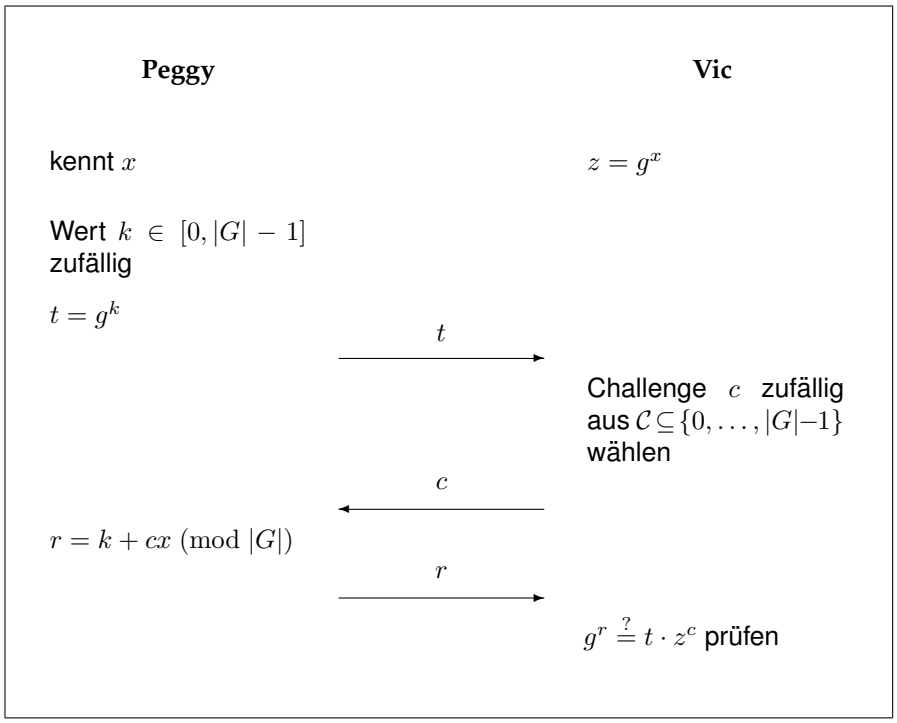


Abb. 1.7: Das Schnorr Protokoll.

Dies kann wie folgt gezeigt werden. Für ein fixiertes t seien r und r' die korrekten Antworten für c und c', d.h. $r^e = t \cdot z^c \pmod m$ und $r'^e = t \cdot z^{c'} \pmod m$. Es gilt

$$\frac{r}{r'} \equiv x^{c-c'} \pmod m.$$

Neben $x^{c-c'}$ kennt man noch eine zweite Potenz von x, nämlich $z = x^e$. Wenn e prim ist, dann sind $c - c'$ und e teilerfremd. Aus zwei verschiedenen, teilerfremden Potenzen eines Gruppenelements x kann man x berechnen. Die Anwendung des erweiterten Euklid'schen ggT-Algorithmus liefert Zahlen a und b, für die $a(c - c') + be = 1$ gilt. Deshalb gilt:

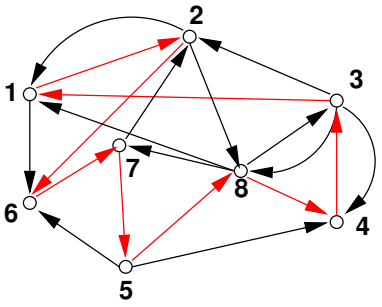
$$x = \left(\frac{r}{r'}\right)^a \cdot z^b \pmod m.$$

1.2.5 Diskrete Logarithmen und das Schnorr-Protokoll

Wir betrachten eine Gruppe G, deren Ordnung $|G|$ prim ist, und in der die Berechnung diskreter Logarithmen schwierig ist. Peggy möchte beweisen, dass sie den DL x eines Elementes z (z.B. ihr Public Key) zur Basis g kennt (d.h. $z = g^x$). Das Protokoll in Abb. 1.7 wurde von Schnorr vorgeschlagen [Sch89]. Da die Gruppe zyklisch ist besitzt jedes Element einen diskreten Logarithmus zur Basis g. Dieses Protokoll ist also ein Beweis von Wissen, kann aber nicht als Beweis einer Aussage interpretiert werden.

Zuerst sendet Peggy einen Wert $t = g^k$, wobei sie k zufällig aus $\{0, \ldots, |G| - 1\}$ wählt. Anschliessend wählt Vic als Challenge einen Wert c aus einem bestimmten Teilbereich $\mathcal{C}$ von $\{0, \ldots, |G| - 1\}$ (z.B. $\mathcal{C} = \{0, \ldots, |G| - 1\}$). Peggy muss $r = k + cx \pmod{|G|}$ senden. Vic verifiziert, ob $g^r = z^c \cdot t$.

Die Analyse dieses Protokolls ist wiederum ähnlich wie die bisherigen Betrachtungen. Wenn Peggy x kennt, dann akzeptiert Vic mit Wahrscheinlichkeit 1. Vic ist überzeugt, dass Peggy den Wert x kennt, weil jemand, der zwei verschiedene Challenges c und c' richtig beantworten kann, aus den beiden Antworten x direkt berechnen kann.



	1	2	3	4	5	6	7	8
1	0	**1**	0	0	0	1	0	0
2	1	0	0	0	0	**1**	0	1
3	**1**	1	0	1	0	0	0	0
4	0	0	**1**	0	0	0	0	1
5	0	0	0	1	0	1	0	**1**
6	0	0	0	0	0	0	**1**	0
7	0	1	0	0	**1**	0	0	0
8	0	0	1	**1**	0	0	1	0

Abb. 1.8: Hamiltonsche Kreise.

Dies kann wie folgt gezeigt werden. Für ein fixiertes t seien r und r' die korrekten Antworten für c und c', d.h. $g^r = t \cdot z^c$ und $g^{r'} = t \cdot z^{c'}$. Es gilt

$$g^{r-r'} = zc - c' = g^{x(c-c')}$$

und deshalb

$$r - r' \equiv x(c - c') \pmod{|G|},$$

woraus wir

$$x \equiv \frac{r_1 - r_2}{c_1 - c_2} \pmod{|G|}$$

erhalten. Damit diese Berechnung immer möglich ist, muss $|G|$ prim sein.

Ein Beweiser ohne Kenntnis von x hat nur mit Wahrscheinlichkeit $1/|\mathcal{C}|$ Erfolg, was bei grossem $|\mathcal{C}|$ vernachlässigbar ist.[8]

Wie bei den früher diskutierten Protokollen kann sich jemand, der x nicht kennt, auf jeden Challenge c vorbereiten. Deshalb ist das Protokoll zero-knowledge, falls der Verifizierer garantiert korrekt spielt, im allgemeinen ist dieses Protokoll aber nicht zero-knowledge, wie wir in Abschnitt 1.6 sehen werden.

1.2.6 Hamiltonsche Kreise (HK)

In diesem Abschnitt betrachten wir als weiteres Beispiel den zero-knowledge Beweis für die Aussage, dass ein gegebener gerichteter Graph einen Hamiltonschen Kreis (HK) besitzt, also einen geschlossenen Pfad, der jeden Knoten des Graphen genau einmal besucht (siehe Abb. 1.8). Dieses Protokoll ist auch ein Beweis von Wissen, da Peggy beweist, dass sie den HK kennt.

Ein HK entspricht in der $n \times n$ Adjazenzmatrix von $\mathcal{G}$ einer Konstellation von n Einsen (die n Kanten des HK), wobei in jeder Zeile und jeder Kolonne je eine der Einsen vorkommt. (Diese Bedingung allein genügt aber noch nicht. Der Leser kann sich als Übungsaufgabe überlegen, welche zusätzliche Bedingung erfüllt sein muss.)

Dieses Beispiel ist speziell interessant, weil das Entscheidungsproblem, ob ein Graph einen HK besitzt **NP**-vollständig ist. Das heisst, dass sich jede Instanz eines **NP**-Entscheidungsproblems effizient (d.h. in polynomialer Zeit) auf einen Graphen abbilden lässt, so dass die Lösung des Entscheidungsproblems genau dann "ja" ist, wenn der entsprechende Graph einen HK besitzt. Ein zero-knowledge Beweis für HK kann also als zero-knowledge Beweis für jedes **NP**-Problem verwendet werden. Ein solcher Beweis wäre zwar ziemlich ineffizient, aber immerhin polynomial. In Abschnitt 1.8 wird ein effizienteres Protokoll für den Beweis jeder **NP**-Aussage besprochen.

Dieses Beispiel erlaubt uns auch, eine neue wichtige kryptographische Primitive, *Bit-Commitments*, informell einzuführen (siehe Abschnitt 1.4 für eine Definition). Bit-Commitments werden in vielen zero-knowledge Beweisen benötigt. Die bisherigen Beispiele sind Spezialfälle, da sie auf Grund der im Problem

[8] Wiederum ist zu bemerken, dass diese Aussage nicht präzise ist (siehe Abschnitt 1.7).

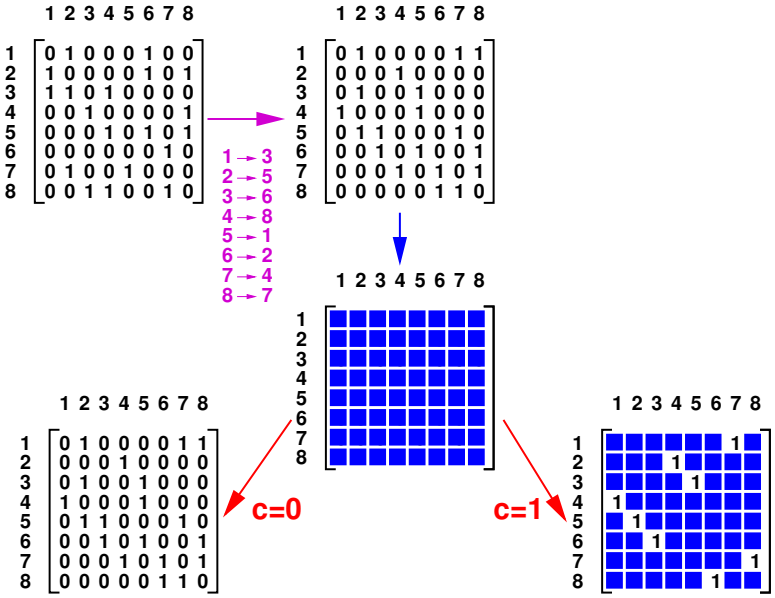


Abb. 1.9: Illustration des Protokolls für Hamiltonsche Kreise.

vorhandenen mathematischen Struktur ohne Bit-Commitments auskommen. In diesem Abschnitt verwenden wir für die Beschreibung die physikalische Realisierung von Bit-Commitments mittels Klebband. Die digitale Realisierung wird im nächsten Abschnitt behandelt.

Das folgende Protokoll (siehe Abb. 1.9) wird s mal wiederholt.

1. Vic schaut weg und Peggy erzeugt durch eine zufällige Permutation σ einen Graphen $\mathcal{H} = \sigma\mathcal{G}\sigma^{-1}$ aus $\mathcal{G}$, schreibt die entsprechende Adjazenzmatrix auf den Tisch und klebt die n^2 Bits mit undurchsichtigen Klebstreifen ab.

2. Vic darf nun hinschauen und wählt ein Challengebit c zufällig. Falls $c = 0$ muss Peggy den ganzen Graphen aufdecken und die Permutation σ angeben. Falls $c = 1$ muss Peggy nur die n Stellen der Adjazenzmatrix aufdecken, die dem HK entsprechen.

3. Vic verifiziert im Fall $c = 0$, ob die Permutation, angewandt auf $\mathcal{G}$ wirklich $\mathcal{H}$ ergibt, und im Fall $c = 1$, ob die aufgedeckten Einsen wirklich einen HK bilden.

Vic akzeptiert, wenn die Prüfung in allen s Runden erfolgreich ist.

Die Analyse des Protokolls ist wieder analog zur Analyse des GI-Protokolls und des Fiat-Shamir Protokolls. Jemand, der den HK nicht kennt, kann eine Runde des Protokolls nur mit Wahrscheinlichkeit $1/2$ erfolgreich bestehen, weil aus den Antworten auf beide möglichen Challenges der HK berechnet werden kann.

Das Protokoll ist (informell) zero-knowledge, weil Vic sich die gesamte Kommunikation selbst erzeugen kann. Er kann nämlich sowohl für $c = 0$ wie für $c = 1$ eine konsistente und zufällige Kommunikation erzeugen. Allerdings ist diese Analyse subtil, wie sich zeigen wird, weil noch nicht klar ist, wie Klebband simuliert wird. Im Fall $c = 1$ kann Vic z.B. nicht einfach einen korrekt permutierten Graphen hinschreiben und dann einen HK öffnen, dazu müsste er ja den HK kennen. Er kann in diesem Fall aber z.B. eine Matrix bestehend aus lauter Einsen hinschreiben und anschliessend einen beliebigen HK „öffnen“.

1.3 Zwei Anwendungen interaktiver Beweise

Interaktive Beweise sind nicht nur theoretisch von grossem Interesse, sondern auch für die Praxis. In diesem Abschnitt disktutieren wir zwei verschiedene Anwendungsbereiche. Ein dritter Anwendungsbereich, sichere Multi-Party Berechnungen, wird im nächsten Kapitel besprochen.

1.3.1 Interaktive Beweise als Identifikationsprotokolle

Ein Identifikationsprotokoll erlaubt einer Person oder Entität Peggy, sich gegenüber einer anderen Entität Vic als Peggy auszuweisen. Grundvoraussetzung ist, dass Vic einen authentischen Referenzparameter besitzt, der in irgend einem Sinn Peggy zugeordnet ist. Bei einer Identifikation über eine Kommunikationsleitung ist der Referenzparameter digital. Es kann ein gemeinsamer geheimer Schlüssel oder ein Public Key sein, zu dem Peggy den geheimen Schlüssel kennt. Der Vorteil eines Public-Key basierten Identifikationsverfahrens ist, dass der gleiche Public Key für beliebige Anwendungen verwendet werden kann und im Prinzip als universeller digitaler Repräsentant von Peggy dienen kann.

Ein solches Verfahren kann man aus jedem Public-Key Kryptosystem konstruieren, indem Peggy eine mit ihrem Public Key verschlüsselte Challenge-Nachricht korrekt entschlüsseln muss, und dadurch die Kenntnis des geheimen Schlüssels beweist. Ein digitales Signaturverfahren kann ebenfalls verwendet werden, wobei Peggy eine Challenge-Nachricht signieren muss.

Um diese Idee umzusetzen, braucht es aber nicht notwendigerweise ein Public-Key Kryptosystem oder ein digitales Signaturverfahren. Es genügt weniger kryptographische Funktionalität und damit in der Regel auch weniger mathematische Struktur (siehe aber Abschnitt 1.3.2). Dies zu erreichen ist, wie schon im Abschnitt 1.2.3 (Fiat-Shamir) erwähnt, eine wichtige Anwendung interaktiver Beweise. Eine Benutzerin Peggy kann als Public Key eine Instanz eines schwierigen Problems (z.B. einen Graphen, in welchem sie einen Hamiltonschen Kreis kennt) generieren und sich später durch Kenntnis des zugehörigen Private Keys (d.h. z.B. des Hamiltonschen Kreises) ausweisen. Wenn das Protokoll zero-knowledge ist, bleibt der Private Key auch nach beliebig häufiger Identifikation gleich sicher wie zu Beginn.

Im Gegensatz dazu sind Protokolle, die auf einem Public-Key Kryptosystem oder einem digitalen Signaturverfahren beruhen, nicht zero-knowledge, wie einfach zu sehen ist. Muss z.B. ein Benutzer eine Challengemeldung unterschreiben, so kann sich der Verifizierer die Signatur nicht selbst erzeugen, erfährt also im Verlauf des Protokolls etwas, das er noch nicht kannte.

Die beschriebene Idee, einen interaktiven Beweis als Identifikationsprotokoll zu verwenden, ist allgemein. Die Kunst beim Entwurf solcher Protokolle ist, einerseits hohe Effizienz zu erreichen und andererseits auf einem Berechnungsproblem zu beruhen, das mit guten Gründen als schwierig angenommen werden kann. **NP**-Vollständigkeit ist keine Garantie für die Sicherheit eines solchen Protokolls, weil es nur eine Aussage über die schwierigsten (aber vielleicht sehr selten vorkommenden) Instanzen des Problems ist.

Beispiel 1.2. Die erste praktische Umsetzung dieser Idee wurde 1986 von Fiat und Shamir vorgeschlagenen (siehe Abschnitt 1.2.3). Der Modulus m kann z.B. als RSA-Modulus gewählt werden. Das Fiat-Shamir Protokoll besitzt gegenüber einem auf dem RSA-Systems basierenden Identifikationsprotokoll zwei wesentliche Vorteile. Es ist wesentlich effizienter als eine RSA Entschlüsselung, und es ist zero-knowledge. Es gibt verschiedene Varianten des Fiat-Shamir Protokolls. In der beschriebenen Version müssen die Faktoren von m niemand bekannt sein. Man kann also annehmen, der Modulus m sei auf irgend eine Art entstanden, z.B. durch eine vertrauenswürdige Instanz erzeugt worden, die anschliessend p und q endgültig gelöscht hat.

1.3.2 Digitale Signaturen mittels interaktiver Beweise

In Abschnitt 1.3.1 diskutierten wir, dass ein Public-Key basiertes Identifikationsverfahren nicht notwendigerweise die Funktionalität und Struktur eines Public-Key Kryptosystems oder eines digitalen Signaturverfahrens erfordert. In diesem Abschnitt zeigen wir aber umgekehrt, dass aus einem interaktiven Beweis ein digitales Signaturverfahren konstruiert werden kann. Diese einfache Idee ist in [FS86] beschrieben und

wird Fiat-Shamir Heuristik genannt. Im Gegensatz zu einem digitalen Signaturverfahren scheint es aber unmöglich, ein Public-Key Kryptosystem aus einem interaktiven Beweis zu konstruieren.

Betrachten wir den Fall, dass Peggy nicht-interaktiv, d.h. ohne die Anwesenheit eines Verifizierers, einen Beweis erzeugen möchte, der anschliessend von jedem Verifizierer ohne Interaktion mit Peggy akzeptiert wird, wobei Peggy das Geheimnis durch den Beweis nicht weggibt. Ein solcher Beweis könnte z.B. verwendet werden, um ein mathematisches Theorem zu beweisen, ohne den Beweis wegzugeben. Natürlich[9] kann ein solcher Beweis nicht zero-knowledge sein.

Wie kann also ein interaktiver Beweis nicht-interaktiv gemacht werden? Das Problem ist, dass Peggy die Challenges selbst wählen muss. Wenn das interaktive Protokoll zero-knowledge ist, dann kann ein solcher Beweis mit selbst erzeugten Challenges keine Überzeugungskraft besitzen, weil ja jedermann einen solchen Beweis erzeugen kann.

Dieses Dilemma besitzt eine Lösung. Die Rundenzahl s wird so gewählt, dass die Liste aller k Challenges höchstens mit vernachlässigbarer Wahrscheinlichkeit erraten werden kann. Bei einem Protokoll mit binären Challenges muss also $k \approx 100 \ldots 200$ sein, während beim Schnorr-Protokoll $s = 1$ genügt. Der nicht-interaktive Beweis wird wie folgt erzeugt:

1. Peggy generiert für $i = 1, \ldots, s$ die erste Nachricht t_i der i-ten Runde.
2. Peggy erzeugt die s Challenges, indem sie eine kryptographisch sichere Hashfunktion h auf die Liste der in Schritt 1 erzeugten Nachrichten anwendet: $[c_1, \ldots, c_s] = h(t_1, \ldots, t_s)$
3. Peggy generiert die s Antworten $r_1, \ldots, r_s$ für diese Challenges.

Dieses Verfahren ist (zumindest heuristisch) sicher, weil Peggy die Challengewerte $c_1, \ldots, c_s$ nicht genügend beeinflussen kann, wenn die Hashfunktion nicht eine spezielle ausnützbare Struktur besitzt. Deshalb kann Peggy ohne Kenntnis des Geheimnisses mit sehr grosser Wahrscheinlichkeit die Antworten in Schritt 3 nicht berechnen. Die präzise Eigenschaft der Hashfunktion, die benötigt ist, um ein solches Signaturverfahren sicher zu machen, hängt vom interaktiven Beweis ab.

Eine oft hinreichende Eigenschaft wäre, dass sich die Hashfunktion wie eine zufällige Funktion (ein „Random Oracle") verhält. Die Sicherheit vieler kryptographischer Systeme kann unter einer solchen plausibel erscheinenden Annahme bewiesen werden.[10] Eine solche idealisierte Welt wird *Random Oracle Modell* genannt. Man kann aber zeigen, das diese Eigenschaft einer Hashfunktion (oder genauer: einer Klasse von Hashfunktionen) nicht erreichbar ist, auch in einem berechenmässigen Sinn nicht.[11]

Es bleibt noch zu beschreiben, wie dieser Beweis als digitales Signaturverfahren verwendet werden kann. Dies ist einfach möglich, indem die zu signierende Meldung m als zusätzliches Argument der Hashfunktion verwendet wird: $[c_1, \ldots, c_s] = h(t_1, \ldots, t_s, m)$. Wie schon erwähnt, ist die Sicherheit eines solchen Verfahrens nur heuristisch, d.h. sie beruht nicht beweisbar auf einem wohldefinierten Berechnungsproblem. Für die Praxis sind aber solche Verfahren sehr attraktiv, erstens weil sie sehr effizient sein können und zweitens, weil sich sehr viele neue Alternativen bieten für die Wahl des zu Grunde liegenden (vermutlich schwierigen) Berechnungsproblems. Dies ist eine echte Alternative zum RSA-System und zum Schnorr-System, deren Sicherheit auf der Schwierigkeit des Faktorisierungs- resp. des diskreten Logarithmus-Problems beruhen.

1.4 Commitment-Verfahren

1.4.1 Das Konzept

In diesem Abschnitt geht es um digitale Versionen der in Abschnitt 1.2.6 beschriebenen Klebbandtechnik mit selektivem Aufdecken. Mit einem sogenannten *(Bit-)Commitment-Verfahren* kann Peggy sich zu einem

[9] Im *shared random string model*, siehe [BFM88], sind jedoch nicht-interaktive zero-knowledge Beweise möglich.

[10] D.h. es kann bewiesen werden, dass das Brechen des Systems so schwierig ist wie ein bekanntes schwieriges Berechnungsproblem.

[11] Es gibt nämlich ein digitales Signaturverfahren, das im Random Oracle Modell bewiesenermassen sicher ist (basierend auf einer kryptographischen Annahme), so dass aber für jede Instantiierung der Hashfunktion das Signaturverfahren unsicher ist.

Wert x (oft nur ein Bit) verpflichten, ohne ihn bekanntgeben zu müssen. Die Verpflichtung bedeutet, dass sie den Wert x nicht mehr ändern kann. Sie kann aber x jederzeit offenlegen.

Im Kontext dieses Kapitels geht Peggy die Verpflichtung gegenüber Vic ein, indem sie ihm einen Wert b, den sogenannten „Blob", sendet. In einem allgemeineren Kontext könnte sich Peggy aber auch gegenüber einer Menge von Spielern zu einem Wert verpflichten, was vor allem im Kontext der Multi-Party Berechnungen verwendet wird.

Ein mechanisches Analogon eines Commitment Verfahrens ist die oben beschriebene Klebstreifentechnik. Ein anderes Analogon sind Umschläge. Eine Nachricht kann in einem Umschlag verschlossen und auf den Tisch gelegt werden. Der Inhalt ist für Vic nicht sichtbar, kann aber von Peggy auch nicht mehr geändert werden. Diese Vergleiche hinken insofern, als ein betrügerischer Vic einfach das Klebband wegreissen respektive den Umschlag aufreissen kann, obwohl dies gemäss Protokoll nicht erlaubt ist. Im digitalen Fall ist das nicht möglich. Ein vielleicht adäquateres Analogon ist deshalb ein Safe, in den Peggy einen Zettel mit darauf geschriebenem Bit einschliesst und ihn dann Vic übergibt. Später kann sie den Schlüssel zum Öffnen des Safes nachliefern.

Definition 1.1. Ein *Commitment-Verfahren* besteht aus zwei Protokollen COMMIT und OPEN zwischen einem Beweiser P und einem Verifizierer V.[12] Zu jeder Instanz von COMMIT gehört eine entsprechende Instanz von OPEN. Die Ausführung jedes dieser Protokolle kann von V entweder akzeptiert oder verworfen werden. Für COMMIT hat P einen Input x (aus einem bestimmten Bereich $\mathcal{X}$) und als Ergebnis erhält V einen Wert b (den Blob resp. das Commitment) und P einen Wert d. Für OPEN hat P den Input (x, d), und V hat den Input b und erhält als Output einen Wert $x' \in \mathcal{X}$. Dabei müssen folgende Bedingungen erfüllt sein:

- **Korrektheit.** Wenn P und V korrekt spielen, dann akzeptiert V COMMIT und OPEN, und bei OPEN erhält V den von P in COMMIT verwendeten Wert x.
- **Hiding.** Die gesamte Sicht von V in COMMIT (insbesondere der Blob b) gibt keine (relevante) Information über x, selbst wenn V falsch spielt.
- **Binding.** Wenn V das Protokoll COMMIT akzeptiert hat, dann gibt es höchstens einen Wert x', den ein beliebiger unehrlicher $\tilde{P}$ im entsprechenden Protokoll OPEN als Output von V provozieren kann.

Meist besteht das Protokoll OPEN lediglich aus dem Senden von x und d von P an V und dem Prüfen durch V mittels einer Verifikationsfunktion, die x, b und d als Input nimmt und einen binären Output liefert. Im Folgenden betrachten wir nur Commitment-Verfahren dieses Typs.

Beispiel 1.3. Ein erster einfacher Ansatz für ein Commitment-Verfahren ist mittels einer kryptographisch sicheren Hashfunktion h. Peggy wählt einen genügend langen zufälligen Bitstring r (z.B. 100 Bits) und berechnet $b = h(x||r)$ und $d = r$. Dieses Verfahren ist nur heuristisch sicher. Um beweisbar sicher zu sein, müsste h spezielle Eigenschaften haben.

1.4.2 Typen von Sicherheit

Sowohl die Binding Eigenschaft (Sicherheit für Vic) als auch die Hiding Eigenschaft (Sicherheit für Peggy) kann entweder informations-theoretisch oder "nur" kryptographisch (d.h. berechenmässig) garantiert sein. Wir unterscheiden dem entsprechend zwei Typen von Commitment Verfahren:

- **Typ H.** Die Hiding-Eigenschaft ist informations-theoretisch, d.h. alle möglichen Werte x ergeben die gleiche Wahrscheinlichkeitsverteilung für b. Dies bedeutet, dass der Blob b statistisch unabhängig von x ist.
- **Typ B.** Die Binding-Eigenschaft ist informations-theoretisch, d.h. x ist durch b eindeutig bestimmt. Es gibt keine Werte x und $x' \neq x$ sowie d und d', so dass die Verifikationsfunktion sowohl für (x, b, d) als auch für (x', b, d') akzeptiert.

[12] Im Fall von mehreren Verifizierern sind COMMIT und OPEN Protokolle zwischen allen Spielern.

Es ist dem Leser als Übung empfohlen, zu beweisen, dass kein Commitment Verfahren gleichzeitig vom Typ B und vom Typ H sein kann.

Die berechenmässige Hiding-Eigenschaft könnte wie folgt formalisiert werden: für Vic ist die Wahrscheinlichkeit, x richtig zu erraten, mit Kenntnis von b nur unbedeutend grösser als ohne Kenntnis von b.

Die berechenmässige Binding-Eigenschaft bedeutet, dass es selbst für eine unehrliche Peggy berechenmässig zu aufwendig ist, nach einer von Vic akzeptierten Durchführung des COMMIT Protokolls Werte x, x', d und d' zu finden, so dass die Verifikationsfunktion sowohl für (x, b, d) als auch für (x', b, d') akzeptiert.

1.4.3 Ein Commitment Verfahren vom Typ H

Gegeben sei eine zyklische Gruppe G mit Ordnung q und zwei Generatoren g und h, wobei der diskrete Logarithmus von h zur Basis g nicht bekannt ist. (Die Generatoren werden z.B. durch einen Pseudozufallsgenerator erzeugt.) Peggy verpflichtet sich zu einem Wert $x \in \{0, \ldots, q-1\}$, indem sie r zufällig aus $\{0, \ldots, q-1\}$ wählt und den Blob[13]

$$b = g^x h^r$$

berechnet. Es kann durch Senden von x und $d = r$ geöffnet werden. Wegen der zufälligen Wahl von r ist b für jeden Wert von x ein völlig zufälliges Element von G, also statistisch unabhängig von x. Peggy könnte aber betrügen, wenn sie $\log_g h$ in G wüsste oder berechnen könnte. Die Fähigkeit, ein Commitment auf zwei verschiedene Arten zu öffnen, ist äquivalent zur Kenntnis von $\log_g h$, wie man sich als Übungsaufgabe einfach überlegen kann.

1.4.4 Ein Commitment Verfahren vom Typ B

Wir verweisen auf den Anhang 1.A. zu diesem Kapitel für eine Einführung der Konzepte *quadratischer Rest*, *Legendre Symbol* und *Jacobi Symbol*. Hier beschreiben wir ein Bit-Commitment Verfahren vom Typ B, dessen Hiding-Eigenschaft auf dem quadratischen Rest-Problem basiert.

Ein RSA-Modulus $m = pq$ sowie ein quadratischer Nichtrest t modulo m mit $\left(\frac{t}{m}\right) = 1$ seien gegeben.[14] In einer konkreten Anwendung werden diese Parameter entweder von einer vertrauenswürdigen Instanz oder von Peggy generiert. Im zweiten Fall muss Peggy zuerst Vic beweisen, dass t wirklich ein quadratischer Nichtrest ist.

Um sich auf ein Bit $x \in \{0, 1\}$ zu verpflichten wählt Peggy ein Element r zufällig aus $\mathbf{Z}_m^*$ und berechnet das Commitment $b = r^2 t^x$ modulo m. Um das Commitment zu öffnen, gibt sie Vic x und $d = r$.

Dieses Verfahren ist vom Typ B, da $x = 0$ genau dann wenn $b \in \mathrm{QR}_m$: Peggy ist mit dem Wert b eindeutig auf x festgelegt. Vic hingegen könnte betrügen, d.h. Information über x erhalten, falls er das QR-Problem (siehe Anhang 1.A) lösen könnte.

1.5 Interaktive Beweise von Aussagen

1.5.1 Einleitung

Eine Aussage ist eine präzise formulierte Behauptung, die entweder wahr oder falsch ist. In einem mathematischen Kontext werden Aussagen von genügender Allgemeinheit oft Theorem genannt. Ein Beweisverfahren für eine Aussage erlaubt Peggy, Vic zu überzeugen, dass die Aussage wahr ist (Vollständigkeit), aber nur, wenn sie tatsächlich stimmt (Widerspruchsfreiheit).

Ein interaktives Beweisverfahren kann für eine konkrete Aussage aufgesetzt werden (z.B. dass zwei konkrete Graphen isomorph sind), meist werden solche Verfahren aber für ganze Aussagenklassen (z.B.

[13] Es handelt sich hier um Pedersen commitments, vgl. [Ped91].

[14] Falls z.B. $p \equiv q \equiv 3 \pmod 4$, dann ist $t = -1$ $(= m - 1)$ ein quadratischer Nichtrest mit $\left(\frac{t}{m}\right) = 1$.

für die Isomorphie von Graphen) formuliert. Dies wird durch das Konzept der Zugehörigkeit zu einer formalen Sprache formalisiert.

Die Definitionen dieses Abschnitts folgen dem in der Komplexitätstheorie üblichen Muster: sie sind *asymptotisch* und es wird in erster Annäherung nur zwischen *polynomialem* und *nicht polynomialem* Verhalten einer Funktion (z.B. der Laufzeit eines Algorithmus) unterschieden. Dies mag wie in der Komplexitätstheorie etwas willkürlich erscheinen, wirklich bessere Definitionen anzugeben scheint aber sehr schwierig. Ein Grundproblem ist, dass bei nicht-asymptotischer Betrachtung das konkrete Rechenmodell (z.B. Turing-Maschine, resp. PC mit bestimmtem Prozessor und Speicherstruktur) in die Analyse eingehen muss, was sicher unerwünscht ist. Wir verweisen auf Anhang 1.B für eine Einführung der wichtigsten komplexitätstheoretischen Begriffe.

1.5.2 Interaktive Beweise für Sprachenzugehörigkeit: Definition

Der Begriff eines *interaktiven Protokolls* ist in der Literatur als Paar von miteinander kommunizierenden (interaktiven) probabilistischen Turing-Maschinen definiert, die ein gemeinsames Inputband, zwei Kommunikationsbänder (für die Kommunikation in beide Richtungen) und je ein privates Rechenband besitzen. Wir müssen hier nicht einen solch formalen Standpunkt einnehmen und denken uns ein interaktives Protokoll einfach als zwei Programme P und V, die miteinander kommunizieren können. Die Programme können probabilistisch sein, d.h. (geheime) zufällige Bits verwenden. In einem interaktiven Beweis macht V am Schluss einen binären Entscheid, den wir als akzeptieren resp. verwerfen interpretieren.

Definition 1.2. Ein *interaktiver Beweis* für Zugehörigkeit zu einer Sprache L ist ein interaktives Protokoll zwischen einem Beweiser P und einem Verifizierer V mit polynomialer Laufzeit, das die folgenden Bedingungen erfüllt:

(i) **(Vollständigkeit, Completeness)** Für alle $x \in L$ akzeptiert V mit Wahrscheinlichkeit grösser als $3/4$.

(ii) **(Widerspruchsfreiheit, Soundness)** Für alle $x \notin L$ und für alle Programme P' (an Stelle von P) akzeptiert V mit Wahrscheinlichkeit höchstens $1/2$.

Die Wahrscheinlichkeiten $3/4$ und $1/2$ sind willkürlich gewählt. Würden sie ersetzt durch Werte beliebig nahe bei 1 resp. 0, durch "überwältigend" und "verschwindend", oder durch zwei beliebig nahe, aber verschiedene Wahrscheinlichkeiten (z.B. 0.06373 und 0.06372), so wäre die Menge der Sprachen, für die ein interaktiver Beweis existiert, immer noch die gleiche. Dies folgt aus der Tatsache, dass ein Protokoll mehrmals unabhängig wiederholt werden kann. Es darf aber nicht verlangt werden, dass die Wahrscheinlichkeit des falschen Akzeptierens 0 sein muss. Man überzeuge sich, dass in diesem Fall die Interaktion in Bezug darauf, welche Aussagen beweisbar sind, nichts nützt, d.h., dass nur die Zugehörigkeit zu **NP**-Sprachen bewiesen werden kann (was auch ohne Interaktion effizient möglich ist).

Intuitiv betrachtet steht P' für einen beliebigen betrügerischen Beweiser, der eine falsche Aussage beweisen möchte. Man beachte, dass für P und P' keine Beschränkungen der Rechenzeit vorgeschrieben sind. Falls die Widerspruchsfreiheit nur gilt, wenn P' polynomial zeitbeschränkt ist, dann spricht man von einem *interaktiven "Argument"* oder von einem *"computationally sound interactive proof"*. In diesem Fall kann P' also nur deshalb nicht betrügen, weil dazu eine zu schwierige Berechnung notwendig wäre.

Im Gegensatz dazu ist in einem interaktiven Beweis ein Betrug nur mit verschwindend kleiner Wahrscheinlichkeit möglich, unabhängig von den verfügbaren Rechenressourcen. Dieser Unterschied ist wesentlich. Man kann argumentieren, dass ein interaktiver Beweis durchaus auch in einem erweiterten mathematischen Sinn als Beweis angesehen werden kann[15], aber ein interaktives Argument kann sicher nicht als Beweis im mathematischen Sinn angesehen werden.

Beispiel 1.4. Das Protokoll für Graphenisomorphismus ist ein interaktiver Beweis für Zugehörigkeit zur Sprache GI. Die Wahrscheinlichkeit, dass ein Wort $x \in$ GI akzeptiert wird ist 1 (also grösser als $3/4$). Die Wahrscheinlichkeit, dass ein Wort $x \notin$ GI akzeptiert wird, ist höchstens $1/2$. Die gleichen Überlegungen

[15] zumal die Fehlerwahrscheinlichkeit viel kleiner gemacht werden kann als die Wahrscheinlichkeit, dass sich die besten 10 Mathematiker gleichzeitig in der Verifikation eines Beweises irren.

gelten für die Protokolle für Graphennichtisomorphismus, für quadratische Reste (Fiat-Shamir), für Hamiltonsche Kreise, etc.

1.5.3 Die Mächtigkeit interaktiver Beweise und die Sprachenklasse IP

Theorem 1.8 auf Seite 36 impliziert, dass die Zugehörigkeit eines Wortes x zu einer Sprache $L \in \mathbf{NP}$ nicht-interaktiv bewiesen werden kann, wobei der Beweis in polynomialer Zeit verifizierbar ist. Wie schon erwähnt, sind interaktive Beweise aber mächtiger als nicht-interaktive Beweise.

Definition 1.3. **IP** ist die Klasse der Sprachen L, für die ein interaktiver Beweis für Zugehörigkeit in L existiert (gemäss Definition 1.2).

Sicher gilt $\mathbf{NP} \subset \mathbf{IP}$, da nicht-interaktive Beweise ein Spezialfall interaktiver Beweise sind.

Der Leser kann sich selbst überzeugen, dass die Klasse **IP** der interaktiv beweisbaren Sprachen nicht eingeschränkt würde, wenn P deterministisch sein muss. Hingegen helfen probabilistische P, das von P an V weggegebene Wissen zu minimieren (siehe Abschnitt 1.6). Das folgende Resultat wurde von Shamir 1990 bewiesen.

Theorem 1.1. $\mathbf{IP} = \mathbf{PSPACE}$.[16]

Beweis. Im Beweis von $\mathbf{IP} \subseteq \mathbf{PSPACE}$ muss man das Paar (P, V), das ja gemäss Definition für jede Sprache $L \in \mathbf{IP}$ existiert, verwenden, um ein mit polynomialem Speicherplatz (aber exponentieller Zeit) laufendes Programm zu konstruieren, das x akzeptiert genau dann wenn $x \in L$, d.h. wenn die Akzeptanzwahrscheinlichkeit von V grösser als $3/4$ ist. Der Leser kann diesen Beweis als Übungsaufgabe führen. Der Beweis von $\mathbf{PSPACE} \subseteq \mathbf{IP}$ ist bedeutend schwieriger und wird hier nicht gegeben. □

Beispiel 1.5. Die Sprache GNI (Graphennichtisomorphismus) ist in **IP**, aber vermutlich nicht in **NP**.

1.5.4 Beweise von mathematischen Aussagen

In diesem Abschnitt diskutieren wir kurz die Anwendung interaktiver Beweise auf gewöhnliche mathematische Aussagen (z.B. das letzte Theorem von Fermat, das vor einigen Jahren von Wiles bewiesen wurde). Solche interaktiven Beweise sind Beweise des Wissens eines Beweises. (Das Konzept des Beweises von Wissen wird erst in Abschnitt 1.7 formalisiert.)

Für spezifische Aussagen (z.B. dass zwei Graphen isomorph sind, oder dass eine Zahl ein quadratischer Rest modulo m ist) ist klar, woraus ein Beweis besteht und wie er verifiziert wird (z.B. der Isomorphismus oder eine Quadratwurzel). Für allgemeine Aussagen ist nicht a priori klar, woraus ein Beweis besteht und wie er verifiziert wird. Man müsste deshalb einen formalen, auf Logik basierenden Verifikationskalkül für mathematische Aussagen betrachten, in dem ein Beweis eine Folge von elementaren Schritten ist. Ein solcher Kalkül nimmt als Input zwei Strings, eine Aussage und deren angeblichen Beweis, und entscheidet, ob der Beweis korrekt ist. Der Beweiser beweist die Kenntnis eines Beweisstrings, der im formalen Verifikationsverfahren zur Akzeptanz führt.

Bei einem solchen Beweis wird mindestens Information über die Länge des Beweises resp. der Verifikationsrechnung weggegeben. Allerdings könnte dieses Problem entschärft werden, indem alle Beweise künstlich bis zu einer gewissen oberen Schranke verlängert würden.

Beim Beweis einer konkreten mathematischen Aussage wird implizit die Zugehörigkeit der Aussage in der formalen Sprache der beweisbaren, und damit in der Sprache der wahren Aussagen bewiesen. Es existiert aber kein interaktiver Beweis für die Sprache der wahren Aussagen, was aus dem Gödelschen Unvollständigkeitssatz folgt.

[16] **PSPACE** ist die Menge der Sprachen, die von einer Turing-Maschine mit polynomial beschränktem Speicher (aber unbeschränkter Laufzeit) akzeptiert werden.

1.6 Zero-knowledge Protokolle

Zero-knowledge ist eine Eigenschaft, die für ein beliebiges interaktives Protokoll zwischen zwei Programmen P und V definiert werden kann, insbesondere für interaktive Beweise von Aussagen, von Sprachzugehörigkeit, und von Wissen. Zero-knowledge Beweise geben absolut keine Information an V, ausser der Aussage, die bewiesen wird.

Im Folgenden wird es häufig nicht mehr ausreichen über konkrete, in einem bestimmten Protokoll auftretende, Werte zu argumentieren (z.B. Peggy schickt einen Wert t an Vic...). Wir werden daher Zufallsvariablen betrachten, deren WSK-Verteilung durch das Protokoll gegeben ist. Zufallsvariablen werden immer durch einen Grossbuchstaben bezeichnet.

Für einen (in einem Protokoll auftretenden) Wert u bezeichnet der entsprechende Grossbuchstaben U jeweils die Zufallsvariable, deren WSK-Verteilung derjenigen von u im entsprechenden Protokoll entspricht.

1.6.1 Ununterscheidbarkeit von Wahrscheinlichkeitsverteilungen

Um den Begriff zero-knowledge präzise definieren zu können, brauchen wir den Begriff der *Ununterscheidbarkeit* von zwei Wahrscheinlichkeitsverteilungen P und Q, die über der gleichen Menge definiert sind. Wie bisher üblich betrachten wir unendliche, durch einen Parameter n indizierte Familien von WSK-Verteilungen, P_n und Q_n, die für gleiches n über der gleichen Menge $\mathcal{M}_n$ definiert sind. Uns interessiert die Frage, ob man unterscheiden kann, welche der beiden Verteilungen vorliegt.

Ununterscheidbarkeit kann auf zwei verschiedene, aber äquivalente Arten definiert werden. Die gebräuchliche Art ist, dass man Algorithmen betrachtet, die als Input n und ein Element x aus $\mathcal{M}_n$ nehmen und als Ouput ein Bit (0 oder 1) ausgeben.

Definition 1.4. Für einen gegebenen Algorithmus A und fixes n ist die *Unterscheidungswahrscheinlichkeit* von A definiert als

$$\Delta_A(P_n, Q_n) := \left| P_{X \leftarrow P_n}(A(X) = 1) - P_{X \leftarrow Q_n}(A(X) = 1) \right|,$$

wobei $X \leftarrow P_n$ bedeutet, dass X zufällig gemäss der Verteilung P_n gewählt wird.

Definition 1.5. Zwei asymptotische Familien von Verteilungen $\{P_n : n \geq 1\}$ und $\{Q_n : n \geq 1\}$ heissen *berechenmässig ununterscheidbar*, wenn für jeden polynomialen (in n) Algorithmus A die Unterscheidungswahrscheinlichkeit $\Delta_A(P_n, Q_n)$ verschwindend ist.[17] Sie heissen *statistisch ununterscheidbar*, wenn dies für jeden (unbeschränkten) Algorithmus gilt.[18] Sind die Verteilungen P_n und Q_n für alle n identisch, so heissen sie *perfekt ununterscheidbar*.

Berechenmässige Ununterscheidbarkeit bedeutet also, dass man mit beschränktem Rechenaufwand keinen wesentlichen Unterschied zwischen den Verteilungen erkennen kann, obwohl dies eventuell theoretisch mit sehr grossem Aufwand möglich wäre.

Beispiel 1.6. Die Definition eines kryptographisch sicheren Pseudozufallsgenerators (asymptotisch definiert, mit der Länge n des Startwertes als Parameter) ist, dass der Output berechenmässig ununterscheidbar von einer echten Zufallsfolge sein soll. Die Länge l der betrachteten Outputfolge ist polynomial in n (beliebiges Polynom). Die Verteilung P_n ist demnach die Verteilung der l-Bit Strings, die für einen zufälligen n-Bit Startwert generiert werden. Die Verteilung Q_n ist die Gleichverteilung über den l-Bit Strings.

Beispiel 1.7. Für fixes n betrachte man die Menge der n-Bit RSA-Moduli mit zwei $n/2$-Bit Primfaktoren. Ein Modulus m werde zufällig aus dieser Menge gewählt und bekannt gegeben. P_n sei die Gleichverteilung über alle quadratischen Reste modulo m, und Q_n sei die Gleichverteilung über alle Elemente von Z_m^* mit Jacobi-Symbol 1. Es wird vermutet, dass diese zwei Familien von Verteilungen berechenmässig ununterscheidbar sind (wobei die Unterscheidungswahrscheinlichkeit über die Wahl von m gemittelt wird). Dies ist eine präzise Formulierung der Schwierigkeit des QR-Problems (siehe Anhang 1.A).

[17] Diese Definition ist äquivalent dazu, ein Zufallsexperiment zu betrachten, in dem für jedes n eine der beiden Verteilungen (P_n oder Q_n) geheim und zufällig gewählt wird (je mit WSK $1/2$), und zu verlangen, dass für jeden effizienten Algorithmus die WSK zu erraten, welcher Fall (P_n oder Q_n) vorliegt, nur verschwindend grösser als $1/2$ ist.

[18] Dies impliziert, dass P_n und Q_n asymptotisch (schnell) gegeneinander konvergieren.

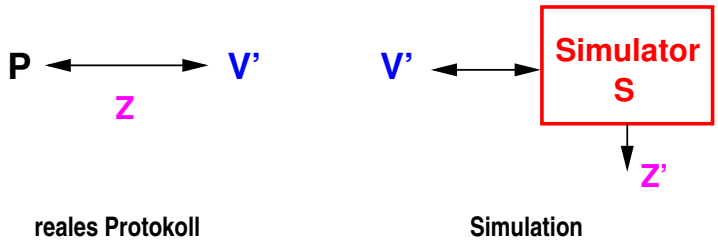


Abb. 1.10: Reales Protokoll mit Transkript Z und simuliertes Protokoll mit Transkript Z'. Wenn Z und Z' ununterscheidbar sind, dann ist das Protokoll mit Prover P und Verifier V' zero-knowledge.

1.6.2 Formale Definition von zero-knowledge Protokollen

Wir wollen definieren, was man unter der zero-knowledge Eigenschaft eines interaktiven Protokolls versteht. Eine minimale Anforderung an ein Protokoll ist, dass Vic nach Durchführung des Protokolls den geheimen Wert von Peggy nicht berechnen kann. Eine stärkere Anforderung wäre, dass Vic den Beweis nicht gegenüber einer anderen Person wiederholen kann, d.h. dass das Transkript des Protokolls niemand anders überzeugt.

Der Begriff zero-knowledge ist noch stärker und bedeutet informell, dass Vic durch die Durchführung des Protokolls *keine* Information erhält, die er nicht schon vorher besass. Wichtig ist, dass dies selbst dann gelten muss, wenn sich Vic beliebig unehrlich verhält, z.B. wenn er seine Challenges (statt zufällig) abhängig vom bisherigen Protokollverlauf wählt. Mit anderen Worten, es wird über alle möglichen Programme V' quantifiziert, die ein unehrlicher Verifizierer verwenden könnte. Einzige Bedingung ist, dass V' effizient sein muss.

Es scheint schwierig zu formalisieren, was es für ein Programm heisst, Information zu besitzen oder nicht zu besitzen. Die folgende elegante Idee ist ein zentraler Punkt in vielen Sicherheitsdefinitionen in der Kryptographie. Ein Verifier erfährt nichts bei der Durchführung des Protokolls, wenn er die gesamte Kommunikation des Protokolls effizient (d.h. in polynomialer Zeit) selbst generieren könnte, ohne mit Peggy zu kommunizieren. Formalisiert wird dies, indem ein sogenanntes Simulatorprogramm existieren muss, das mit dem Verifizierer interagieren kann und die gesamte Kommunikation des Protokolls effizient simuliert, und zwar ununterscheidbar von einer tatsächlichen Protokolldurchführung mit dem echten Beweiser.

Definition 1.6. Ein interaktives Protokoll zwischen einem Beweiser P und einem Verifizierer V ist *zero-knowledge*, wenn es für jedes polynomial begrenzte[19] Programm V' ein polynomial begrenztes Programm S gibt (den *Simulator*, siehe Abb. 1.10), so dass der Output Z' von S ununterscheidbar ist von der gesamten Kommunikation (dem Transkript Z) zwischen P und V' in einer Durchführung des Protokolls. Je nachdem ob die Ununterscheidbarkeit perfekt, statistisch oder berechenmässig ist, ist das Protokoll perfekt, statistisch oder berechenmässig zero-knowledge.

Das Protokoll heisst *honest-verifier zero-knowledge (HVZK)*, wenn der Simulator für den (korrekten) Verifizierer V existiert.[20]

Diese Definition enthält eine Forderung nach Effizienz (des Simulators) und nach Ununterscheidbarkeit und muss deshalb aus formalen Gründen asymptotisch formuliert werden. Es ist aber trotzdem in der Regel klar, wie man die zero-knowledge Eigenschaft eines konkreten Protokolls für eine (nicht asymptotisch definierbare) Aussage vernünftig definieren kann.

Zero-knowledge impliziert, dass das Transkript eines Protokolls keine Beweiskraft gegenüber Dritten besitzt. V (resp. ein betrügerischer V') könnte das gesamte Transkript ja selbst erzeugt haben, ohne dass dies von einem weiteren Verifizierer festgestellt werden kann.[21] Zero-knowledge impliziert zudem, dass ein Verifizierer V' selbst durch Betrug keine Information erhalten kann. Würde die Durchführung eines

[19] Diese Definition könnte verstärkt werden, indem sie für alle V', nicht nur polynomial beschränkte, gelten muss.

[20] und möglicherweise für andere Verifizierer nicht existiert.

[21] Ist das Protokoll nur berechenmässig zero-knowledge, so besitzt es in der Regel Überzeugungskraft für einen weiteren Verifizierer, sofern dieser berechenmässig unbeschränkt ist.

Protokolls mit P einem betrügerischen V' in irgend einer Weise helfen, so könnte sich V' ohne P die gleiche Hilfe ja selbst konstruieren (d.h. simulieren).

Um zu beweisen, dass ein Protokoll zero-knowledge ist, muss im Prinzip für jedes mögliche Programm V' ein Simulator angegeben werden[22] und bewiesen werden, dass dessen Output von einer Kommunikation von V' mit dem echten P nicht unterscheidbar ist. In allen bekannten Fällen kann aber *ein* Simulator S angegeben werden, der das Programm V' als Subroutine verwendet, ohne selbst von V' abzuhängen. Man nennt dies einen *Black-Box* Simulator.

1.6.3 Beweis der perfekten zero-knowledge Eigenschaft

In diesem Abschnitt beweisen wir formal, dass ein interaktiver Beweis mit bestimmten generischen Eigenschaften perfekt zero-knowledge ist. Wir betrachten nur Protokolle, in denen die Challenges durch einen ehrlichen Verifizierer jeweils unabhängig vom Rest gewählt werden, und zwar mit Gleichverteilung über den Challengebereich $\mathcal{C}$.

Die honest-verifier zero-knowledge (HVZK) Eigenschaft kann nicht nur für ein gesamtes Protokoll, sondern ganz natürlich für eine einzelne Runde (mit z.B. 3 Nachrichten) definiert werden. Eine solche Runde ist HVZK, wenn man Tripel (t, c, r) mit der gleichen Verteilung (T, C, R) wie im echten Protokoll mit dem ehrlichen Verifizierer simulieren kann.

Definition 1.7. Eine Protokollrunde, bestehend aus drei Teilrunden, wobei P einen Wert t an V sendet, dann V einen unabhängig und zufällig gewählten Challenge c an P sendet und anschliessend P einen Wert r an V sendet, heisst *C-simulierbar*[23], wenn man effizient für jeden möglichen Wert c ein Tripel (t, c, r) generieren kann, und zwar zufällig gemäss der in der Durchführung des Protokolls auftretenden bedingten Verteilung des Transkripts (T, C, R), gegeben dass $C = c$.

Mit anderen Worten muss ein effizienter Algorithmus existieren, der für jedes c Werte t und r erzeugt, was eine bedingte WSK-Verteilung $Q_{TR|C=c}$ impliziert, sodass $Q_{TR|C}(t, r, c) = P_{TR|C=c}(t, r, c)$ für alle t, r, c.

C-Simulierbarkeit impliziert, dass sich eine Beweiserin P auf jeden möglichen Challenge vorbereiten kann. Alle bisher diskutierten Protokolle bestehen aus C-simulierbaren Runden, was jeweils einfach zu verifizieren ist. Deshalb haben wir den Begriff der C-Simulierbarkeit überhaupt explizit eingeführt. Eine C-simulierbare Runde ist immer HVZK, woran wir eigentlich interessiert sind. (Der Simulator für einen ehrlichen Verifizierer generiert ein $c \in \mathcal{C}$ zufällig und erzeugt t und r entsprechend der C-Simulierbarkeit.)

Theorem 1.2. *Ein aus unabhängigen perfekt HVZK (z.B. C-simulierbaren) Runden $(T_1, C_1, R_1), \ldots, (T_k, C_k, R_k)$ bestehender interaktiver Beweis ist perfekt HVZK. Ist zusätzlich der Challengebereich $\mathcal{C}$ jeder Runde nur polynomial gross mit uniformer Wahl des Challenges (durch einen ehrlichen Verifier), so ist das Protokoll perfekt zero-knowledge.*

Beispiel 1.8. Das GI-Protokoll und das Fiat-Shamir Protokoll sind auf Grund dieses Theorems perfekt zero-knowledge. Das Guillou-Quisquater und das Schnorr-Protokoll sind "nur" perfekt honest-verifier zero-knowledge. Würde die Grösse des Challengebereichs in diesen Protokollen polynomial beschränkt, so wären sie auch perfekt zero-knowledge.

Beweis. Für den Fall eines korrekten Verifizierers (HVZK) besteht der Simulator einfach aus der unabhängigen Simulation der einzelnen Runden, unter Verwendung des HVZK-Simulators für die jeweilige Runde.

Wenn der Verifizierer V' aber nicht korrekt ist, so ist in der Regel die Verteilung der durch den beschriebenen Simulator S erzeugten Tripelfolge nicht gleich wie im echten Protokoll. Mit anderen Worten ist S kein brauchbarer Simulator. Wählt z.B. V' immer den Challenge $c = 0$, so ist dies offensichtlich, weil der oben beschriebene Simulator Tripel mit $c \neq 0$ erzeugt, die im echten Protokoll (mit V') gar nicht auftreten können.

Die allgemeinste Strategie eines betrügerischen V' ist, in jeder Runde (z.B. der i-ten) den Challenge c_i abhängig (probabilistisch oder mittels einer deterministischen Funktion[24]) von der

[22] Man beachte, dass die Konstruktion des Simulators lediglich eine Beweistechnik ist und ein Gedankenexperiment darstellt. In Wirklichkeit wird ein Simulator nie programmiert.

[23] In der Literatur wird dies auch *special honest-verifier zero-knowledge* genannt.

[24] z.B. eine Hashfunktion, wobei nur ein Bit des Hashwertes verwendet wird.

bisherigen Kommunikation im Protokoll zu wählen. Die bisherige Kommunikation besteht aus $(t_1, c_1, r_1), (t_2, c_2, r_2), \ldots, (t_{i-1}, c_{i-1}, r_{i-1})$ und t_i. Sei

$$u_i = [t_1, \ldots, t_i, c_1, \ldots, c_i, r_1, \ldots, r_i]$$

die gesamte Kommunikation der ersten i Runden.

Der Simulator S erzeugt die k Tripel des simulierten Transkripts Runde um Runde und verwendet dabei V' als Subroutine (Black-Box Simulation). In jeder Runde (z.B. der i-ten) führt S eine Berechnung durch und speichert am Ende das Tripel (t_i, c_i, r_i) ab, z.B. in einem File. Die Berechnung für die i-te Runde besteht aus einem oder mehreren Versuchen, das Tripel (t_i, c_i, r_i) zu erzeugen. Jeder Versuch kann erfolgreich sein oder misslingen, was von S erkannt wird. S macht soviele (unabhängige) Versuche, bis einer erfolgreich ist, d.h. bei einem Misserfolg verhält sich der Simulator so, als ob dieser Versuch nicht stattgefunden hätte, und startet einen neuen Versuch für die i-te Runde. Bereits gespeicherte Tripel (t_i, c_i, r_i) müssen nie wieder gelöscht und neu generiert werden.

Nehmen wir an, die ersten $i-1$ Runden seien bereits simuliert, d.h. $(t_1, c_1, r_1), (t_2, c_2, r_2), \ldots, (t_{i-1}, c_{i-1}, r_{i-1})$ seien durch den Simulator erzeugt und abgespeichert. Die Simulation der i-ten Runde erfolgt wie folgt:

1. Ein Tripel (t'_i, c'_i, r'_i) gemäss HVZK-Simulation (für die i-te Runde) generieren.
2. Gemäss Strategie von V' den Challenge c''_i der i-ten Runde bestimmen, abhängig von u_{i-1} und t'_i.
3. Falls $c'_i = c''_i$ das Tripel (t'_i, c'_i, r'_i) als Simulation (t_i, c_i, r_i) der i-ten Runde abspeichern, sonst neu mit der Simulation der i-ten Runde beginnen (Schritt 1).

Es bleibt zu begründen, warum die WSK-Verteilung dieser Simulation korrekt ist und warum sie effizient (d.h. polynomial) ist.

Die *Effizienz* ist einfach zu sehen. Die Wahrscheinlichkeit, dass ein Versuch für die i-te Runde erfolgreich ist, d.h. dass das erzeugte Tripel verwendet werden kann, ist $1/|\mathcal{C}|$ (siehe unten). Im Mittel werden also in der Simulation pro Runde $|\mathcal{C}|$ Versuche benötigt, was polynomialen Aufwand erfordert, da $|\mathcal{C}|$ polynomial beschränkt ist.[25]

Der Beweis, dass die *Verteilung identisch* ist wie im echten Protokoll, ist etwas aufwendiger. Das folgende Argument gilt für einen beliebigen Verifizierer V'. Im Folgenden bezeichnen wir Wahrscheinlichkeiten im tatsächlichen Protokoll zwischen Peggy und V' mit P und bei der Simulation mit $\hat{P}$. Es gilt demnach zu zeigen, dass $P_{U_i} = \hat{P}_{U_i}$. Der Beweis erfolgt mittels Induktion. Nehmen wir an, es gelte $P_{U_{i-1}} = \hat{P}_{U_{i-1}}$. (Die Verankerung für $i = 1$ ist trivial.) Nun ist P_{T_i} die durch Peggy bestimmte WSK-Verteilung von t_i (T_i ist unabhängig von U_{i-1}), $P_{C_i|U_{i-1}T_i}$ die durch V' bestimmte WSK-Verteilung des i-ten Challenges und $P_{R_i|T_iC_i}$ die WSK-Verteilung der i-ten Antwort von Peggy (R_i hängt nur von T_i und C_i ab, nicht aber von U_{i-1}). Wir haben also bei gegebenem Wert u_{i-1} für U_{i-1}:

$$P_{T_iC_iR_i|U_{i-1}}(t, c, r, u_{i-1}) = P_{T_i}(t) \cdot P_{C_i|U_{i-1}T_i}(c, u_{i-1}, t) \cdot P_{R_i|T_iC_i}(r, t, c).$$

Im simulierten Protokoll wird ein Kandidatentripel (t'_i, c'_i, r'_i) erzeugt, das aber nur für die Simulation verwendet wird, wenn der gemäss Strategie von V' (auf Grund von u_{i-1} und t'_i) erzeugte Challenge gleich c'_i ist. Dieses Ereignis wird mit $\mathcal{E}_i$ bezeichnet, und in diesem Fall wird das Tripel (t'_i, c'_i, r'_i) als Tripel (t_i, c_i, r_i) abgespeichert. Die Verteilung jedes versuchten Tripels (t'_i, c'_i, r'_i) ist unabhängig von u_{i-1}, nicht aber das schlussendlich abgespeicherte Tripel (t_i, c_i, r_i), da die Entscheidung über eine Abspeicherung ja von u_{i-1} abhängt. Da das Tripel (t'_i, c'_i, r'_i) mittels HVZK-Simulation erzeugt wird, gilt

$$\hat{P}_{C'_i|U_{i-1}}(c, u_{i-1}) = \frac{1}{|\mathcal{C}|}$$

für alle c und u_{i-1},

$$\hat{P}_{T'_i|C'_iU_{i-1}}(t, c, u_{i-1}) = \hat{P}_{T'_i}(t) = P_{T_i}(t)$$

[25] Die Effizienz der Simulation ist der Grund, warum die Grösse des Challengebereichs C in Theorem 1.2 beschränkt werden muss. Für sehr grosse $|\mathcal{C}|$ funktioniert der Simulator zwar korrekt, aber nicht effizient.

für alle t, c und u_{i-1}, sowie

$$\hat{P}_{R_i'|T_i'C_i'U_{i-1}}(r,t,c,u_{i-1}) = P_{R_i|T_iC_i}(r,t,c)$$

für alle r, t, c und u_{i-1}. Somit gilt

$$\hat{P}_{T_i'C_i'R_i'|U_{i-1}}(t,c,r,u_{i-1}) = \frac{1}{|\mathcal{C}|} \cdot P_{T_i}(t) \cdot P_{R_i|T_iC_i}(r,t,c),$$

und

$$\begin{aligned}
&\hat{P}_{T_i'C_i'R_i'\mathcal{E}_i|U_{i-1}}(t,c,r,u_{i-1}) \\
&\quad = \hat{P}_{T_i'C_i'R_i'|U_{i-1}}(t,c,r,u_{i-1}) \cdot P_{C_i|U_{i-1}T_i}(c,u_{i-1},t) \\
&\quad = \frac{1}{|\mathcal{C}|} \cdot P_{T_i}(t) \cdot P_{R_i|T_iC_i}(r,t,c) \cdot P_{C_i|U_{i-1}T_i}(c,u_{i-1},t).
\end{aligned}$$

Die WSK eines Erfolgs bei einem Simulationsversuch für die i-te Runde ist immer $1/|\mathcal{C}|$. Dies folgt aus der Tatsache, dass die Verteilung von t_i' nicht von c_i' abhängt und dass deshalb die Wahrscheinlichkeit, dass der gemäss Strategie von V' gewählte Challenge c_i gleich der Wahl c_i' des Simulators ist, gleich $1/|\mathcal{C}|$ ist. Die korrekte Herleitung folgt der Vollständigkeit halber:

$$\hat{P}_{\mathcal{E}_i|U_{i-1}}(u_{i-1}) = \sum_{t,c,r} \hat{P}_{T_i'C_i'R_i'\mathcal{E}_i|U_{i-1}}(t,c,r,u_{i-1}) =$$

$$\underbrace{\frac{1}{|\mathcal{C}|} \cdot \sum_t \Big(P_{T_i}(t) \underbrace{\sum_c \Big(P_{C_i|U_{i-1}T_i}(c,u_{i-1},t) \underbrace{\sum_r P_{R_i|T_iC_i}(r,t,c)}_{=1 \text{ für alle } t \text{ und } c} \Big)}_{=1 \text{ für alle } t \text{ und } u_{i-1}} \Big)}_{=1}.$$

Die Summen sind jeweils 1, weil über eine WSK-Verteilung summiert wird. Folglich gilt (durch Einsetzen):

$$\begin{aligned}
\hat{P}_{T_iC_iR_i|U_{i-1}}(t,c,r,u_{i-1}) &= \frac{\hat{P}_{T_i'C_i'R_i'\mathcal{E}_i|U_{i-1}}(t,c,r,u_{i-1})}{\hat{P}_{\mathcal{E}_i|U_{i-1}}(u_{i-1})} \\
&= P_{T_i}(t) P_{C_i|U_{i-1}T_i}(c,u_{i-1},t) P_{R_i|T_iC_i}(r,t,c) \\
&= P_{T_iC_iR_i|U_{i-1}}(t,c,r,u_{i-1}),
\end{aligned}$$

und somit wegen $P_{U_{i-1}} = \hat{P}_{U_{i-1}}$ auch $P_{U_i} = \hat{P}_{U_i}$. □

1.6.4 Protokolle basierend auf Bit-Commitments

Interaktive Beweise, die auf Bit-Commitments beruhen, haben je nach verwendetem Typ des Bit-Commitment Verfahrens grundlegend unterschiedliche Eigenschaften. Wir nehmen im Folgenden an, das Protokollgerüst, d.h. wenn alle Blobs aus dem Protokoll "entfernt" werden, sei perfekt zero-knowledge.

- **Typ H Commitments.** Das Protokoll ist perfekt zero-knowledge, weil sich Commitments zu 0 und zu 1 nicht unterscheiden und deshalb perfekt simulierbar sind. Da das Bit-Commitment Verfahren aber nur berechenmässig sicher für Vic ist, könnte Peggy mit genügend Computerressourcen betrügen, weil sie Blobs beliebig öffnen könnte. Ein solches Protokoll wird deshalb nicht interaktiver Beweis, sondern *interaktives Argument* genannt. Man spricht auch von "computational soundness".
- **Typ B Commitments.** Das Protokoll ist höchstens berechenmässig zero-knowledge, da man mit sehr grossen Computerressourcen das echte vom simulierten Transkript unterscheiden könnte. Dafür kann Peggy selbst mit unendlichen Computerressourcen nicht betrügen. Ein solches Protokoll nennen wir, wie schon früher definiert, einen *interaktiven Beweis.*

In den folgenden beiden Abschnitten werden die Beweise der zero-knowledge Eigenschaft für diese beiden Fälle diskutiert.

1.6.5 Commitments vom Typ H

Der in Abschnitt 1.6.3 beschriebene Beweis der zero-knowledge Eigenschaft eines Dreirundenprotokolls gilt auch für Protokolle, die auf Bit-Commitments vom Typ H basieren. Wir betrachten Protokolle, bei denen in jeder Runde (z.B. der i-ten) Blobs als Teil von t_i gesendet und, als Teil von r_i, einige davon geöffnet werden. Das HK-Protokoll[26] hat diese Form.

Der Simulator kann in der Simulation der i-ten Runde nach der Wahl des Challengewertes c'_i eine zufällige Antwort r_i erzeugen, inklusive den für diesen Wert von c'_i notwendigen Öffnungen von zufälligen Blobs zu den entsprechenden Bits. Diese Blobs werden als Teil von t'_i verwendet. Werden in der ersten Nachricht (d.h. t_i) zusätzliche Blobs benötigt, so können diese beliebig gewählt werden, weil Blobs für 0 und 1 die identische Verteilung haben und demnach Blobs, die der Simulator nicht öffnen muss, beliebige Werte enthalten können.[27]

Beispiel 1.9. Beim Protokoll für Hamiltonsche Kreise in einem Graphen mit n generiert der Simulator für die Simulation der i-ten Runde einen Challengewert c'_i. Im Fall $c'_i = 0$ permutiert er den Graphen zufällig und generiert als t'_i die n^2 Blobs zu den Bits des permutierten Graphen und als r'_i die Permutation und die Öffnungen aller n^2 Blobs. Im Fall $c'_i = 1$ erzeugt der Simulator als t'_i n^2 zufällige Blobs mit Wert 1 und als r'_i einen zufälligen HK und die Öffnungen der entsprechenden n Blobs. Dann wird, wie schon früher beschrieben, c_i gemäss Strategie von V' bestimmt, und falls $c_i \neq c'_i$ wird die i-te Runde neu simuliert. Dies ist eine perfekte zero-knowledge Simulation für Commitments vom Typ H.

1.6.6 Commitments vom Typ B

Wenn das Bit-Commitment Verfahren vom Typ B ist, dann gilt es zu beweisen, dass ein Distinguisher für das echte und das simulierte Transkript verwendet werden kann, um Commitments zum Wert 0 von Commitments zum Wert 1 zu unterscheiden. Dies impliziert, dass das Protokoll berechenmässig zero-knowledge ist.

Dieser Beweis ist deutlich anspruchsvoller und wird hier nicht gegeben. Das schwierige Problem ist, ein gegebenes Commitment (für das entschieden werden soll, ob es 0 oder 1 enthält) in ein Transkript einzubetten, welches dann dem Distinguisher für Transkripte vorgelegt werden kann.

1.6.7 Randomisierbare Blobs

Durch Wiederholung des Unterscheidungsexperiments kann man die Unterscheidungswahrscheinlichkeit von 0-Blobs und 1-Blobs verstärken. Damit dieser statistische Verstärkungseffekt nicht nur heuristisch funktioniert, sondern bewiesen werden kann, muss man Blobs randomisieren können (siehe unten). Für randomisierbare Bit-Commitment Verfahren kann ein beliebiger Blob (nicht notwendigerweise zufällig) mit überwältigender Wahrscheinlichkeit geöffnet werden, sofern ein effizienter Unterscheidungsalgorithmus A gegeben ist.

Definition 1.8. Ein Commitment-Verfahren heisst *randomisierbar*, wenn man für jeden Blob einen zufälligen Blob erzeugen kann, der den gleichen Wert enthält, ohne diesen zu kennen.

Beispiel 1.10. Die Blobs der Abschnitte 1.4.3 und 1.4.4 sind randomisierbar, wie man einfach sieht.

1.6.8 Parallelisierung von interaktiven Argumenten

Bei der praktischen Verwendung eines interaktiven Beweises (z.B. über das Internet) ist nicht nur die gesamte Anzahl kommunizierter Bits, sondern auch (oder sogar primär) die Anzahl Runden relevant.

[26] Hamiltonsche Kreise, vgl. 1.2.6.

[27] "Enthalten" ist eigentlich die falsche Bezeichnung, da ein Blob ja auf beide Arten geöffnet werden könnte. Es geht hier darum, für welches Bit der Simulator den Blob öffnen *könnte*. Da die Öffnung aber nicht durchgeführt wird, spielt es keine Rolle, wenn der Simulator die Blobs gar nicht korrekt öffnen könnte.

Jedes Protokoll, das aus der s-maligen Wiederholung eines Subprotokolls besteht, kann durch eine Parallelisierung auch in einer einzigen (drei-phasigen) Runde durchgeführt werden. Zuerst generiert Peggy $t_1, \ldots, t_s$ und sendet sie an Vic, dann wählt Vic die s Challenges $c_1, \ldots, c_s$, und anschliessend sendet Peggy die s Antworten $r_1, \ldots, r_s$ an Vic. Obwohl dieses Protokoll intuitiv gleich gut ist wie das serielle Protokoll mit s Runden, besteht doch ein wesentlicher Unterschied: das parallele Protokoll ist im Allgemeinen *nicht* zero-knowledge, weil der Challengebereich exponentiell in s ist und die Laufzeit des Simulators proportional zur Grösse des Challengebereichs ist. Formal gesehen ist ein paralleles Protokoll nur zero-knowledge, falls die Anzahl Runden $s = O(\log n)$, d.h. falls 2^s polynomial in n ist (wodurch aber die Betrugswahrscheinlichkeit nicht vernachlässigbar wird).

Dank einem speziellen Typ von Bit-Commitment Verfahren, den sogenannten *Trapdoor*-Bit-Commitments, kann die zero-knowledge Eigenschaft auch für die parallele Version eines interaktiven Arguments (mit Commitments vom Typ H) erreicht werden, z.B. für das Protokoll für Hamiltonsche Kreise.

Definition 1.9. Ein Commitment Verfahren vom Typ H heisst *Trapdoor-Commitment Verfahren*, wenn es einen geheimen Parameter (die Trapdoor) gibt, der erlaubt, committete Werte beliebig zu öffnen.

Man beachte, dass dies nicht im Widerspruch zur Sicherheit des Commitment-Verfahrens ist. Die zero-knowledge Eigenschaft gilt bei der Verwendung von Trapdoor-Commitments selbst für die parallele Version, weil der Simulator, der das Programm V' verwenden kann, bei der Simulation nie ein Problem hat. Er kann unter Verwendung der Trapdoor alle Blobs korrekt öffnen.

Beispiel 1.11. Das Bit-Commitment Verfahren aus Abschnitt 1.4.3 hat die Trapdoor Eigenschaft, wenn Vic (aber natürlich nicht Peggy) den diskreten Logarithmus von h zur Basis g kennt. Vic könnte z.B. h als $h = g^t$ für ein zufälliges t erzeugen und (in einem separaten Protokoll) zuerst Peggy beweisen, dass er t kennt.

1.7 Interaktive Beweise von Wissen (Proofs of Knowledge)

Wenn jemand einen bestimmten String kennt, impliziert dies insbesondere, dass ein solcher String existiert. Umgekehrt gilt die Aussage natürlich nicht. Selbst wenn ein bestimmter String (z.B. der zu einem Public Key gehörende geheime Schlüssel) existiert, ist offen, ob eine bestimmte Entität diesen String kennt. Dies zu beweisen macht in vielen Anwendungen Sinn, unter anderem für Beweise von Aussagen, indem man die Kenntnis eines Beweises der Aussage zeigt. Beweise von Wissen müssen aber anders formalisiert werden als Beweise von Aussagen.

1.7.1 Definition

Wissen w wird formalisiert in Bezug auf einen gegebenen String z (den Input für Peggy und Vic) und ein effizient berechenbares Verifikationsprädikat

$$Q : \{0,1\}^* \times \{0,1\}^* \to \{\texttt{false}, \texttt{true}\}.$$

Peggy möchte beweisen, dass sie einen String w kennt mit $Q(z, w) = \texttt{true}$. Im Allgemeinen können mehrere solche w existieren und es ist egal, welches w Peggy kennt, solange sie ein w kennt. Beispiele sind:

- z ist die binäre Codierung einer ganzen Zahl, w ist ein Paar von Zahlen, Q zeigt an, ob das Produkt dieser Zahlen gleich z ist und ob die Faktoren > 1 sind.
- z codiert einen Graphen mit n Knoten, w ist eine Liste von n Kanten und $Q(z, w)$ ist `true` genau dann, wenn w ein Hamiltonscher Kreis im Graphen ist.
- z ist der Public Key, w ein zugehöriger Secret Key und Q die Verifikationsfunktion dieser Tatsache.
- z ist eine Aussage, w ist ein formaler Beweis dieser Aussage, und Q ist die Verifikationsfunktion des Beweiskalküls.

Bei einem Beweis von Wissen ist oft vom Kontext klar, dass der String w existiert. Dann ist die Frage nur, ob Peggy w kennt, nicht, ob eine solches w existiert. Ein solcher Beweis macht nur Sinn, wenn man davon ausgeht, dass Peggy's Computerressourcen (z.B. polynomial) beschränkt sind.

Ein interaktives Protokoll ist ein Beweis von Wissen für das Prädikat Q in Bezug auf einen String z, wenn jemand ohne Kenntnis eines passenden Strings w mit $Q(z, w) = \texttt{true}$ den Verifizierer Vic nicht (oder nur mit verschwindender Wahrscheinlichkeit) zum Akzeptieren bringen kann. Mit anderen Worten, erfolgreiches Überzeugen von Vic soll gleichbedeutend sein mit dem Wissen eines (von möglicherweise mehreren) passenden w.

In Abschnitt 1.6 wurde in der Definition von zero-knowledge formalisiert, was es heisst, dass ein Programm etwas *nicht* weiss. Hier müssen wir definieren, wann ein Programm einen String w "weiss". Ist dies der Fall, wenn das Programm den String w als Teil des Outputs hinschreibt, oder wenn w irgendwann als Wert einer bestimmten Variablen angenommen wird, oder wenn sich w aus den Werten der internen Variablen berechnen lässt? Keine dieser Formulierungen ist präzise und allgemein genug.

Wir müssen als Gedankenexperiment wiederum ein spezielles Programm konstruieren, den sogenannten *Wissensextraktor* W. Ein Wissensextraktor kann jedes beliebige Programm $\hat{P}$ verwenden, das Vic mit nicht vernachlässigbarer Wahrscheinlichkeit zum Akzeptieren bringt, und kann daraus effizient und mit überwältigender Wahrscheinlichkeit einen passenden String w erzeugen. Dabei wird lediglich die Eigenschaft von $\hat{P}$ verwendet, im Beweis mit nicht vernachlässigbarer Wahrscheinlichkeit erfolgreich zu sein.

Definition 1.10. Ein interaktives Protokoll zwischen[28] P und V ist ein *Beweis von Wissen für das Prädikat* Q, falls folgendes gilt:

- *(Vollständigkeit)* V akzeptiert immer, wenn P ein passendes w kennt und das Protokoll korrekt durchführt
- *(Korrektheit)* Es gibt ein effizientes (polynomielles) Programm W (den Wissensextraktor), welches mit einem Beweiser P interagieren kann und für alle z folgende Eigenschaft hat. Jedes probabilistische polynomiale interaktive Programm $\hat{P}$, das im interaktiven Protokoll V mit nicht vernachlässigbarer Wahrscheinlichkeit zum Akzeptieren bringt (z wird $\hat{P}$ und V als Input gegeben), kann von W als Blackbox verwendet werden[29], so dass W mit überwältigender Wahrscheinlichkeit ein w mit $Q(z, w) = \texttt{true}$ erzeugt.

1.7.2 Konstruktion des Wissensextraktors

Definition 1.11. Eine dreiphasige Protokollrunde vom beschriebenen Typ mit Challengebereich $\mathcal{C}$ für einen Beweis von Wissen für das Prädikat $Q(z, w)$ heisst *2-extrahierbar*[30], wenn aus beliebigen zwei Tripeln (t, c', r') und (t, c'', r'') mit $c' \neq c''$ in polynomialer Zeit ein w mit $Q(z, w) = \texttt{true}$ berechnet werden kann.

Die bisher besprochenen Protokolle haben diese Eigenschaft.

Theorem 1.3. *Jeder interaktive Beweis für das Prädikat $Q(z, w)$, der aus s 2-extrahierbaren Runden mit Challengeraum $\mathcal{C}$ besteht, ist ein Beweis von Wissen, sofern $1/|\mathcal{C}|^s$ verschwindend (in der Inputgrösse $|z|$) ist.*

Der letzte Punkt impliziert die natürliche Bedingung, dass jede fixe Challengefolge $c_1, \ldots, c_s$ nur verschwindende WSK hat und somit nicht erraten werden kann. Die Rundenzahl s muss entsprechend gewählt werden.[31]

Beweis. Für einen konkreten String z, sei p die (nicht vernachlässigbare) Wahrscheinlichkeit, dass $\hat{P}$ (wie in Definition 1.10) V zum Akzeptieren bringt. Wir konstruieren einen Wissensextraktor W, der mit überwältigender Wahrscheinlichkeit einen passenden String w findet. (Das Theorem folgt dann direkt aus Definition 1.10).

[28] P ist berechenmässig polynomiell zeitbeschränkt
[29] Falls $\hat{P}$ probabilistisch ist, muss W die Wahl der internen Zufallsbits von $\hat{P}$ bestimmen können.
[30] Diese Eigenschaft wird bei genügend grossem Challengebereich $\mathcal{C}$ auch als *special soundness* bezeichnet.
[31] Z.B. ist für $|\mathcal{C}| = 2$ die WSK $1/|\mathcal{C}|^s$ verschwindend falls $s = (\log |z|)^2$, jedoch nicht verschwindend falls $s = \log |z|$.

Wenn $\hat{P}$ probabilistisch ist, kann die verwendete Zufälligkeit innerhalb von $\hat{P}$ ohne Verlust an Allgemeinheit als Bitstring ℓ modelliert werden. Für jeden festen Wert von ℓ ist $\hat{P}$ deterministisch. Der Mittelwert der Erfolgswahrscheinlichkeit aller Werte von ℓ ist p.

Der Wissensextraktor W kann wie folgt definiert werden:

1. ℓ zufällig wählen.
2. Zwei unabhängige Protokolldurchläufe (mit den fixen Zufallsbits ℓ) zwischen $\hat{P}$ und V erzeugen.
3. Falls in Schritt 2. V für beide Durchläufe akzeptiert und verschiedene Challengefolgen gewählt hat, dann wird die erste Runde der s Runden betrachtet, in der sich die beiden Challengefolgen unterscheiden. Aus den entsprechenden zwei Tripeln wird w extrahiert.

   Sonst zurück zu Schritt 1.

Da im Schritt 2. beide Protokolldurchläufe mit dem gleichen ℓ durchgeführt werden, sind die t_i's in beiden Durchläufen bis und mit der ersten Runde in der sich die c_i's unterscheiden identisch. Existiert eine solche Runde mit unterschiedlichen Challenges bei beiden Durchläufen, so kann, da jede Runde des Protokolls 2-extrahierbar ist, in Schritt 3. ein korrektes w generiert werden.

Es bleibt nur zu zeigen, dass die Laufzeit, d.h. die Anzahl Versuche bis zu einem Erfolg, polynomial ist. Sei $f(\ell)$ die Erfolgswahrscheinlichkeit von $\hat{P}$ für die Wahl ℓ des Zufallsstrings. Wenn L die Zufallsvariable für die Wahl von ℓ bezeichnet, so gilt gemäss Annahme $E[f(L)] = p$. Die Wahrscheinlichkeit, dass der beschriebene Wissensextraktor für die Wahl ℓ zwei akzeptierende Durchläufe generiert ist $f(\ell)^2$, wobei wir vernachlässigen, dass beide Durchläufe mit sehr kleiner WSK identisch sein können. Demzufolge ist die Wahrscheinlichkeit, bei zufälliger Wahl von L (Schritt 1.) in Schritt 2. zwei solche Durchläufe zu generieren $E[f(L)^2]$. Die erwartete Anzahl der notwendigen Versuche ist demnach $O(1/E[f(L)^2])$. Da die Funktion $x \mapsto x^2$ konvex ist, können wir Jensen's Ungleichung anwenden, die besagt, dass für eine Zufallsvariable X gilt: $E[X^2] \geq E[X]^2$. Dies impliziert

$$E[f(L)^2] \geq E[f(L)]^2 = p^2.$$

Die Laufzeit des Wissensextraktors ist somit $O(1/p^2)$, was polynomial ist, weil p nicht vernachlässigbar ist. □

Beispiel 1.12. Im Fiat-Shamir Protokoll (oder jedem analogen Protokoll) genügt es nicht für die Konstruktion eines effizienten Wissensextraktors, wenn die Erfolgswahrscheinlichkeit von $\hat{P}$ bei s Runden grösser als 2^{-s} ist. Dann gäbe es (vermutlich) keinen effizienten Wissensextraktor (mit polynomialer Laufzeit). Es ist notwendig, dass man in Definition 1.10 fordert, dass $\hat{P}$ mit *nicht vernachlässigbarer* WSK Erfolg hat.

1.7.3 Witness Hiding und Witness Independence

Interaktive Beweise von Wissen haben Anwendungen als Identifikationsprotokolle. Aus Sicht des Beweisers wäre es am besten, die zero-knowledge Eigenschaft des Protokolls zu beweisen, aber in gewissen Fällen, z.B. wenn der Challengebereich pro Runde zu gross ist, ist dies nicht möglich. Eine fast so gute Aussage wäre, dass ein unehrlicher Verifizierer V' keine Information erhalten kann, die ihm erlaubt, sich selbst als Beweiser auszugeben, d.h. das Identifikationsprotokoll mit einem ahnungslosen Verifier V erfolgreich zu bestehen.[32] Dies wollen wir formalisieren und für bestimmte Protokolle beweisen.

Definition 1.12. Ein interaktiver Beweis von Wissen zwischen P und V heisst *witness hiding*, wenn es keinen polynomialen Verifizierer V' gibt, der nach beliebiger Interaktion mit dem Beweiser P selbst die Rolle eines Beweisers übernehmen kann und mit nicht vernachlässigbarer Wahrscheinlichkeit V zum Akzeptieren bringen kann.

[32] Es ist also zwar möglich, dass V' gewisse Information erhält, die er nicht selbst effizient generieren kann, aber sie kann im Kontext der Identifikation nicht nützlich sein.

Im Folgenden sei Q ein Prädikat, und für ein z bezeichne $\mathcal{W}_z$ die Menge aller Zeugen (Witnesses) für z, d.h. die Menge aller w, so dass $Q(z, w) = \texttt{true}$. Um die witness-hiding Eigenschaft zu beweisen, ist die folgende Eigenschaft nützlich. In vielen Fällen gibt es für einen gegebenen String z mehrere Zeugen, d.h. $|\mathcal{W}_z| > 1$.

Definition 1.13. Ein interaktiver Beweis von Wissen für Q zwischen P (welche ein $w \in \mathcal{W}_z$ kennt) und V heisst *witness independent*[33], wenn für alle z und alle möglichen V' die Verteilung des Transkripts identisch ist für jedes $w \in \mathcal{W}_z$.

Mit anderen Worten, das Transkript des Protokolls gibt V' keine Information darüber, welchen Witness der Beweiser kennt.

Theorem 1.4. *Falls es berechenmässig schwierig ist, ein Tripel (z, w, w') mit $w \in \mathcal{W}_z$, $w' \in \mathcal{W}_z$ und $w \neq w'$ zu finden, und man effizient ein Paar (z, w) erzeugen kann, wobei $|\mathcal{W}_z| > 1$ und w zufällig[34] aus $\mathcal{W}_z$ ist , dann ist jeder Beweis von Wissen für das Prädikat Q, welcher witness independent ist, auch witness hiding.*

Beweis. Man kann (wie oben beschrieben) ein zufälliges Paar (z, w) erzeugen und dafür den Beweis von Wissen zwischen P und V betrachten. Nehmen wir an, es gäbe einen Verifizierer V' gemäss Definition 1.12, der zuerst mit dem Beweiser P interagiert und anschliessend den Verifier V zum Akzeptieren bringen kann. Folglich kann man mit Hilfe des Wissensextraktors einen Witness w' extrahieren. Weil w zufällig aus $\mathcal{W}_z$ ist und wegen der witness independence Eigenschaft, haben wir mit nicht vernachlässigbarer WSK, dass $w' \neq w$. Dies widerspricht der Annahme im Theorem, dass kein solches (z, w, w') gefunden werden kann. □

Beispiel 1.13. Wir betrachten wie schon in Abschnitt 1.4.3 eine Gruppe G mit Ordnung q (q prim) sowie zwei Generatoren g und h von G, so dass $\log_g h$ unbekannt ist. Eine *Repräsentation* eines Elementes $a \in G$ ist ein Paar (x, y) mit $a = g^x h^y$. Kenntnis von zwei verschiedenen Repräsentationen von a impliziert Kenntnis von $\log_g h$, was (vermutlich) schwierig zu berechnen ist.

Der Leser kann als Übungsaufgabe das Protokoll von Schnorr abändern, um daraus einen Beweis von Wissen einer Repräsentation eines Elementes $a \in G$ zu erhalten, welcher witness independent ist. Dieses Protokoll, das ursprünglich von Okamoto vorgeschlagen wurde, ist demzufolge gemäss Theorem 1.4 witness hiding.

1.8 Das Brassard-Chaum-Crépeau Protokoll für Circuit Satisfiability

Der Artikel von Brassard, Chaum und Crépeau [BCC88] wird als Beilage verteilt. Wir beschränken uns deshalb hier auf einige Bemerkungen und Ergänzungen. Mit dem BCC-Protokol kann man in zero-knowledge beweisen, dass man eine Inputbelegung zu einer gegebenen Schaltung kennt, so dass der Output der Schaltung 1 ist. Dies impliziert ein zero-knowledge Protokoll für die Sprache Circuit-SAT (satisfiability), und damit für alle Sprachen in **NP**. Dass jede Sprache in **NP** einen (berechenmässigen) zero-knowledge interaktiven Beweis besitzt, folgte schon aus dem Protokoll für Hamiltonsche Kreise. Das BCC-Protokoll ist für uns trotzdem interessant, weil in der Regel eine zu beweisende Aussage mit diesem Protokoll viel direkter bewiesen werden kann.

Konkreter: Gegeben ist eine digitale Schaltung mit n Inputs und einem Output, für die Peggy behauptet, eine Inputbelegung zu kennen, welche am Ausgang 1 ergibt. Statt diese Inputbelegung Vic einfach zu zeigen, möchte sie den Beweis mit einem zero-knowledge Protokoll durchführen.

Wie schon beim Protokoll für Hamiltonsche Kreise existieren zwei Typen von Realisierungen. Je nach verwendetem Bit-Commitment Verfahren ist das Protokoll ein berechenmässig zero-knowledge interaktiver Beweis (Typ B) oder ein perfekt zero-knowledge interaktives Argument (Typ H).

Im Folgenden werden einige zusammenfassende Erklärungen zum Artikel gegeben. Die logischen Gatter der Schaltung (mit zwei Inputs und einem Output) werden durch eine Wertetabelle (bestehend aus 12

[33] Oft wird der Begriff *witness indistinguishable* verwendet. Dieser Begriff schliesst mit ein, dass die Unterscheidbarkeit nur berechenmässig ist.

[34] Es reicht hier, wenn mindestens zwei Elemente aus $\mathcal{W}_z$ nicht vernachlässigbare WSK haben.

Bits) spezifiziert. (siehe Figur 1 in [BCC88]). Ähnlich wie im Protokoll für Hamiltonsche Kreise verschleiert Peggy zuerst die Schaltung (was verschleiern in diesem Kontext heisst, ist in Figur 2 dargestellt) und wird von Vic aufgefordert, einen von zwei Challenges zu beantworten. Entweder muss sie zeigen, dass die verschleierte Schaltung der ursprünglichen entspricht, oder sie muss zeigen, dass sie in der verschleierten Schaltung einen Input kennt, der zum Output 1 führt. Beide möglichen Antworten zusammen ergeben die Inputbelegung für die ursprüngliche Schaltung, d.h. man kann beide Challenges nur beantworten, wenn man die Inputbelegung kennt.

Die Verschleierung einer Schaltung umfasst eine zufällige (unabhängige) Permutation der Zeilen jedes Gatters sowie für jeden Draht der Schaltung (ausser beim Ausgang) die Addition eines zufälligen Bits (dies entspricht einer Art One-Time Pad Verschlüsselung). Dass diese Art von Verschleierung genügt, um das Protokoll zero-knowledge zu machen, muss man grundsätzlich durch die Angabe eines Simulators beweisen. Eine einfache Betrachtung zeigt, dass sich der Simulator auf jeden einzelnen Challenge vorbereiten kann (aber natürlich nicht auf beide). Im einen Fall wird einfach die Schaltung zufällig verschleiert, im anderen Fall wird in jedem Gatter zufällig eine der 4 Zeilen ausgewählt und mit zufälligen Bits belegt, wobei die Konsistenzbedingungen der Drähte berücksichtigt werden. Die anderen Bits der Tabellen werden zufällig gewählt. Anschliessend werden die Commitments erzeugt. Diese Überlegungen sind analog zu denjenigen für das Protokoll für Hamiltonsche Kreise.

1.9 Einweg-Homomorphismen und Beweise von Wissen

In diesem Abschnitt behandeln wir ein Framework für spezielle Typen von Beweisen von Wissen. Ziel ist es, die grundlegenden Eigenschaften vieler Protokolle zu abstrahieren und ein einziges Protokoll zu beschreiben, von dem einige der bisher besprochenen Protokolle sowie eine Fülle weiterer Protokolle Spezialfälle sind. Der Vorteil eines solchen Vorgehens ist, dass mit einer einzigen Analyse sehr viele Protokolle analysiert werden können, und dass die Beweise auf der (vermutlich) richtigen Abstraktionsstufe erfolgen. Die Notation ist in diesem Abschnitt leicht geändert.

1.9.1 Einweg-Homomorphismen

Sei $f : G \to H$ eine Einwegfunktion von einer Menge G auf eine Menge H. Wir werden für $f(x)$ auch die Kurznotation $[x]$ verwenden, manchmal auch $[\![x]\!]$.

Mit Hilfe der Funktion f kann ein Commitment Verfahren für Elemente aus G realisiert werden, wobei f für die "Hiding" und "Binding" Eigenschaften eines Commitments noch zusätzliche Forderungen erfüllen muss. Wenn f z.B. nicht bijektiv ist, dann muss man verlangen, dass f kollisionsfrei ist, d.h. dass es berechenmässig schwierig ist, x und x' zu finden mit $f(x) = f(x')$.[35]

Sehr viele interaktive Beweise können formuliert werden als Beweise von Wissen eines Wertes x mit $z = f(x)$ (resp. $z = [x]$) für einen gegebenen Wert z, für eine bestimmte Einwegfunktion $f : G \to H$ (für bestimmte Mengen G und H). Das BCC-Protokoll ist eine allgemeine, aber relativ ineffiziente Lösung für diesen Beweis, basierend auf einer Schaltung für die Berechnung von f. Wenn G, H und f aber bestimmte algebraische Struktur besitzen, dann existieren viel effizientere Beweise.

Im Folgenden nehmen wir immer an, dass $(G, \star)$ und $(H, \otimes)$ Gruppen sind mit Gruppenoperationen $\star$ respektive $\otimes$, und dass $f : G \to H$ ein Gruppenhomomorphismus ist, d.h. dass

$$f(x \star y) = f(x) \otimes f(y)$$

gilt. Die Abbildung f muss nicht bijektiv sein, d.h. die Gruppe H kann viel kleiner sein als die Gruppe G. Wir nehmen ferner an, dass die Gruppenoperationen in G und in H effizient berechenbar sind. Weil f eine Einwegfunktion ist wird f *Einweg-Gruppenhomomorphismus* genannt. Wegen der effizienten Berechenbarkeit der Operation in H kann man für gegebene Werte $[x]$ und $[y]$ effizient $[x \star y] = [x] \otimes [y]$ berechnen, ohne x, y oder $x \star y$ zu kennen. Oft wird im Folgenden G die (additive) Gruppe $\mathbf{Z}_q$ für eine Primzahl q sein, oder $G = \mathbf{Z}_q \times \mathbf{Z}_q$, oder $G = \mathbf{Z}_q \times \mathbf{Z}_q \times \mathbf{Z}_q$.

[35] Wie schon früher bemerkt kann Kollisionsfreiheit nur im Kontext einer Klasse von Funktionen sinnvoll definiert werden.

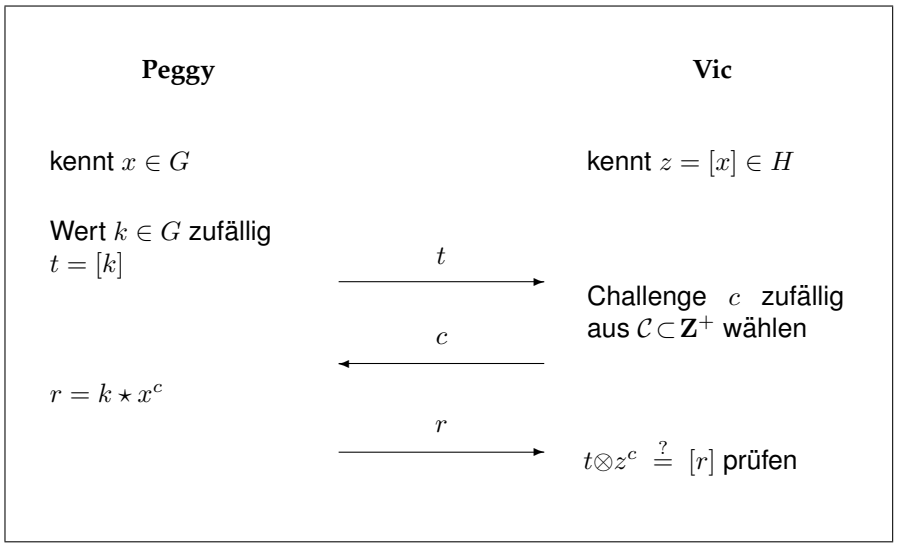


Abb. 1.11: Beweis des Wissens eines Arguments x eines Einweg-Gruppenhomomorphismus $(G, \star) \to (H, \otimes)$ für ein gegebenes $z \in H$. Ein hochgestellter Exponent (hier c) bedeutet in beiden Gruppen die c-fache Verknüpfung des Elementes mit sich selbst. (Für additive geschriebene Gruppen, wie die weiter unten betrachtete Gruppe $\mathbf{Z}_q$, müsste man x^c durch cx ersetzen.) Der Challengebereich $\mathcal{C}$ kann gleich $[0, \ldots, |G| - 1]$ oder eine Teilmenge davon sein, wenn $|G|$ bekannt ist.

Zum Teil werden wir nicht nur an der Addition in $\mathbf{Z}_q$, sondern auch an der Multiplikation interessiert sein. In diesem Fall schreiben wir $GF(q)$ für den Körper statt $\mathbf{Z}_q$ für die additive Gruppe dieses Körpers.

1.9.2 Ein allgemeiner Beweis von Wissen

Das Protokoll in Abb. 1.11 ist eine Abstraktion bisher betrachteter interaktiver Beweise. Damit es ein Beweis von Wissen ist, muss eine zusätzliche Bedingung erfüllt sein.

Theorem 1.5. *Falls es ℓ und u gibt, sodass*

(1) $\mathrm{ggT}(c_1 - c_2, \ell) = 1$ *für alle $c_1, c_2 \in \mathcal{C}$ (mit $c_1 \neq c_2$), und*

(2) $[u] = z^\ell$,

so ist die Protokollrunde in Abb. 1.11 C-simulierbar (Def. 1.7) und 2-extrahierbar (Def. 1.11). Ein Protokoll bestehend aus genügend vielen Runden ist ein Beweis von Wissen sowie zero-knowledge, falls $|\mathcal{C}|$ polynomial beschränkt ist.

Beweis. Es gilt 2-Extrahierbarkeit: Aus den Antworten r_1 und r_2, welche $[r_1] = t \otimes z^{c_1}$ und $[r_2] = t \otimes z^{c_2}$ für zwei verschiedene Challenges c_1 und c_2 erfüllen, kann ein x', welches $[x'] = z$ erfüllt, berechnet werden mittels

$$x' = u^a \star \left(r_2^{-1} \star r_1\right)^b,$$

wobei a und b mit dem Euklid'schen ggT-Algorithmus so berechnet werden können, dass

$$a\ell + b(c_1 - c_2) = 1$$

gilt. Wir verwenden

$$[r_2^{-1} \star r_1] = [r_2^{-1}] \otimes [r_1] = z^{-c_2} \otimes t^{-1} \otimes t \otimes z^{c_1} = z^{-c_2} \otimes z^{c_1} = z^{c_1 - c_2}.$$

Dass wirklich $[x'] = z$ gilt, sieht man folgendermassen:

$$\begin{aligned}[x'] &= [u^a \star (r_2^{-1} \star r_1)^b] \\ &= [u]^a \otimes [r_2^{-1} \star r_1]^b \\ &= (z^\ell)^a \otimes (z^{c_1-c_2})^b \\ &= z^{a\ell+b(c_1-c_2)} \\ &= z.\end{aligned}$$

Aus Theorem 1.3 folgt, dass das Protokoll ein Beweis von Wissen ist. Die zero-knowledge Eigenschaft folgt aus Theorem 1.2. □

1.9.3 Zwei Spezialfälle des Protokolls: Schnorr und GQ

Sei H eine Gruppe der Ordnung $|H| = q$ (q prim), in der die Berechnung diskreter Logarithmen schwierig ist, und sei h ein Generator von H. Das Schnorr-Protokoll ist ein Spezialfall des Protokolls in Abb. 1.11 für die additive Gruppe $G = \mathbf{Z}_q$ (q prim) und den Einweg-Gruppenhomomorphismus

$$G \to H: \quad x \mapsto [x] = h^x.$$

Der Challengebereich $\mathcal{C}$ kann eine beliebige Teilmenge von $[0, \ldots, q-1]$ sein (aber polynomial beschränkt, wenn das Protokoll zero-knowledge sein soll.) Mit $\ell = q$ gilt $z^\ell = 1$ für alle $z \in H$, also erfüllt $u := 0$ obige Bedingung $[u] = z^\ell$. Weil $\ell = q$ prim ist, gilt $\mathrm{ggT}(c_1 - c_2, \ell) = 1$ für alle verschiedenen $c_1, c_2 \in \mathcal{C}$.

Das Guillou-Quisquater Protokoll ist ebenfalls ein Spezialfall des Protokolls in Abb. 1.11 für $G = H = \mathbf{Z}_m^*$ und den Einweg-Gruppenhomomorphismus

$$\mathbf{Z}_m^* \to \mathbf{Z}_m^*: \quad x \mapsto [x] = x^e$$

(e fix und prim), wobei $\mathcal{C} \subseteq [0, \ldots, e-1]$. Hier kann $\ell = e$ gewählt werden, denn mit $u := z$ ist nun trivialerweise $[u] = z^\ell$.

1.9.4 Beweis der Kenntnis einer Repräsentation

Der Beweis, dass man eine Repräsentation eines Elements $z \in H$ (mit $|H| = q$) bezüglich zweier Generatoren h_1 und h_2 kennt (siehe Beispiel 1.13), ist ebenfalls ein Spezialfall des Grundprotokolls für den Einweg-Gruppenhomomorphismus

$$\mathbf{Z}_q \times \mathbf{Z}_q \to H: \quad (x_1, x_2) \mapsto [(x_1, x_2)] = h_1^{x_1} h_2^{x_2}$$

Die Bedingung aus Theorem 1.5 ist mit $\ell = q$ und $u := (0, 0)$ erfüllt da $[(0, 0)] = 1 = z^\ell$ für jedes $z \in H$.

Dieser Beweis entspricht dem Beweis, dass Peggy ein Pedersen-Commitment (siehe Abschnitt 1.4.3) öffnen kann. Das Protokoll kann einfach verallgemeinert werden auf Repräsentationen bezüglich mehrerer Generatoren $h_1, \ldots, h_k$. Ebenso kann man auch beweisen, dass man Werte $x_1, \ldots, x_m$ kennt, die mehrere Repräsentationen bezüglich der Generatoren $h_1, \ldots, h_k$ erfüllen (siehe folgendes Beispiel).

Beispiel 1.14. Seien 3 Generatoren h_1, h_2, h_3 von H gegeben. Peggy kann beweisen, dass sie für gegebene Werte $z_1, z_2 \in H$ Werte $x_1, x_2, x_3, x_4 \in \mathbf{Z}_q$ kennt mit $z_1 = h_1^{x_3} h_2^{x_1}$ und $z_2 = h_1^{x_2} h_2^{x_4} h_3^{x_1}$. Der Leser kann sich davon selbst überzeugen.

Eine weitere Verallgemeinerung liefert einen Beweis, dass Peggy Werte $x_1, \ldots, x_m$ kennt, sodass für gegebene Werte $z_1, \ldots, z_k$ und für gegebene lineare (über $GF(q)$) Funktionen $l_1, \ldots, l_k$ gilt $z_i = [l_i(x_1, \ldots, x_m)]$ für $i = 1, \ldots, k$. Die linearen Funktionen $l_1, \ldots, l_k$ können als die Zeilen einer Matrix A geschrieben werden, wobei gilt $z_i = [y_i]$ für $i = 1, \ldots, k$ mit

$$(y_1, \ldots, y_k) = A(x_1, \ldots, x_m).$$

1.9.5 Multiplikation homomorpher Commitments

Commitments sind eine wichtige Primitive für die sichere Multi-Party Berechnung. Sie können verwendet werden, um einen Spieler zu seinen geheimen Werten (sein Input oder ein nur diesem Spieler bekanntes Zwischenresultat) verpflichtet zu halten, ohne dass er diese öffnen muss. Falls man aus zwei Commitments ein neues Commitment (des entsprechenden Spielers) zur Summe oder zum Produkt (in einem bestimmten endlichen Körper) der beiden Werte berechnen kann, so kann der Spieler algebraische Funktionen seiner geheimen Werte berechnen, ohne dass er betrügen kann, und ohne dass die anderen Spieler etwas über die Werte erfahren. Wenn er später einen Wert öffnen möchte, dann ist er bereits dazu committet.

Die verwendeten Commitment-Verfahren sind homomorph bezüglich der Addition, d.h. ein Commitment zur Summe zweier Werte kann ohne den committeten Spieler berechnet werden. Für die Multiplikation geht das nicht. Wir beschreiben deshalb hier und im Folgenden Abschnitt Multiplikationsprotokolle für Commitments als Spezialfälle des generischen Protokolls. Die Verwendung homomorpher Commitments in der Multi-Party Berechnung wird in Kapitel 2.8 detailliert diskutiert. Hier geht es nur um einen dabei wichtigen Schritt, den Beweis der korrekten Multiplikation committeter Werte.

Wir betrachten zunächst wieder den Fall $G = \mathbf{Z}_q$ (q prim) und $[x] = h^x$ für einen fixen Generator h der Gruppe H mit $|H| = q$. Wir betrachten aber hier den Körper $GF(q)$, nicht nur dessen additive Gruppe $\mathbf{Z}_q$, weil wir ja auch an der Multiplikation modulo q interessiert sind.

Dieses Commitment Verfahren hat allerdings im Kontext der Multi-Party Berechnung den gravierenden Nachteil (wie jedes andere deterministische Commitment-Verfahren), dass alle möglichen Werte für x durchprobiert werden können. Ist die Anzahl möglicher Werte von x klein (z.B. wenn x binär ist), dann bietet dieses Verfahren keine Geheimhaltung. Interessanter sind für uns deshalb Commitments vom Typ H (siehe nächster Abschnitt). Dieses Beispiel diskutieren wir aus didaktischen Gründen, als Vorbereitung für den nächsten Abschnitt.

Zudem kann das folgende Protokoll auch angesehen werden als Beweis, dass ein gegebener Diffie-Hellman Schlüssel aus zwei gegebenen Public Keys resultiert. Diese Tatsache ist ja bekanntlich nicht effizient verifizierbar. Das Problem, zu entscheiden, ob ein gegebener Diffie-Hellman Schlüssel zu zwei gegebenen Public Keys gehört oder rein zufällig ist, ist im Allgemeinen im Falle einer Gruppe mit primer Ordnung sehr schwierig und ist als Diffie-Hellman Entscheidungsproblem bekannt. Mit dem folgenden Protokoll kann Peggy dies beweisen, ohne den Diffie-Hellman Schlüssel oder die zu den Public Keys gehörenden geheimen Schlüssel aufdecken zu müssen.

Gegeben seien also drei Werte $A = [a]$, $B = [b]$ und $C = [c]$, und wir möchten beweisen, dass $c = ab \pmod q$, d.h. dass A, B und C ein sogenanntes Diffie-Hellman Tripel bilden.

Zu diesem Zweck definieren wir einen neuen Einweg-Gruppenhomomorphismus, mit $[\![.]\!]$ bezeichnet, der abhängig von B und mittels des Einweg-Gruppenhomomorphismus $[.]$ definiert ist, wobei b als Konstante erscheint:

$$GF(q) \to H \times H : \quad x \mapsto [\![x]\!] = ([x], [xb]) = (h^x, B^x).$$

Man beachte, dass $[xb]$ wegen der homomorphen Eigenschaft aus $B = [b]$ und x berechnet werden kann, ohne b zu kennen.

Dies liefert direkt den gewünschten Beweis als weiteren Spezialfall des allgemeinen Protokolls: Peggy beweist, dass sie einen Wert x kennt (nämlich $x = a$) mit

$$[\![x]\!] = (A, C).$$

Wegen der Wahl des Einweg-Gruppenhomomorphismus $[\![.]\!]$ ist klar, dass der in C enthaltene Wert c gleich b mal dem in A enthaltenen Wert a ist, wobei b durch B definiert ist.

Das Protokoll beweist nicht, dass der Beweiser B öffnen kann (d.h. b kennt mit $B = [b]$) oder C öffnen kann, sondern nur, dass er A öffnen kann und dass $c = ab$, wobei b und c durch $B = [b]$ und $C = [c]$ definiert sind. Ist hingegen aus dem Kontext klar, dass der Beweiser B öffnen kann, dann folgt daraus natürlich, dass er auch C öffnen kann.

1.9.6 Multiplikationsprotokoll für Pedersen-Commitments

Betrachten wir das Pedersen Commitment Verfahren aus Abschnitt 1.4.3, das natürlich nicht nur für Bits, sondern für beliebige Werte aus $GF(q)$ definiert ist. Die Notation ist bewusst leicht anders gewählt.

Für eine zyklische Gruppe H seien zwei Generatoren g und h gegeben, so dass der diskrete Logarithmus von h zur Basis g unbekannt (und schwierig zu berechnen) ist. Der Wert $x \in GF(q)$ wird committet, indem $\rho_x \in GF(q)$ zufällig gewählt wird und als Commitment für x der Wert $g^x h^{\rho_x}$ berechnet wird. Der Beweis aus Abschnitt 1.9.4, dass man eine Repräsentation kennt, ist ein Beweis, dass man ein Commitment öffnen kann. Der diesem Commitment zu Grunde liegende Einweg-Gruppenhomomorphismus ist

$$GF(q) \times GF(q) \to H: \ (x, \rho_x) \mapsto [(x, \rho_x)] = g^x h^{\rho_x}.$$

Da dieses Commitment Verfahren informationstheoretisch geheim ist, ist der in einem Wert $X \in H$ committete Wert nicht bestimmt. Dieser ist nur bestimmt durch die Art, wie der Committer X öffnen kann.

Drei Commitments $A, B, C \in H$ seien gegeben. Wir gehen im Folgenden davon aus, dass aus dem Kontext klar ist, dass Peggy das Commitment B öffnen kann (als (b, ρ_b)). Im Folgenden diskutieren wir ein Protokoll, das Peggy erlaubt, zu beweisen, dass sie C öffnen kann als (c, ρ_c), und dass $c = ab$ gilt. Das Protokoll beweist auch, dass sie A öffnen kann (als (a, ρ_a)), obwohl dies in der Regel auch aus dem Kontext schon klar ist.

Zu diesem Zweck definieren wir wiederum einen neuen Einweg-Gruppenhomomorphismus, abhängig von B, mittels

$$GF(q)^3 \to H \times H: \ (x, \rho_x, \sigma_x) \mapsto [\![(x, \rho_x, \sigma_x)]\!] = ([(x, \rho_x)], [(xb, x\rho_b + \sigma_x)]),$$

wobei die zweite Komponente mittels

$$[(xb, x\rho_b + \sigma_x)] = B^x h^{\sigma_x}$$

bei Kenntnis von B (aber ohne Kenntnis von b und ρ_b mit $B = [(b, \rho_b)]$) berechnet werden kann.

Dies liefert direkt den gewünschten Beweis als weiteren Spezialfall des allgemeinen Protokolls: Es ist zu beweisen, dass man ein Tripel (x, ρ_x, σ_x) kennt, so dass

$$[\![(x, \rho_x, \sigma_x)]\!] = (A, C).$$

Dieser Beweis ist erfolgreich durchführbar für die Wahl

$$(x, \rho_x, \sigma_x) = (a, \rho_a, \rho_c - a\rho_b),$$

wie sich einfach verifizieren lässt.

Für den Fall, dass nicht klar ist, dass der Beweiser B öffnen kann, muss das Protokoll dies auch beweisen. Dies kann erreicht werden mit der früher diskutierten Methode, zwei Beweise zu kombinieren (siehe Übungen).

1.A Quadratische Reste

Definition 1.14. Eine Zahl a heisst *quadratischer Rest* modulo m wenn $\mathrm{ggT}(a, m) = 1$ und wenn es ein w gibt mit

$$w^2 \equiv a \pmod{m}.$$

Ein solches w heisst *Quadratwurzel* modulo m von a. Wenn $\gcd(a, m) = 1$, aber es keine Quadratwurzel modulo m von a gibt, so heisst a *quadratischer Nichtrest* modulo m. Wir bezeichnen die Mengen der quadratischen Reste bzw. Nichtreste mit QR_m bzw. QNR_m.

Das folgende Theorem ist einfach zu beweisen. Es gilt noch allgemeiner für zyklische Gruppen mit gerader Ordnung.

Theorem 1.6. *Für jede ungerade Primzahl p gilt $|\mathrm{QR}_p| = |\mathrm{QNR}_p| = (p-1)/2$ und jedes $a \in \mathrm{QR}_p$ hat genau zwei Quadratwurzeln modulo p.*

Es gibt effiziente Algorithmen für die Berechnung von Quadratwurzeln modulo einer Primzahl. Für Primzahlen der Form $p \equiv 3 \pmod 4$ ist diese Berechnung speziell einfach (siehe Übungen).

Definition 1.15. Für eine ungerade Primzahl p ist das *Legendre Symbol* von a modulo p definiert als

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{wenn } a \in \mathrm{QR}_p \\ -1 & \text{wenn } a \in \mathrm{QNR}_p \\ 0 & \text{wenn } a \equiv 0 \pmod p. \end{cases}$$

Theorem 1.7. *Für jede ungerade Primzahl p gilt*

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod p.$$

Beweis. Der Fall $a \equiv 0 \pmod p$ ist trivial. Sei g ein Generator von $\mathbf{Z}_p^*$. Es gilt $g^{(p-1)/2} = -1$. Jedes $a \in \mathbf{Z}_p^*$ kann geschrieben werden als $a = g^i$ für ein $i \in [0, p-2]$, wobei a bei geradem i ein quadratischer Rest und bei ungeradem i ein quadratischer Nichtrest ist, d.h. $\left(\frac{a}{p}\right) = (-1)^i$. Auf der anderen Seite gilt $a^{(p-1)/2} = g^{i(p-1)/2} = (-1)^i$. □

Definition 1.16. Für eine Zahl $m = p_1 \cdots p_r$, wobei die p_i (nicht notwendigerweise verschiedene) Primzahlen sind, ist das *Jacobi Symbol* von a modulo m definiert als

$$\left(\frac{a}{m}\right) = \left(\frac{a}{p_1}\right) \cdots \left(\frac{a}{p_r}\right).$$

Das Jacobi Symbol ist eine strikte Erweiterung des Legendre Symbols. Es gibt einen effizienten Algorithmus für die Berechnung des Jacobi Symbols, auch wenn die Faktorisierung von m nicht bekannt ist. Interessanterweise ist dieser Algorithmus selbst für Primzahlen viel effizienter, als $a^{(p-1)/2}$ zu berechnen. Der Algorithmus ist in [vT88] beschrieben.

Wir betrachten im Folgenden einen RSA-Modulus der Form $m = pq$. Eine Zahl a ist genau dann quadratischer Rest modulo m, wenn dies auch modulo p und modulo q gilt:

$$a \in \mathrm{QR}_m \iff (a \in \mathrm{QR}_p) \wedge (a \in \mathrm{QR}_q).$$

Es sind also nur $1/4$ aller Zahlen in $\mathbf{Z}_m^*$ quadratische Reste, und für alle quadratischen Reste a gilt $\left(\frac{a}{m}\right) = 1$.

Definition 1.17. Sei $m = pq$ ein Modulus mit unbkannter Faktorisierung. Das *Quadratische Rest Problem (QR-Problem)* ist, für ein gegebenes a mit $\left(\frac{a}{m}\right) = 1$ zu entscheiden, ob a ein quadratischer Rest ist oder nicht.

Für die Lösung des QR-Problems ist keine effizientere Methode bekannt, als m zu faktorisieren. Der Leser kann sich als Übung davon überzeugen, dass selbst eine Approximation der Lösung mit Wahrscheinlichkeit signifikant grösser als 50% schwierig ist, wenn das QR-Problem selbst schwierig ist.

1.B Kurzrepetitorium der Komplexitätstheorie

1.B.1. Formale Sprachen und Entscheidungsprobleme

Definition 1.18. Eine *formale Sprache* L ist eine Teilmenge der binären Strings endlicher Länge: $L \subseteq \{0,1\}^*$.

Jedes wohldefinierte Entscheidungsproblem (z.B. Graphenisomorphismus, Hamiltonsche Kreise, etc.) kann formuliert werden als das Problem, zu entscheiden, ob ein gegebener String x (auch Wort genannt) in einer bestimmten formalen Sprache L enthalten ist.

Umgekehrt definiert jede formale Sprache L ein entsprechendes Entscheidungsproblem, nämlich für ein gegebenes Eingabewort $x \in \{0,1\}^*$ zu entscheiden, ob $x \in L$ oder $x \notin L$.

Beispiel 1.15. Die Sprache PRIMES ist die Menge aller Strings, welche die binäre Darstellung einer Primzahl sind. Es gibt natürlich mehrere Darstellungsarten für ganze Zahlen und man muss deshalb annehmen, dass eine bestimmte Darstellungsart verwendet wird, die aber nicht explizit beschrieben werden muss. Verschiedene Darstellungsarten sind in der Regel äquivalent in dem Sinn, dass verschiedene Darstellungen effizient ineinander transformiert werden können.

Beispiel 1.16. Das Graphenisomorphieproblem könnte wie folgt als Sprache codiert werden. Ein Graph wird codiert durch die Knotenzahl n (präfixfrei binär codiert), gefolgt von den n^2 Elementen der Adjazenzmatrix. Ein Graphenpaar wird durch das Aneinanderfügen der entsprechenden Strings beider Graphen codiert. Die Sprache GI ist z.B. definiert als die Menge aller binärer Strings, die korrekt ein Graphenpaar codieren, wobei die beiden Graphen isomorph sind. Die meisten binären Strings stellen nicht einmal Graphenpaare dar, geschweige denn isomorphe, und sind deshalb nicht in der Sprache GI.

1.B.2. Berechnungsmodelle und Turing Maschinen

Um die Schwierigkeit eines Berechnungsproblems (z.B. das Zugehörigkeitsproblem für eine Sprache L) zu definieren, muss zuerst definiert werden, welche Berechnungsverfahren resp. Algorithmen zur Lösung erlaubt sind, und was es genau bedeutet, das Problem zu lösen. Zudem muss für dieses Berechnungsmodell der Begriff der Laufzeit (z.B. Anzahl Elementarschritte) definiert werden. Eine untere Schranke für die Schwierigkeit eines Problems zu beweisen bedeutet, zu zeigen, dass es kein Verfahren (z.B. Algorithmus) innerhalb des Modells gibt mit Laufzeit kleiner als die Schranke.

Das Konzept eines Berechnungsverfahrens wird normalerweise formalisiert als *Turing-Maschine (TM)* mit einem endlichen Zustandsraum, aber einem unbeschränkten Band (Speicher). Die technischen Details einer TM spielen hier keine Rolle[36] und man kann sich einfach einen Algorithmus auf einem gebräuchlichen Computer denken.

Der Input einer TM steht zu Beginn der Berechnung auf ihrem Band. Es gibt grundsätzlich zwei Arten, wie eine TM Output liefern kann: entweder durch den Zustand, in dem die Berechnug endet (in dem die TM anhält), oder durch den String, der am Ende der Berechnung auf dem Band steht. Für Entscheidungsprobleme betrachtet man die erste Art. Die TM hat einen speziellen *akzeptierenden Haltezustand* H. Sie kann sich für einen gegebenen Input x auf mindestens drei verschiedene Arten verhalten:

(1) im Zustand H anhalten,

(2) in einem anderen (nicht akzeptierenden) Haltezustand anhalten,

(3) nie anhalten (Endlos-Schleife).

Definition 1.19. Eine TM *akzeptiert* ein Wort x, wenn sie bei Eingabe von x den akzeptierenden Haltezustand erreicht (Variante (1)). Die Menge der von einer TM akzeptierten Worte ist die von ihr *akzeptierte Sprache* (resp. *erkannte* Sprache).[37]

Das *Erkennungsproblem* einer Sprache ist nicht symmetrisch definiert: Im Fall $x \notin L$ wird nur verlangt, dass die TM nicht im akzeptierenden Haltezustand hält, aber ob Variante (2) oder (3) auftritt ist nicht festgelegt. Insbesondere muss das Konzept des Nichtakzeptierens eines Wortes x nicht definiert sein. Im Gegensatz dazu ist das *Entscheidungsproblem der Zugehörigkeit zu einer Sprache L* so definiert, dass die Möglichkeit (3) ausgeschlossen ist, d.h. die TM muss immer anhalten in einem akzeptierenden resp. nicht akzeptierenden Zustand. Im Allgemeinen ist also für eine fixe Sprache L das Entscheidungsproblem schwieriger als

[36] Wir verweisen auf [Weg] für eine genaue Definition einer TM.

[37] Das Halteproblem für TM ist unentscheidbar und deshalb lässt sich die von einer beliebigen TM akzeptierte Sprache im Allgemeinen nicht bestimmen.

das Erkennungsproblem, d.h. für letzteres gibt es in der Regel eine TM mit kleinerer Laufzeit. Das Komplement einer Sprache L, d.h. die Sprache $\overline{L} := \{x : x \notin L\}$ zu erkennen, ist nicht notwendigerweise gleich schwierig wie die Erkennung von L.

1.B.3. Die Sprachenklassen P und NP

Für eine TM, welche die Sprache L akzeptiert, ist für jedes $x \in L$ definiert, wie viele Schritte $s(x)$ die Berechnung dauert, bis der Haltezustand erreicht ist. Dadurch lässt sich eine Laufzeitfunktion $t : \mathbf{Z} \to \mathbf{Z}$ definieren als

$$t(n) := \max\big(s(x) : x \in L,\ |x| \leq n\big).$$

Definition 1.20. Eine Laufzeitfunktion $t(n)$ heisst *polynomial*, wenn es ein Polynom $p(n)$ gibt, so dass $t(n) \leq p(n)$ für alle $n > n_0$. Äquivalente Formulierung: $\exists c, n_0\ \forall n > n_0 : t(n) \leq n^c$.

Definition 1.21. **P** ist die Klasse der Sprachen L, für die es eine L akzeptierende TM mit polynomialer Laufzeitfunktion gibt.

Definition 1.22. Eine *nichtdeterministische* TM kann beliebig viele Berechnungen parallel durchführen[38], und ein Wort wird akzeptiert, wenn mindestens eine dieser parallelen Berechnungen im akzeptierenden Zustand endet.

Definition 1.23. **NP** ist die Klasse der Sprachen L, für die es eine nichtdeterministische TM mit polynomialer Laufzeitfunktion gibt, die L akzeptiert.

Eines der berühmtesten offenen Probleme der theoretischen Informatik, und sogar der gesamten Mathematik, ist die Frage, ob $\mathbf{P} = \mathbf{NP}$ oder $\mathbf{P} \neq \mathbf{NP}$.

Definition 1.24. Eine Sprache L heisst **NP**-*vollständig* wenn $L \in \mathbf{NP}$ ist, und es für jede Sprache $L' \in \mathbf{NP}$, eine von einer TM in polynomialer Zeit berechenbare Funktion f gibt, so dass $x \in L' \Leftrightarrow f(x) \in L$.

Die folgende alternative Charakterisierung der Sprachenklasse **NP** ist für unseren Zweck natürlicher (siehe Übungen).

Theorem 1.8. *Die Sprache L ist in* **NP** *genau dann, wenn es für jedes $x \in L$ einen in polynomialer Zeit verifizierbaren Beweis für die Zugehörigkeit gibt. Präziser: es gibt eine in polynomialer Zeit berechenbare Funktion $f_L : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$, so dass es für jedes $x \in L$ (und nur für $x \in L$) einen Zeugen (witness) $w \in \{0,1\}^*$ gibt mit $f_L(x,w) = 1$ und $|w|$ polynomial in $|x|$.*

1.B.4. Verschwindende und überwältigende Wahrscheinlichkeit

Definition 1.25. Eine Funktion $f : N \to R^+$ heisst *verschwindend* (vernachlässigbar, negligible) falls sie schneller als invers zu jedem Polynom abfällt. Formal: $\forall c > 0\ \exists n_0\ \forall n > n_0 :\ \ f(n) < 1/n^c$.

Definition 1.26. Eine Wahrscheinlichkeit $p(n)$ (als Funktion eines Grössenparameters n) heisst *überwältigend* (overwhelming), wenn $1 - p(n)$ verschwindend ist.

Definition 1.27. Eine Wahrscheinlichkeit $p(n)$ heisst *nicht vernachlässigbar*[39] (non-negligible, noticeable), falls sie für alle genügend grossen n grösser als eine invers polynomial abfallende Funktion ist. Formal: $\exists c, n_0\ \forall n > n_0 :\ \ p(n) \geq 1/n^c$.

[38] In jeder Verzweigung der Berechnung können beide Pfade parallel verfolgt werden.

[39] Diese Terminologie ist leider unglücklich, da "nicht vernachlässigbar" nicht die logische Negation von "vernachlässigbar" ist.

Kapitel 2

Multi-Party Computation

2.1 Einleitung und Motivation

Das Forschungsgebiet *Multi-Party Computation* (MPC) innerhalb der Kryptographie wurde vor etwa 15 Jahren begründet und ist das vielleicht faszinierendste Thema innerhalb der modernen Kryptographie. Das Problem wurde von Yao [Yao82] 1982 erstmals vorgeschlagen, die erste allgemeine Lösung wurde aber erst 1987 von Goldreich, Micali und Wigderson [GMW87] publiziert. In den letzten Jahren hat sich die Forschung auf diesem Gebiet wieder stark intensiviert.

2.1.1 Problemstellung

Bei der Multi-Party Computation geht es um das Problem, wie mehrere Personen oder Entitäten miteinander eine Berechnung durchführen (kooperieren) können, *ohne sich gegenseitig zu vertrauen*. Dabei können gewisse Entitäten versuchen zu betrügen; dies muss aber durch das Protokoll abgefangen werden. Ein bekanntes Beispiel einer MPC ist eine Abstimmung durch mehrere Personen ohne eine gemeinsame, vertrauenswürdige Instanz zur Auszählung der Stimmen, und ohne physikalische Urnen zur Sicherung der Geheimhaltung der Stimmen. Ein anderes Beispiel ist das sogenannte Millionärsproblem: zwei Millionäre möchten lediglich mittels Durchführung eines Protokolls zwischen ihnen beiden herausfinden, wer reicher ist, ohne einander irgendwelche zusätzliche Information über ihr eigenes Vermögen zu geben.

Ein weiteres Beispiel sind kombinierte Abfragen in unabhängigen Datenbanken. Nehmen wir an, eine staatliche Stelle möchte täglich Statistiken über die Wirtschaftsdaten aller Unternehmen erheben. Die Unternehmen sind bereit, die Daten für die Berechnung einer Statistik einzubringen, trauen aber dem Staat nicht, dass die Daten nicht zu anderen Zwecken verwendet werden (z.B. einer Konkurrenzfirma zur Verfügung stellen, oder für spätere Konsistenzprüfungen bei der Einsendung der Steuererklärung verwenden).

2.1.2 Simulation einer vertrauenswürdigen Instanz

All diese Kooperationsprobleme könnten einfach durch die Verwendung einer vertrauenswürdigen Instanz gelöst werden. Oft ist aber keine solche für alle Teilnehmer vertrauenswürdige Instanz vorhanden. Das Ziel der MPC ist es, eine solche Instanz durch ein Protokoll zwischen den Teilnehmern zu *simulieren*. Es geht dabei nicht nur um die Lösung bestimmter konkreter Probleme, wie in den erwähnten Beispielen, sondern allgemeiner um die Minimierung des notwendigen Vertrauens für die Lösung eines Problems.

Ein Spezialfall der skizzierten Simulation einer vertrauenswürdigen Instanz ist *sichere Funktionsevaluation* (secure function evaluation): Mehrere Parteien (oder Spieler) haben je einen geheimen Input und möchten eine spezifizierte Funktion der Inputs berechnen, so dass jede Partei das korrekte Resultat (aber nicht mehr) erfährt. In einer etwas allgemeineren Version ist die Funktion mehrwertig, wobei jeder Spieler einen Wert erhält. In diesem Kapitel beschränken wir uns auf dieses spezielle Szenario der MPC.

2.1.3 Gegnermodell

Wir betrachten zwei Typen von Gegnern (resp. unehrlichen Spielern).

- **passive** Gegner führen das Protokoll korrekt durch, versuchen aber nebenbei und am Ende des Protokolls Information über die Inputs der ehrlichen Spieler zu erhalten.
- **aktive** Gegner weichen beliebig vom vorgeschriebenen Protokoll ab und versuchen gemeinsam, die Berechnung zu verfälschen und/oder geheime Information zu erhalten.

Oft ist die Sichtweise, dass ein (zentraler) Gegner vorhanden ist, der gewisse Spieler kontrollieren kann (passiv oder aktiv). Wir sprechen deshalb oft einfach vom Gegner. Wie schon in den vorherigen Kapiteln unterscheiden wir auch hier bezüglich der Rechenleistung des Gegners (limitiert oder unbeschränkt) und bezeichnen die Sicherheit entsprechend als **berechenmässig** oder **informationstheoretisch**.

Die Sicherheit bekannter Protokolle beruht auf der Annahme, dass der Gegner nicht mehr als eine bestimmte Anzahl resp. ein bestimmter Bruchteil (z.B. ein Drittel oder die Hälfte) der Spieler kontrolliert. Allgemeiner kann man eine Menge von Teilmengen der Spielermenge betrachten (eine sogenannte Gegnerstruktur), wobei der Gegner genau die Spieler einer dieser Mengen kontrollieren darf. Welche Teilmenge dies ist, ist a priori (und oft auch nach Durchführung des Protokolls) nicht bekannt. Die Angabe einer Schwelle t für die maximale Anzahl Gegner entspricht der speziellen Gegnerstruktur, die alle Teilmengen der Grösse t oder kleiner enthält.

2.1.4 Kommunikationsmodell

In Bezug auf die verfügbaren Kommunikationskanäle werden folgende Annahmen gemacht:

- Die Spieler können **paarweise sicher kommunizieren**. Geht es nur um berechenmässige Sicherheit, so können diese sicheren Kanäle mittels Public-Key Kryptographie und einer vorhandenden PKI erreicht werden. Im informationstheoretischen Szenario nehmen wir an, die paarweisen sicheren Kanäle seien entweder physisch gesichert oder z.B. mit Hilfe von One-Time Pad Verschlüsselung realisiert.
- Wenn man zusätzlich annimmt, dass ein **Broadcastkanal** verfügbar ist, dann lassen sich in der Regel stärkere Resultate zeigen als ohne einen solchen Kanal. Ein Broadcastkanal stellt sicher, dass alle Spieler den gleichen Wert erhalten, der über diesen Kanal versandt wird. Es ist für den Sender unmöglich, den Spielern unterschiedliche Werte zu senden. In der Regel (z.B. Internet) ist kein Broadcastkanal vorhanden, aber Broadcast kann unter bestimmten Voraussetzungen durch ein Protokoll zwischen den Spielern simuliert werden (vergleiche Abschnitt 2.6).

2.1.5 Sicherheitsanforderungen

Die Sicherheitsanforderungen an MPC-Protokolle sind:

- **Privacy.** Die allfälligen unehrlichen Spieler sollen keine Information über die Inputs der anderen Spieler erhalten, ausser was durch ihre eigenen Inputs und den Output sowieso bekannt ist.
- **Correctness.** Die unehrlichen Spieler können das Resultat (resp. die Resultate) der ehrlichen Spieler nicht verfälschen.
- **Fairness.** Ein Protokoll ist fair, wenn die unehrlichen Spieler zu keinem Zeitpunkt das Protokoll mit einem Vorteil abbrechen können. Mit anderen Worten, sobald sie irgendwelche Information über das Resultat erhalten haben, dann erhalten die ehrlichen Spieler das gesamte Resultat.
- **Robustness.** Ein Protokoll ist robust, wenn es gar nicht möglich ist, das Protokoll abzubrechen. Die Inputs werden einmal in ein Protokoll „eingegeben“ (ähnlich einem Commitment) und anschliessend können die Spieler das Protokoll durchführen, selbst wenn gewisse Spieler versuchen, dies zu verhindern.

Etwas präziser wird die Sicherheit eines MPC-Protokolls relativ zu einer *Spezifikation* definiert. Eine Spezifikation ist ein Protokoll zwischen den Spielern und einer vertrauenswürdigen Instanz (trusted third party, TTP), wobei diese TTP typischerweise die eigentliche Berechnung durchführt. Wir sagen, ein MPC-Protokoll berechnet eine Spezifikation, wenn die unehrlichen Spieler im MPC-Protokoll nur Dinge erreichen können, welche sie auch in der Spezifikation erreichen könnten (wobei natürlich die TTP immer ehrlich ist).

Beispiel 2.1. Die Spezifikation für eine Abstimmung wäre zum Beispiel, dass jeder Spieler seine Stimme der vertrauenswürdigen Instanz zuschickt, diese wählt unter allen erhaltenen Stimmen die gültigen Stimmen aus (z.B. Stimmen mit „0“ oder „1“), berechnet die Summe, und schickt das Resultat allen Spielern. In dieser Spezifikation können unehrliche Spieler allenfalls ihre Stimmen gezielt falsch wählen, können aber das Resultat nicht weiter verfälschen. Ausserdem erfahren sie nur das Endresultat, und keine Teilresultate oder sogar einzelne Stimmen. Das Ziel ist nun, ein MPC-Protokoll zu finden, welches die gleichen Eigenschaften besitzt wie diese Spezifikation, aber keine TTP verwendet.

2.1.6 Bekannte Resultate

Die klassischen Resultate für Berechnungen zwischen n Spielern sind:

- Kryptographisches Szenario, passive Gegner [GMW87]: Jede Funktion kann sicher berechnet werden, selbst wenn bis zu $t < n$ Spieler passive Gegner sind.
- Kryptographisches Szenario, aktive Gegner [GMW87]: Jede Funktion kann sicher berechnet werden, selbst wenn bis zu $t < n/2$ Spieler aktiv betrügen. Bei mehr Betrügern ist dies nicht mehr möglich.
- Informationstheoretisches Szenario, passive Gegner [BGW88, CCD88]: Jede Funktion kann sicher berechnet werden, selbst wenn bis zu $t < n/2$ Spieler passive Gegner sind. Bei mehr Gegnern ist dies nicht mehr möglich.
- Informationstheoretisches Szenario, aktive Gegner [BGW88, CCD88]: Jede Funktion kann sicher berechnet werden, selbst wenn bis zu $t < n/3$ Spieler betrügen. Bei mehr Betrügern ist dies nicht mehr möglich. Wenn ein Broadcastkanal zur Verfügung steht, dann ist die Schwelle $n/2$ [RB89, Bea91].
- In [HM97] wurden diese Resultate auf allgemeine Gegnerstrukturen verallgemeinert. Sichere MPC ist möglich genau dann, wenn sich in der Gegnerstruktur keine zwei (für Resultate mit Schwelle $n/2$) resp. drei (für Resultate mit Schwelle $n/3$) potentielle Gegnermengen zur ganzen Spielermenge ergänzen. Der kryptographische Fall wurde in [CDM00] bewiesen.

In der folgenden Tabelle werden die Resultate zusammengefasst:

Setting	**Adv. Type**	**Condition**	**Literature**
cryptographic	passive	$t < n$	[GMW87]
cryptographic	active	$t < n/2$	[GMW87]
information-theoretic	passive	$t < n/2$	[BGW88, CCD88]
information-theoretic	active	$t < n/3$	[BGW88, CCD88]
i.-t. with broadcast	active	$t < n/2$	[RB89, Bea91]

Beispiel 2.2. n Personen $P_1, \ldots, P_n$ möchten den Durchschnitt ihrer Saläre berechnen, ohne einander ihr Salär mitzuteilen. Dies kann in einem ersten Ansatz wie folgt erreicht werden: P_1 wählt zufällig eine grosse Zahl r, addiert sein Salär hinzu und sendet das Resultat (geheim) an P_2. Dieser addiert sein Salär hinzu und sendet das Resultat an P_3, usw., wobei P_n sein Resultat an P_1 sendet. P_1 subtrahiert r von der Summe und teilt den anderen Personen die Summe der Saläre mit.

Natürlich ist dieses Protokoll nicht sicher gegen aktive Gegner, da jeder das Resultat beliebig verfälschen kann. So kann ein Spieler den erhaltenen Wert verkleinern, was dem unmöglichen Fall eines negativen Salärs entspricht. Das Protokoll ist aber sicher gegen *einen* passiven Gegner. Zwei passive Gegner können hingegen unerlaubt Information erhalten: z.B. können P_i und P_{i+2} das Salär von P_{i+1} berechnen.

Beispiel 2.3. Das folgende Protokoll für die Berechnung der Summe von Zahlen (resp. des Durchschnittsalärs) ist sicher gegen beliebige $n-1$ passive Gegner. Wir gehen davon aus, dass eine obere Grenze m für die Summe bekannt ist (z.B. $m = 10^{12}$). Die Inputs der Spieler seien $x_1, ..., x_n$ und werden als Elemente der additiven Gruppe $\mathbb{Z}_m$ aufgefasst. Jeder Spieler P_i verteilt seinen Input x_i an alle Spieler so, dass sie nur gemeinsam den Wert rekonstruieren können. Das Protokoll umfasst die folgenden 4 Schritte:

1) Jeder Spieler P_i wählt n zufällige Werte $r_{i1}, \ldots, r_{in}$ aus $\mathbb{Z}_m$, so dass $x_i = r_{i1} + \ldots + r_{in}$.
2) Jeder Spieler P_i sendet (geheim) r_{ij} an Spieler P_j. (Natürlich muss sich P_i den Wert r_{ii} nicht wirklich zusenden. :–)
3) Jeder Spieler P_j berechnet die Summe $R_j = \sum_{i=1}^{n} r_{ij}$ der erhaltenen Werte und sendet ihn an alle anderen Spieler.
4) Jeder Spieler berechnet
$$S = \sum_{j=1}^{n} R_j \quad \Big(= \sum_{j=1}^{n}\sum_{i=1}^{n} r_{ij} \; = \; \sum_{i=1}^{n} x_i \Big).$$

Wenn in diesem Protokoll beliebige $t < n$ Spieler (passive Gegner) ihre Information zusammenbringen, so können sie lediglich die Summe der Saläre der verbleibenden Spieler berechnen, was sie auch aus ihren eigenen Inputs und dem Resultat tun könnten (selbst wenn eine vertrauenswürdige Instanz die Berechnung vorgenommen hätte). Insbesondere erhalten sie keine Information über die einzelnen Saläre oder über andere Teilsummen von Salären, ausser natürlich, was aus dem Total geschlossen werden kann.

Aktive Gegner zu tolerieren scheint viel schwieriger. Zwei Probleme sind das Verhindern negativer Inputs und die Berechnung nichtlinearer Funktionen (z.B. des Produktes zweier Werte). Dazu werden wir noch einige weitere Schritte und Techniken einführen müssen.

2.2 Oblivious Transfer und Zwei-Party Protokolle

2.2.1 Oblivious Transfer

Eines der elementarsten MPC-Protokolle zwischen zwei Spielern A und B ist *Oblivious*[1] *Transfer* (OT), das von Michael Rabin erstmals vorgeschlagen wurde. Es existieren unterschiedliche Varianten von OT, die kurz diskutiert werden.

Die einfachste Version ist Bit-OT: A sendet ein Bit b, aber B kann dieses Bit nur mit Wahrscheinlichkeit $1/2$ empfangen. Etwas formaler ist Bit-OT wie folgt spezifiziert: A besitzt als Input ein Bit b und B hat keinen Input. A erhält keinen Output aber B erhält als Output ein Indikatorbit c (zufällig durch das Protokoll gewählt, ohne Einflussmöglichkeit von A oder B) sowie das Bit b, falls $c = 1$. Die Idee ist, dass B das Bit b mit Wahrscheinlichkeit $1/2$ erfährt und mit WSK $1/2$ nichts über b erfährt, und dass A nicht weiss, welcher Fall eingetreten ist.

Beim 1-out-of-2 Bit-OT hat A zwei Inputbits b_0 und b_1 und B kann ein Selektionsbit c frei wählen. A erhält keinen Output und B erhält b_c, aber keine Information über b_{1-c}.

Beide Versionen gibt es auch als String-OT, d.h. statt um Bits handelt es sich bei den von A bereit gestellten Werten um Bitstrings. Bei 1-out-of-2 String-OT soll B über den nicht gewählten String keine Information erhalten. Eine weitere Verallgemeinerung ist der 1-out-of-k String-OT, bei dem A k statt nur 2 Inputstrings besitzt, von denen B einen frei auswählen kann und über die restlichen Strings keine Information erhält.

Kryptographische Realisierung von 1-out-of-2 String-OT: A wählt die Parameter zweier RSA-Systeme, $[m_0 = p_0 q_0, e_0]$ und $[m_1 = p_1 q_1, e_1]$, wobei m_0 und m_1 fast gleich gross sind (z.B. in den höchsten 100 Bits übereinstimmen). B wählt r zufällig aus $\mathbb{Z}_{m_c}$ (c ist das Auswahlbit) und berechnet

$$y = r^{e_c} \pmod{m_c}.$$

[1] Oblivious heisst auf deutsch vergesslich.

A berechnet $x_0 = y^{d_0} \pmod{m_0}$ und $x_1 = y^{d_1} \pmod{m_1}$ und sendet $u_0 = x_0 + s_0$ sowie $u_1 = x_1 + s_1$ an B, wobei s_0 und s_1 die beiden Inputstrings von A sind. B berechnet $s_c = u_c - r$.

Diese Realisierung von 1-out-of-2 String-OT ist (vermutlich) berechenmässig sicher, wenn A ein passiver Gegner ist oder wenn B ein aktiver (oder passiver) Gegner ist.

Kryptographische Realisierung von 1-out-of-k String-OT: 1-out-of-2 String-OT kann verwendet werden, um 1-out-of-k String-OT zu realisieren. Das folgende Protokoll realisiert berechenmässig sichereren[2] 1-out-of-k String-OT für eine Zweierpotenz $k = 2^t$: Zuerst wählt A t zufällige String-Paare $(s_{10}, s_{11}), \ldots, (s_{t0}, s_{t1})$. Dann definieren wir die k Schlüssel $K_{00\ldots0}, \ldots, K_{11\ldots1}$ als Konkatenation der entsprechenden Strings (also $K_{ij\ldots} = s_{1i}|s_{2j}|\ldots$). Schliesslich sendet A die String-Paare jeweils mittels 1-out-of-2 String-OT an B, und die Nachrichten $x_1, \ldots, x_k$ verschlüsselt als $E_{K_{00\ldots0}}(x_1)$, …, $E_{K_{11\ldots1}}(x_k)$. B empfängt die Strings entsprechend seiner Auswahl, und kann nun exakt einen Schlüssel $K_{ij\ldots}$ vollständig rekonstruieren und somit nur eine Nachricht entschlüsseln.

Dieses Protokoll für 1-out-of-k String-OT ist berechenmässig sicher, falls zum einen das verwendete 1-out-of-2 String-OT Protokoll sicher ist, und zum anderen eine Verschlüsselung mit E selbst dann sicher ist, wenn alle Teil-Strings des verwendeten Schlüssels bis auf einen bekannt sind.

2.2.2 Berechnung einer allgemeinen Funktion durch zwei Spieler

Wir wenden uns nun dem allgemeinen MPC-Problem mit 2 Spielern zu. Eine Funktion $f : \{0,1\}^s \times \{0,1\}^t \to \{0,1\}^u$ mit zwei Argumenten sei gegeben. A und B haben Inputs x_1 und x_2 und möchten gemeinsam den Funktionswert $f(x_1, x_2)$ berechnen, ohne sich x_1 oder x_2 mitzuteilen. Dies kann einfach mittels 1-out-of-k String-OT gelöst werden: A sendet B per 1-out-of-k String-OT die $k = 2^t$ Funktionswerte $f(x_1, .)$ für alle Argumente x_2. B wählt den Wert mit Index x_2 aus, also $f(x_1, x_2)$, und teilt das Resultat A mit.

Dieses Protokoll ist sicher gegen einen passiven Gegner. Sowohl A als auch B kann mit einer aktiven Strategie betrügen (wie?). Das Protokoll ist exponentiell in der Länge des Inputs x_2 und für typisch auftretende Funktionen also nicht anwendbar. Effizientere und sicherere Protokoll werden später in diesem Kapitel betrachtet.

2.3 MPC für n Spieler: die Grundidee

Im Rest dieses Kapitels betrachten wir die Berechnung einer allgemeinen Funktion durch n Spieler $P_1, \ldots, P_n$. Die Funktion ist als Schaltung, bestehend aus Additions- und Multiplikationsgattern, über einem endlichen Körper $GF(q)$ gegeben.[3] Für den binären Fall ($q = 2$) entspricht die Addition dem XOR-Gatter und die Multiplikation dem AND-Gatter. Viele Protokolle erfordern einen grösseren Körper ($q > n$), aber die Wahl des Körpers ist für die Anwendung nur von beschränkter Wichtigkeit. Eine Schaltung, die in einem gegebenen Körper spezifiziert ist, kann in eine äquivalente Schaltung über einem anderen Körper transformiert werden. Die resultierende Schaltung kann zwar grösser sein, aber nur um einen polynomialen Faktor (in $\log(q)$).

Die Berechnung der Schaltung erfolgt Gatter um Gatter. Als *Invariante* der Berechnung wird sichergestellt, dass jedes Zwischenresultat (d.h. jeder Input und Output eines Gatters) unter den Spielern mittels Secret-Sharing (siehe Abschnitt 2.4) aufgeteilt ist, so dass die ehrlichen Spieler den Wert berechnen könnten, jede potentielle Betrügerkoalition aber keine Information über den aufgeteilten Wert besitzt.

Ein Protokoll besteht üblicherweise aus den folgenden drei Phasen:

- **Input.** Jeder Input zur Berechnung wird vom entsprechenden Spieler mittels Secret-Sharing (im Fall von aktiven Gegnern mittels Verifiable Secret-Sharing) unter allen Spielern aufgeteilt.
- **Berechnung.** Jedes Gatter wird unter Erhaltung der Invarianten berechnet. Additionen (und allgemein jede lineare Berechnung) werden in der Regel trivial sein und keine Kommunikation erfordern.

[2] Es gibt auch eine informationstheoretisch sichere Reduktion von 1-out-of-k String-OT auf 1-out-of-2 Bit-OT.

[3] Für jeden Körper gilt, dass sich jede Funktion mittels Addition und Multiplikation schreiben lässt, d.h. dass Addition und Multiplikation für die Berechnung vollständig sind. Diese Tatsache ist für den binären Fall aus der Digitaltechnik bekannt.

Die Multiplikation wird die nicht-triviale Operation sein.

- **Output.** Wenn ein Spieler einen berechneten Wert als Output erhalten soll, senden alle Spieler ihre Shares des entsprechenden Wertes an diesen Spieler. Sollen mehrere oder alle Spieler den Output erhalten, so wird diese Öffnungsoperation für jeden dieser Spieler durchgeführt.

In der obigen Beschreibung, wie auch in einem grossen Teil der Literatur zur Multi-Party Computation, sind es die gleichen Spieler, die Input liefern, die Berechnung durchführen, und Output erhalten. Diese drei Funktionen können aber problemlos getrennt werden: es können auch Spieler Input liefern und Output erhalten, die nicht an der eigentlichen Berechnung beteiligt sind. Dies ist zum Beispiel dann sinnvoll, wenn mehrere Entitäten treuhänderisch eine Berechnung für eine Gruppe von Entitäten durchführen. Die Auswertung einer Abstimmung kann so interpretiert werden: mehrere Auswertungsinstanzen führen für die (viel grössere Zahl der) Abstimmenden die Summation der Stimmen treuhänderisch durch.

Es besteht natürlich ein grosser Unterschied, ob aktive oder nur passive Betrüger auftreten. Wir werden zunächst Protokolle betrachten, die sicher sind gegen passive Gegner. Anschliessend werden diese Protokolle erweitert, um Sicherheit gegen aktive Gegner zu gewährleisten.

2.4 Secret-Sharing

Eine der zentralen Ideen der sicheren Multi-Party Computation ist, wie schon früher erwähnt, die Aufteilung eines geheimen Wertes s auf mehrere Spieler, *Secret-Sharing* genannt. In diesem Abschnitt behandeln wir deshalb Secret-Sharing in einem allgemeinen Kontext. Secret Sharing ist in der Kryptographie aber auch von unabhängigem Interesse, z.B. für die sichere redundante Speicherung geheimer Schlüssel.

2.4.1 Definition

Secret-Sharing erlaubt einem Spieler, einen geheimen Wert s (z.B. einen kryptographischen Schlüssel oder eine Safekombination) auf mehrere Personen (resp. Spieler oder Entitäten) $P_1, \ldots, P_n$ aufzuteilen, so dass nur bestimmte *qualifizierte* Teilmengen von Personen den Wert s rekonstruieren können, alle anderen (*nicht qualifizierten*) Teilmengen aber keine Information über s erhalten. Die einem Spieler bekannte Information wird *Share* genannt. Alle hier behandelten Secret-Sharing Schemes sind informationstheoretisch sicher, d.h. die Shares einer nicht qualifizierten Spielermenge sind statistisch unabhängig vom verteilten Wert.

Eine wichtige Klasse von Secret-Sharing Schemes sind sogenannte *Schwellenverfahren*. In einem k-aus-n Schwellenverfahren sind beliebige k Spieler qualifiziert, jede Menge von $k-1$ Spielern ist hingegen nicht qualifiziert. In vielen Fällen genügt ein Schwellenverfahren aber nicht.

Beispiel 2.4. In einer Bank mit 3 Direktoren und 5 Vizedirektoren soll die Safekombination so auf die Direktoren und Vizedirektoren aufgeteilt werden, dass beliebige 2 Direktoren oder jeder Direktor mit beliebigen 2 Vizedirektoren die Kombination rekonstruieren können.

Der Spieler, der die Shares verteilt, wird oft *Dealer* genannt. Wir werden ohne Verlust an Allgemeinheit annehmen, dass s aus einer (additiven) endlichen Gruppe stammt. In der Regel brauchen wir sogar die stärkere algebraische Struktur eines Körpers.

Definition 2.1. Sei $\mathcal{P}$ die Menge der Spieler und $2^{\mathcal{P}}$ die Potenzmenge, d.h. die Menge der Teilmengen von $\mathcal{P}$. Eine *Zugriffsstruktur* Γ ist eine monotone Menge von Teilmengen von $\mathcal{P}$, also

$$\Gamma \subseteq 2^{\mathcal{P}} \text{ mit } \mathcal{M} \in \Gamma \wedge \mathcal{M} \subseteq \mathcal{M}' \subseteq \mathcal{P} \Rightarrow \mathcal{M}' \in \Gamma.$$

Oft wird Γ nur durch die minimalen Teilmengen spezifiziert.

Definition 2.2. Ein *Secret-Sharing Scheme* für $\mathcal{P}$ und Γ besteht aus einem Paar von effizienten Protokollen:

- SHARE für das Verteilen eines Wertes s. SHARE ist probabilistisch und verwendet geheime Kanäle für die Verteilung der Shares.

- RECONSTRUCT für das (spätere) Rekonstruieren von s. Dieses Protokoll nimmt als Input die Beschreibung einer qualifizierten Menge $M \in \Gamma$ sowie die Shares jedes Spielers in dieser Menge, und berechnet daraus s.[4]

Jede Menge $\mathcal{M} \in \Gamma$ von Spielern ist qualifiziert und kann mittels RECONSTRUCT den Wert s rekonstruieren, und jede Menge $\widetilde{\mathcal{M}} \notin \Gamma$ ist nicht qualifiziert, d.h. deren Shares sind gemeinsam statistisch unabhängig von s.

Es wird später behandelt, wie beim Secret-Sharing unehrliche Spieler oder sogar ein unehrlicher Dealer toleriert werden können.

Ein Secret-Sharing Scheme heisst *perfekt*, wenn die Rekonstruktion von s immer funktioniert (d.h., es gibt keine noch so kleine Fehlerwahrscheinlichkeit). Ein Scheme heisst *ideal*, wenn der Share jedes Spielers aus dem gleichen Bereich wie der zu verteilende Wert stammt. Die Leserin kann sich selbst überzeugen, dass dies optimal ist, d.h. dass kürzere Shares nicht möglich sind.

2.4.2 Realisierung allgemeiner Zugriffsstrukturen

Die wohl einfachste nicht völlig triviale Zugriffsstruktur wird in folgendem Beispiel behandelt und entspricht dem in Beispiel 2.3 implizit verwendeten Secret-Sharing Scheme.

Beispiel 2.5. Sei $\mathcal{P} = \{P_1, \ldots, P_n\}$ und $\Gamma = \{\mathcal{P}\}$, d.h. nur alle Spieler gemeinsam können das Geheimnis rekonstruieren. Das Geheimnis s stammt aus der endlichen Gruppe G. Der Dealer generiert n zufällige, uniform gewählte Elemente $r_1, \ldots, r_n$ in G, wobei $s = r_1 + \ldots + r_n$ erfüllt sein muss, und sendet r_i als Share an P_i. Es ist offensichtlich, dass dies ein Secret-Sharing Scheme für die Zugriffsstruktur $\Gamma = \big\{\{P_1, \ldots, P_n\}\big\}$ ist.

Dieses Verfahren kann verwendet werden, um Secret-Sharing für eine allgemeine Zugriffsstruktur zu realisieren: Der Dealer verwendet für jede einzelne minimale Teilmenge in Γ ein unabhängiges Secret-Sharing Scheme gemäss obigem Beispiel. Dies ist bei grossen Mengen Γ natürlich sehr ineffizient, da jede Person P_i so viele Shares erhält wie die Anzahl minimaler Mengen in Γ, in denen P_i vorkommt.

Beispiel 2.6. Sei $\mathcal{P} = \{P_1, \ldots, P_4\}$ und $\Gamma = \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3, P_4\}\}$. Der Dealer generiert 4 zufällige und unabhängige Elemente $r_1, \ldots, r_4$ aus G und sendet P_1 die Shares r_1 und r_2, P_2 die Shares $s - r_1$ und r_3, P_3 die Shares $s - r_2$ und r_4, und P_4 den Share $s - r_3 - r_4$.

Eine weitere Konstruktion von Secret-Sharing Scheme für allgemeine Zugriffsstrukturen wird in den Übungen diskutiert.

2.4.3 Lineare Secret-Sharing Schemes

Ein Secret-Sharing Scheme heisst *linear* (über einen bestimmten Körper), falls sich die Shares $s_1, \ldots, s_n$ der Spieler als lineare Funktion des Secrets s und von zufälligen Werten $r_1, \ldots, r_m$ berechnen lassen. Das heisst, es gibt eine (konstante) $n \times (m+1)$ Matrix A, so dass

$$\begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix} = \begin{bmatrix} A_{10} & A_{11} & \cdots & A_{1m} \\ \vdots & & & \vdots \\ A_{n0} & A_{n1} & \cdots & A_{nm} \end{bmatrix} \begin{bmatrix} s \\ r_1 \\ \vdots \\ r_m \end{bmatrix}.$$

Lineare Secret-Sharings Schemes haben die wichtige Eigenschaft, dass man aus einem Sharing von a und einem Sharing von b ein Sharing von $a + b$ berechnen kann (indem die Spieler ihre Shares addieren). Ganz allgemein kann eine beliebige lineare Funktion auf verteilte Werte angewendet werden, indem jeder Spieler diese Funktion auf seine Shares anwendet.

[4] In der Regel sendet einfach jeder Spieler seinen Share an alle anderen Spieler und jeder berechnet s lokal.

2.4.4 Shamir's Schwellenverfahren

In einem k-aus-n Schwellenverfahren können beliebige k Spieler das Geheimnis rekonstruieren, jede Menge von $k-1$ Spielern erhält aber keine Information über das Geheimnis s:

$$\Gamma = \{\mathcal{M} \subseteq \mathcal{P} : |\mathcal{M}| \geq k\}.$$

Das Verfahren des vorherigen Abschnitts wäre für diese Zugriffsstruktur völlig ineffizient: jedem Teilnehmer würden $\binom{n}{k}\frac{k}{n}$ Teilshares zugewiesen. Shamir [Sha79] schlug folgende viel elegantere Realisierung vor, die ideal ist. Alle Variablen sind Elemente eines endlichen Körpers $GF(q)$. Jeder Person P_i ist fix ein allgemein bekanntes Element $\alpha_i \neq 0$ aus $GF(q)$ zugewiesen. Die α_i's müssen verschieden sein, deshalb muss $q > n$ gelten. Der Dealer wählt $k-1$ zufällige Werte $a_1, \ldots, a_{k-1}$ aus $GF(q)$ und verwendet das Polynom

$$a(x) = s + a_1 x + \cdots + a_{k-1} x^{k-1}$$

mit dem konstanten Koeffizienten s zur Berechnung der Shares. Spieler P_i erhält den Share $a(\alpha_i)$. Beliebige k Shares erlauben, das Polynom zu interpolieren und damit s zu berechnen (z.B. mit Lagrange-Interpolation).

Beliebige $k-1$ Shares geben keine Information über s. Dies kann wie folgt gezeigt werden. Das Geheimnis s kann selbst als Share (für den Wert $x = 0$) angesehen werden. Für die gegebenen $k-1$ Shares von Spielern ist jede Wahl von s ein kompatibler Wert, der das Polynom eindeutig bestimmt. Jeder Wert ist also möglich, und die Wahrscheinlichkeitsverteilung über s ist die gleiche vor und nach Kenntnis der $k-1$ Shares.

Shamir's Secret-Sharing Scheme ist linear. Die entsprechende Sharing-Matrix lautet:

$$A = \begin{bmatrix} \alpha_1^0 & \alpha_1^1 & \cdots & \alpha_1^{k-1} \\ \alpha_2^0 & \alpha_2^1 & \cdots & \alpha_2^{k-1} \\ \vdots & & & \vdots \\ \alpha_n^0 & \alpha_n^1 & \cdots & \alpha_n^{k-1} \end{bmatrix}$$

2.5 Informationstheoretisch sichere MPC mit passiven Gegnern

In diesem Abschnitt beweisen wir das folgende Theorem von Ben-Or, Goldwasser und Wigderson [BGW88] und (unabhängig) Chaum, Crépeau und Damgård [CCD88]:

Theorem 2.1. *Eine Menge von n Spielern kann genau dann jede Funktion informationstheoretisch sicher gegen t passive Gegner berechnen, wenn $t < n/2$.*

Das hier behandelte Protokoll basiert auf dem Protokoll von [BGW88], enthält aber eine Reihe von Verbesserungen und Vereinfachungen. Es bildet auch die Grundlage für die später behandelten Protokolle, die sicher gegen aktive Gegner sind, sowohl für das kryptographische wie auch für das informationstheoretische Szenario.

Wie schon früher erwähnt gehen wir im Folgenden davon aus, dass die Spieler durch paarweise sichere Kanäle verbunden sind. Im informationstheoretischen Modell ist dies eine unvermeidbare Annahme.

2.5.1 Unmöglichkeit bei $n/2$ oder mehr passiven Gegnern

Bevor wir das Protokoll beschreiben, zeigen wir, dass die Grenze von $t < n/2$ passiven Gegnern optimal ist, wenn beliebige Funktionen berechnet werden sollen.[5] Hierzu beweisen wir zuerst, dass gewisse Funktionen nicht sicher berechnet werden können mit $n = 2$ Spielern und $t = 1$ Gegnern, und reduzieren anschliessend den allgemeinen Fall mit $t \geq n/2$ auf den Basisfall.

[5] Früher sahen wir, dass lineare Funktionen bei bis zu $n-1$ passiven Gegnern informationstheoretisch sicher berechnet werden können (siehe Beispiel 2.3). Lineare Funktionen sind in Anwendungen aber oft nicht von grossem Interesse.

Wir beweisen, dass zwei Spieler Alice und Bob nicht das AND ihrer Inputs x_A und x_B berechnen können, ohne dass entweder Alice oder Bob unerlaubte Information über den Input des anderen erhält.[6] Wir beweisen die Unmöglichkeit für $n = 2$ über Widerspruch: Nehmen wir an, es gibt ein Protokoll zur sicheren Berechnung des ANDs, definiert durch die beiden (interaktiven) Programme π_A und π_B. Diese Programme können probabilistisch sein; dies modelieren wir, indem wir als zusätzlichen Input einen zufälligen Bitstring r_A bzw. r_B erlauben. Nun durchlaufen Alice und Bob das Protokoll und erhalten beide das Transkript T (die Menge aller übertragenen Nachrichten), welches natürlich von x_A, x_B, r_A und r_B abhängt. Wegen Privacy muss gelten, dass T keine Information über x_A gibt, falls $x_B = 0$ ist, und wegen Correctness muss gelten, dass T volle Information über x_A gibt, falls $x_B = 1$. Alice kann nun das Transkript analysieren und prüfen, ob es Information über x_A enthält. Hierzu ermittelt sie (mittels durchprobieren), für wieviele Werte von r'_A mit $x'_A = 0$ das beobachtete Transkript T erzeugt worden wäre, und für wieviele Werte von r'_A mit $x'_A = 1$ das beobachtete Transkript erzeugt worden wäre.[7] Sind die beiden Werte gleich (bzw. die Differenz im Verhältnis zum grösseren Wert vernachlässigbar), dann beinhaltet das Transkript (fast) keine Information über x_A, und also muss Bob's Bit $x_B = 0$ sein. Sind die beiden Werte nicht-vernachlässigbar verschieden, dann beinhaltet das Transkript Information über x_A, und Bob's Bit muss $x_B = 1$ sein. Somit kann Alice durch Analyse des Transkripts das Bit von Bob bestimmen.

Falls für alle Berechnungen von n Spielern $t \geq n/2$ passive Gegner toleriert werden könnten, so gibt es zwei Mengen $\mathcal{M}_1$ und $\mathcal{M}_2$ von tolerierbaren Spielern, die sich zur ganzen Spielermenge ergänzen.[8] Eine mögliche Funktion ist, dass je ein Spieler in $\mathcal{M}_1$ und $\mathcal{M}_2$ ein Inputbit hat, alle anderen Spieler keinen Input haben, und dass die AND-Funktion der beiden Bits berechnet werden soll. Falls ein solches Protokoll existiert, könnte es direkt als Protokoll für zwei Spieler Alice und Bob verwendet werden: Alice würde alle Spieler in $\mathcal{M}_1$ und Bob alle Spieler in $\mathcal{M}_2$ simulieren. Gemäss obiger Überlegung existiert aber kein solches Protokoll, d.h. es entsteht ein Widerspruch, wodurch die eine Richtung von Theorem 2.1 bewiesen ist.

2.5.2 Das Protokoll

Im Folgenden sei $t < n/2$ die maximale Anzahl passiver Gegner, d.h. beliebige t Spieler sollen durch Zusammenbringen ihres gesamten Wissens keine Information erhalten, die nicht auch direkt durch ihre eigenen Inputs und den Output bestimmt ist. Die Funktion sei als arithmetische Schaltung über einem endlichen Körper $GF(q)$ mit $q > n$, bestehend aus Additions- und Multiplikationsgattern, gegeben. Jede digitale (binäre, d.h. über $GF(2)$ definierte) Schaltung kann relativ einfach in eine entsprechende Schaltung über einem grösseren Körper umgeformt werden.

Das Protokoll besteht aus den in Abschnitt 2.3 beschriebenen Phasen: Zuerst werden die Inputs mittels $(t+1)$-aus-n Secret-Sharing verteilt, dann wird die Berechnung Gatter für Gatter durchgeführt, und am Schluss wird der Output gemeinsam rekonstruiert. Die Grundidee des Protokolls ist, dass alle Inputs, Zwischenresultate und Outputs gemäss einem $(t+1)$-aus-n Secret-Sharing Scheme zwischen den Spielern aufgeteilt sind.

Die Protokolle SHARE und RECONSTRUCT des $(t+1)$-aus-n Secret-Sharing Scheme wurden bereits diskutiert.[9] Wir müssen also nur noch zeigen, wie Additions- und Multiplikationsgatter evaluiert werden unter Einhaltung der Invarianten (d.h. des $(t+1)$-aus-n Secret-Sharings): Jeder neue Wert soll so aufgeteilt sein, dass keine t Spieler irgendwelche Information über den Wert erhalten, die sie nicht schon vorher besassen, und dass auch bei einer allfälligen Rekonstruktion (z.B. wenn der Wert ein Outputwert ist) keine zusätzliche Information freigegeben wird.

Da das Secret-Sharing Scheme linear ist, kann die Summe (über $GF(q)$) zweier verteilter Werte direkt

[6] Ein Spieler kann bei der AND-Funktion natürlich immer den Input 1 wählen und dadurch als Resultat der Berechnung das Bit des anderen Spielers erhalten. Dies ist aber ein grundsätzliches Problem der Spezifikation, da jeder Spieler ja die volle Wahlfreiheit für den Input hat, und kann nicht verhindert werden.

[7] Alice kann dies prüfen, ohne x_B und r_B zu kennen, weil das Programm π_B deterministisch ist. Solange Alice gleiche Nachrichten an Bob sendet, antwortet Bob auch gleich.

[8] Dieser Beweis zeigt ganz allgemein, dass sich keine zwei potentiellen passiven Gegnermengen zur ganzen Spielermenge ergänzen dürfen. Die Spezifikation der Gegnermengen durch eine Schwelle ist ein Spezialfall. Im allgemeinen kann der Gegner durch eine sogenannte Gegnerstruktur (die Menge der potentiell korrumpierten Spielermengen) charakterisiert werden.

[9] Die Rekonstruktion eines Wertes kann sowohl für einen, für mehrere, oder für alle Spieler durchgeführt werden. Für jeden Spieler muss das Protokoll wiederholt werden.

durch die Addition der entsprechenden Shares berechnet werden, was keine Kommunikation erfordert. Jeder Spieler speichert als Share der Summe die Summe der Shares der Summanden. Da keine Kommunikation stattfindet, wird dadurch die Privacy sicher nicht verletzt (niemand erfährt etwas Neues, was er nicht schon vorher wusste). Dies gilt auch dann, wenn die Summe rekonstruiert wird. Die Korrektheit dieser Additions-Operation ist offensichtlich.

Die gleiche Überlegung zeigt, dass jede lineare Funktion

$$y := f(x_1, \ldots, x_d) = \sum_{j=1}^{d} v_j x_j$$

(über $GF(q)$) von mehreren verteilten Werten $x_1, \ldots, x_d$ für beliebige Koeffizienten $v_1, \ldots, v_d$ berechnet werden kann, indem jeder Spieler diese lineare Funktion auf den entsprechenden Shares berechnet. Das heisst, Spieler P_i berechnet den Share y_i des Resultates y gemäss

$$y_i = \sum_{j=1}^{d} v_j x_{ji},$$

wobei x_{ji} der Share von Spieler P_i des Wertes x_j ist.

Der schwierige Fall ist die Multiplikation zweier verteilter Werte a und b. Das Resultat sei $c = ab$. Jeder Spieler P_i besitzt zu Beginn dieses Multiplikationsprotokolls die Shares a_i und b_i und soll am Ende einen Share c_i des Produktes besitzen. Das Sharing $c_1, \ldots, c_n$ von c soll aus Sicht von beliebigen t Spielern völlig zufällig sein, als ob ein Orakel die Shares aus c durch ein unabhängiges zufälliges Sharing generiert und verteilt hätte.

Das zur Addition analoge Vorgehen wäre, dass jeder Spieler P_i seine Shares a_i und b_i multipliziert, resultierend in $d_i = a_i b_i$. (Hier verwenden wir absichtlich d_i statt c_i.) Dabei treten aber die folgenden zwei Probleme auf:

- Der Grad des Polynoms, auf dem die entsprechenden Werte $a_1 b_1, \ldots, a_n b_n$ liegen, verdoppelt sich auf $2t$. Dies ist zwar noch kleiner als n und die Interpolation von c aus den n Werten $a_1 b_1, \ldots, a_n b_n$ wäre noch möglich, aber bei mehreren Multiplikationen würde der Grad rasch grösser als n werden.
- Ausserdem ist das Produkt-Polynom nicht zufällig (insbesondere ist es reduzibel), was die Privacy des Protokolls verletzt (siehe Übung).

Aus diesen zwei Gründen muss für die Multiplikation ein anderes Protokoll verwendet werden. Die zentrale Beobachtung ist, dass die Interpolation eines Polynoms vom Grad kleiner als n aus n Polynomwerten eine lineare Operation ist (Formel von Lagrange):

$$f(x) = \sum_{i=1}^{n} \prod_{\substack{k=1 \\ k \neq i}}^{n} \frac{x - \alpha_k}{\alpha_i - \alpha_k} \, s_i.$$

Da die α_i-s konstant sind, lässt sich das Secret $s = f(0)$ berechnen als gewichtete Summe $s = w_1 s_1 + \ldots + w_n s_n$ der Shares $s_1, \ldots, s_n$ mit geeigneten (konstanten!) Gewichten $w_1, \ldots, w_n$. Das Produkt c kann also berechnet werden mit

$$c = \sum_{i=1}^{n} w_i d_i, \quad \text{wobei } w_i = \prod_{\substack{k=1 \\ k \neq i}}^{n} \frac{\alpha_k}{\alpha_k - \alpha_i}.$$

Die Situation ist also wie folgt: Jeder Spieler P_i besitzt einen geheimen Input s_i, und die Spieler wollen gemeinsam eine fixierte, bekannte Funktion der Inputs berechnen. Hierzu können die Spieler eine MPC durchführen, wobei das Resultat (c) nicht rekonstruiert wird. Das bedeutet also, dass jeder Spieler seinen Input s_i mittels Secret-Sharing verteilt, die Spieler dann die lineare Funktion von oben verteilt berechnen, und das Sharing des Funktionswertes c speichern. Weil die zu berechnende Funktion linear ist, wird in der MPC kein Multiplikationsprotokoll verwendet.

Multiplikationsprotokoll:

1. Jeder Spieler P_i berechnet das Produkt der Shares, $d_i := a_i b_i$, und benutzt das $(t+1)$-aus-n Secret-Sharing Scheme, um d_i unter allen Spielern zu verteilen. Der Share von Spieler P_j sei d_{ij}.

2. Jeder Spieler P_j berechnet (lokal, ohne Kommunikation) den Wert seines Shares c_j von $c = ab$ als Linearkombination der $d_{1j}, \ldots, d_{nj}$,

$$c_j = \sum_{i=1}^{n} w_i d_{ij}, \text{ wobei } w_i = \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{\alpha_k}{\alpha_k - \alpha_i}.$$

Man beachte, dass die neuen Shares $c_1, \ldots, c_n$ auf einem *zufälligen* Polynom mit Grad $\leq t$ liegen.

2.5.3 Analyse

Da wir hier nur passive Gegner betrachten, ist die Korrektheit der Protokolle einfach zu verifizieren.

Die Geheimhaltung gilt aus folgenden Gründen. Es ist zu zeigen, dass der Gegner (der die Information von t Spielern erhält) bis vor der Rekonstruktion des Resultates keine Information über die anderen Inputs oder irgendwelche anderen Zwischenresultate erhalten kann. Durch die Öffnung des Resultates erhält er nur Information, die er gemäss Spezifikation der zu berechnenden Funktion erhalten soll.

Beim gesamten Protokoll zur gatterweisen Berechnung einer Funktion treten nur drei Operationen auf, und bei jeder Operation können wir uns überzeugen, dass die Geheimhaltung nicht verletzt werden kann.

- Beim *Sharen* eines Wertes durch einen Spieler (eines Inputs oder eines Wertes $a_i b_i$ während der Berechnung) ist die Geheimhaltung gewährleistet, weil der Dealer neue Zufallskoeffizienten verwendet.
- Bei der *lokalen Berechnung* einer linearen Funktion wird nicht kommuniziert und die Geheimhaltung deshalb sicher nicht verletzt.
- Bei der *Rekonstruktion* eines Outputs erhalten lediglich diejenigen Spieler Information (die Kommunikationskanäle sind sicher), die das Resultat erhalten sollen. Die Geheimhaltung kann deshalb nicht verletzt werden.

2.6 Broadcast

Von eminenter Bedeutung für sichere MPC ist die Möglichkeit, einen Wert an alle Spieler zu senden, so dass garantiert ist, dass alle Spieler den gleichen Wert erhalten. In einem Model mit passiven Gegnern kann dies einfach erreicht werden: Der Sender schickt einfach die Nachricht an alle (anderen) Spieler. Im aktiven Model ist die Konsistenz jedoch viel schwieriger zu erreichen: ein unehrlicher Sender könnte verschiedenen Spielern unterschiedliche Werte zuschicken. Wir betrachten im Folgenden ein Protokoll, mit Hilfe dessen garantiert werden kann, dass alle Spieler die gleiche Nachricht erhalten, selbst wenn der Sender beliebig unehrlich ist.

Etwas formaler sagen wir, dass ein Protokoll *Broadcast* erreicht, wenn die folgenden drei Bedingungen erfüllt sind:

CONSISTENCY: Alle ehrlichen Spieler entscheiden sich für denselben Wert, d.h. es herrscht „Agreement" nach der Protokolldurchführung.

VALIDITY: Falls der Sender ehrlich ist, dann entscheiden sich alle ehrlichen Spieler auf denjenigen Wert, welchen der Sender gesendet hat.

TERMINATION: Alle ehrlichen Spieler entscheiden sich irgendwann für einen Wert.

Man beachte, dass es für den Fall eines betrügerischen Senders keine Rolle spielen soll, auf welchen Wert sich die ehrlichen Spieler einigen — solange sie sich alle auf denselben Wert einigen.

Consensus ist das zu Broadcast verwandte Problem, bei dem *jeder* Spieler anfangs einen eigenen Inputwert besitzt (d.h. alle Spieler sind Sender zugleich). Alle Spieler müssen sich auf einen Ausgabewert entscheiden, so dass die bisherigen Consistency- und Terminations-Bedingungen nach wie vor erfüllt sind, wobei sich die Persistency-Bedingung entsprechend der multiplen Inputsituation modifiziert.Formal sagen wir, dass ein Protokoll *Consensus* erreicht, wenn die folgenden drei Bedingungen erfüllt sind:

CONSISTENCY: Alle ehrlichen Spieler P_i entscheiden sich für denselben Wert $y_i = y$.

PERSISTENCY: Falls alle ehrlichen Spieler P_i denselben Inputwert $x_i = x$ besitzen, dann entscheiden sich sämtliche ehrlichen Spieler für Output $y_i = x$.

TERMINATION: Alle ehrlichen Spieler entscheiden sich irgendwann für einen Wert.

Alternativ formuliert bedeutet die Persistency-Bedingung, dass Agreement bezüglich eines bestimmten Inputwerts Agreement bezüglich desselben Outputwerts zur Folge hat, d.h., dass dieser Inputwert persistent bleibt.

Falls $t < n/2$ gilt (d.h. falls die Mehrheit der Spieler ehrlich ist), sind Broadcast und Consensus äquivalent, d.h. jedes beliebige Broadcastprotokoll kann effizient in ein Consensusprotokoll transformiert werden und umgekehrt.

Broadcast ⇒ Consensus: Jeder Spieler broadcastet seinen Inputwert. Jeder Spieler entscheidet sich auf jenen Wert, den er am häufigsten erhalten hat (Majority Voting).

Consensus ⇒ Broadcast: Der Sender verteilt seinen Wert via paarweise Kanäle an alle n Spieler. Anschliessend wird mit Hilfe des Consensus-Protokolls sichergestellt, dass sich alle Spieler für den selben Output-Wert entscheiden.

Im Folgenden betrachten wir das Broadcast-Protokoll von [BGP89], welches für n Spieler und $t < n/3$ informationstheoretisch sicher ist. Das Protokoll wird stufenweise konstruiert: Als erstes beschreiben wir ein Protokoll, welches ein stark abgeschwächte Form von Consensus erreicht, und entwickeln daraus Schritt um Schritt ein vollständiges Consensus-Protokoll (aus welchem mit der obigen Konstruktion ein Broadcast-Protokoll gebaut werden kann).

Des weiteren konzentrieren wir uns auf *binären* Broadcast und Consensus, bei dem ein einziges Bit gesendet wird (d.h. mit Inputbereich $\{0, 1\}$). Ein entsprechendes Protokoll für einen beliebigen endlichen Inputbereich kann z.B. durch mehrfache (parallele) Ausführungen eines beliebigen Bit-Protokolls erreicht werden.

2.6.1 Weak Consensus

Bei Weak Consensus startet jeder Spieler P_i mit einem Inputwert $x_i \in \{0, 1\}$ und entscheidet sich schliesslich auf einen Outputwert $y_i \in \{0, 1, \perp\}$, wobei das Symbol $\perp$ als „ungültig“ interpretiert werden kann. Weak Consensus muss nur eine abgeschwächte Konsistenz-Bedingung erfüllen, nämlich dass sich keine zwei ehrlichen Spieler auf unterschiedliche (gültige) Werte in $\{0, 1\}$ entscheiden dürfen — die Spieler aber mit $y_i = \perp$ den Output für ungültig erklären dürfen.

Formal führen wir eine neue Anforderung ein:

WEAK CONSISTENCY: Falls der Output $y_i \in \{0, 1\}$ für einen ehrlichen Spieler P_i gilt, dann gilt $y_j \in \{y_i, \perp\}$ für alle ehrlichen Spieler P_j.

Ein Protokoll erreicht Weak Consensus, falls es Weak Consistency, Persistency, und Termination erfüllt. Wir geben im Folgenden ein Protokoll für Weak Consensus:

Protocol `WeakConsensus` $(x_1, \ldots, x_n) \to (y_1, \ldots, y_n)$**:**

1. $\forall P_i$: sende x_i an jeden P_j
2. $\forall P_j$: $y_j = \begin{cases} 0 & \text{falls Anzahl erhaltener Nullen} \geq n-t \\ 1 & \text{falls Anzahl erhaltener Einsen} \geq n-t \\ \perp & \text{sonst} \end{cases}$
3. $\forall P_j$: return y_j

Lemma 2.2. *Das Protokoll* `WeakConsensus` *erreicht Persistency, Weak Consistency, und Termination.*

Beweis. PERSISTENCY: Falls alle ehrlichen Spieler denselben Input x besitzen, dann verteilen mindestens $n-t$ Spieler den Wert x. Jeder ehrliche Spieler P_j erhält also (mindestens) $n-t$ mal den Wert x, und folglich (höchstens) $t < n-t$ mal den Wert $1-x$, und entscheidet sich folglich auf $y_i = x$.

WEAK CONSISTENCY: Widerspruchsangenommen, Consistency sei nicht erfüllt, d.h., es gäbe zwei ehrliche Spieler P_i und P_j mit $y_i = 0$ und $y_j = 1$. Dazu müssen mindestens $n-t$ Spieler 0 an P_i und mindestens $n-t$ Spieler 1 an P_j geschickt haben. Also haben mindestens $2(n-t)-n$ Spieler (unehrlicherweise) inkonsistente Werte an P_i und P_j geschickt, was mehr als t ist, ein Widerspruch. Also ist Weak Consistency erfüllt.

TERMINATION: Offensichtlich. □

2.6.2 Graded Consensus

Jeder Spieler P_i startet mit einem Input $x_i \in \{0,1\}$ und entscheidet sich schliesslich auf einen Wert $y_i \in \{0,1\}$. Zudem — als ein zusätzlicher Output des Protokolls — berechnet jeder Spieler P_i einen Grade-Wert $g_i \in \{0,1\}$. Der Grade-Wert kann als Bewertung des erreichten Agreementlevels interpretiert werden, d.h., $g_i = 0$ für „keine Konsistenz" und $g_i = 1$ für „Konsistenz erreicht".

Wir führen die folgenden Anforderungen ein:

GRADED CONSISTENCY: Falls ein ehrlicher Spieler P_i einen Wert $y_i \in \{0,1\}$ mit $g_i = 1$ akzeptiert, dann gilt $y_j = y_i$ für jeden ehrlichen Spieler P_j.

GRADED PERSISTENCY: Falls alle ehrlichen Spieler denselben Inputwert x besitzen, dann entscheiden sich sämtliche ehrlichen Spieler P_i für den Output $(y_i, g_i) = (x, 1)$.

Ein Protokoll erreicht Graded Consensus, falls es Graded Consistency, Graded Persistency und Termination erreicht. Ein solches Protokoll kann wie folgt konstruiert werden:

Protocol `GradedConsensus` $(x_1, \ldots, x_n) \to ((y_{1,1}), \ldots, (y_n, g_n))$**:**

1. $(z_1, \ldots, z_n) =$ WeakConsensus$(x_1, \ldots, x_n)$
2. $\forall P_i$: sende z_i an jeden P_j.
3. $\forall P_j$:
$$y_j = \begin{cases} 0 & \text{falls \#Nullen} \geq \text{\#Einsen} \\ 1 & \text{falls \#Nullen} < \text{\#Einsen} \end{cases}$$
$$g_j = \begin{cases} 1 & \text{falls \#}y_j\text{-er} \geq n-t \\ 0 & \text{sonst} \end{cases}$$
4. $\forall P_j$: return (y_j, g_j)

Lemma 2.3. *Das Protokoll* `GradedConsensus` *erreicht Graded Persistency, Graded Consistency und Termination.*

Beweis. GRADED PERSISTENCY: Alle ehrlichen Spieler P_i beginnen mit dem selben Input $x_i = x$. Die Persistency-Bedingung des anfänglichen `WeakConsensus`-Protokolls garantiert, dass alle ehrlichen Spieler in Schritt 2 wieder den Wert $z_i = x$ verteilen. Folglich gilt für jeden ehrlichen Spieler P_j, dass er (mindestens) $n-t$ mal den Wert x erhält, und (höchstens) t mal den Wert $1-x$, und jeder ehrliche P_i setzt $y_i = x$ und $g_i = 1$.

GRADED CONSISTENCY: Die Spieler P_i und P_j seien ehrlich, und es sei $g_i = 1$. Es gilt zu zeigen, dass dann $y_i = y_j$ gilt. Aus $g_i = 1$ folgt, dass mindestens $n - t$ Spieler P_k den Wert $z_k = y_i$ an P_i gesendet haben. Von diesen $n - t$ Spieler sind mindestens $n - 2t$ ehrlich und haben den gleichen Wert z_k auch an P_j gesendet. Also hat P_j den Wert y_i mindestens $n - 2t$ mal erhalten. Andererseits wissen wir von der Weak Consistency Eigenschaft des Protokolls `WeakConsensus` (in Schritt 1), dass sich die Werte z_k der ehrlichen Spieler P_k nicht widersprechen (also entweder alle in $\{0, \perp\}$ oder alle in $\{1, \perp\}$ sind). Da P_i den Wert y_i mindestens $n - 2t > t$ mal erhalten hat, muss mindestens ein ehrlicher Spieler P_k im `WeakConsensus` auf $z_k = y_i$ entschieden haben; also hat kein ehrlicher Spieler P_k auf $z_k = 1 - y_i$ entschieden. Der Spieler P_j hat also höchstens t mal den Wert $1 - y_i$ erhalten (von den unehrlichen Spielern). Da P_j den Wert y_i mindestens $n - 2t$ mal erhalten hat, den Wert $1 - y_i$ aber höchstens t mal, und weil $n - 2t > t$ gilt, setzt P_j seinen Output $y_j = y_i$.

TERMINATION: Offensichtlich. □

2.6.3 King Consensus

Beim King Consensus übernimmt ein beliebiger Spieler P_k die Rolle des Königs. Falls der König ehrlich ist, dann soll King Consensus alle Anforderungen an Consensus erfüllen. Falls der König unehrlich ist, dann soll zumindest Preagreement erhalten bleiben.

Wir führen die folgende Anforderung ein:

KING CONSISTENCY: Falls der König P_k ehrlich ist, dann entscheiden sich am Ende alle ehrlichen Spieler auf den selben Wert $y \in \{0, 1\}$.

Ein Protokoll erreicht King Consensus, falls es King Consistency, Persistency und Termination erreicht. Ein solches Protokoll kann wie folgt konstruiert werden:

Protocol $\texttt{KingConsensus}_{P_k}(x_1, \ldots, x_n) \to (y_1, \ldots, y_n)$**:**

1. $((z_1, g_1), \ldots, (z_n, g_n)) = \text{GradedConsensus}(x_1, \ldots, x_n)$
2. P_k: sende z_k an jeden P_j.
3. $\forall P_j$: $y_j = \begin{cases} z_j & \text{falls } g_j = 1 \\ z_k & \text{sonst} \end{cases}$
4. $\forall P_j$: return y_j

Lemma 2.4. *Das Protokoll* `KingConsensus` *erreicht King Consensus, Persistency, und Termination.*

Beweis. PERSISTENCY: Falls alle ehrlichen Spieler das Protokoll mit demselben Input $x_i = x$ beginnen, dann gilt nach Schritt 1, dass $z_i = x$ und $g_i = 1$ für alle ehrlichen Spieler P_i, und somit gemäss Schritt 3, dass $y_i = z_i = x$ für alle ehrlichen Spieler P_i.

KING CONSISTENCY: Wir können annehmen, dass der König P_k ehrlich ist. Falls jeder ehrliche Spieler P_j im Schritt 1 den Grade-Wert $g_j = 0$ erhält, dann akzeptiert jeder ehrliche Spieler P_j den Wert z_k vom König als seinen Output, also $y_j = z_k$, und Konsistenz ist erreicht. Falls hingegen ein ehrlicher Spiele P_j den Grade-Wert $g_j = 1$ erhält, dann folgt aus Graded Consistency, dass alle ehrlichen Spieler P_i den gleichen Wert $z_i = z$ erhalten, also insbesondere ist dann $z_j = z_k$, und in Schritt 3 entscheided sich jeder ehrliche Spieler auf das gleiche z_k.

TERMINATION: Offensichtlich. □

2.6.4 Consensus und Broadcast

Schliesslich kann Consensus dadurch erreicht werden, dass `KingConsensus` mit $t+1$ verschiedenen Königen P_k durchgeführt wird, so dass sich garantiert ein ehrlicher Spieler darunter befindet.

Protocol `Consensus` $(x_1, \ldots, x_n) \to (y_1, \ldots, y_n)$**:**

1. for $k := 1$ to $t + 1$ do
   $(x_1, \ldots, x_n) :=$ `KingConsensus`$_{P_k}(x_1, \ldots, x_n)$
   od
2. $\forall P_j$: return $y_j = x_j$

Und Broadcast kann somit durch die generische Konstruktion (siehe Einleitung) erreicht werden:

Protocol `Broadcast` $(x) \to (y_1, \ldots, y_n)$**:**

1. Sender: schicke x an alle P_j
2. $(y_1, \ldots, y_n) =$ `Consensus` $(x_1, \ldots, x_n)$
3. $\forall P_j$ return y_j

Theorem 2.5. *Die Protokolle* `Consensus` *und* `Broadcast` *erreichen Consensus und Broadcast, respektive, falls höchstens $t < n/3$ Spieler unehrlich sind. Kommunikations- und Rechenkomplexität sind polynomial in n.*

Beweis. Wir betrachten nur das `Consensus` Protokoll, die entsprechende Aussage für das `Broadcast` Protokoll folgt dann trivialerweise.

PERSISTENCY: Gilt Agreement bereits vor der ersten Runde von `KingConsensus`, dann bleibt dies aufgrund der Persistency-Bedingung (Lemma 2.4) bis zum Schluss erhalten.

CONSISTENCY: Mindestens einer der $t + 1$ Spieler $P_k \in \{P_1, \ldots, P_{t+1}\}$ ist ehrlich. Somit hat man unmittelbar nach dessen `KingConsensus` Agreement, das erhalten bleibt aufgrund der Persistency-Bedingung.

TERMINATION UND EFFIZIENZ: Code-Inspektion! □

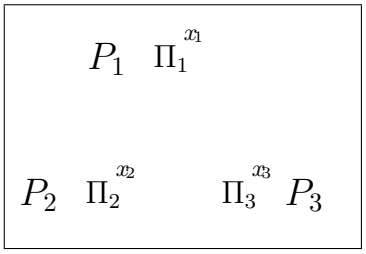
2.6.5 Unmöglichkeit von $t \geq n/3$

In diesem Abschnitt wird nach [PSL80, Lyn96] bewiesen, dass es kein sicheres Broadcastprotokoll geben kann, das $t \geq n/3$ aktive Gegner toleriert — dass das obige Protokoll also optimal im Sinne der Anzahl tolerierter Betrüger ist.

Die Idee des folgenden Beweises beruht darauf, in einem ersten Schritt zu beweisen, dass der Fall $n = 3$ und $t \geq 1$ unmöglich ist. Dieser Spezialfall kann dann durch ein Simulationsargument auf den beliebigen Fall $t \geq n/3$ verallgemeinert werden.

Unmöglichkeit von $n = 3$ und $t = 1$

Nehmen wir zum Widerspruch an, es gäbe ein Consensus-Protokoll für $n = 3$ und $t = 1$. Dieses Consensus-Protokoll wird durch drei interaktive Programme Π_1, Π_2 und Π_3 charakterisiert, wobei Spieler P_i das Programm Π_i verwendet. Jedes Programm Π_i nimmt einen Input x_i und gibt am Ende des Protokoll-Ablaufs einen Output y_i. Ferner bietet jedes Programm Π_i zwei „Kommunikations-Steckdosen“ an, welche mit den Programmen der anderen Spieler verbunden werden müssen:



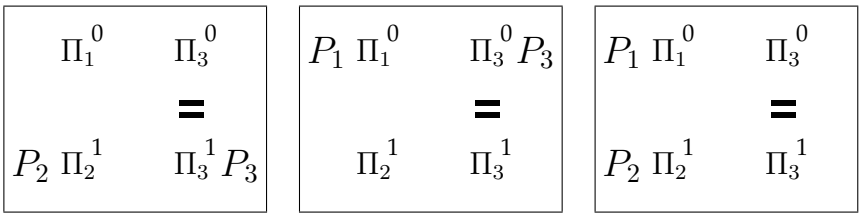
Gemäss Annahme wissen wir, dass wenn wir die Programme richtig „verkabeln“, die drei Bedingungen für Consensus erfüllt werden. Im Folgenden machen wir stark Gebrauch von dieser technischen Sicht, dass die Programme verkabelt werden müssen. Der Gegner, welcher einen Spieler P_i korrumpiert, kann dessen Programm Π_i durch ein beliebiges Programm ersetzen, welches ebenfalls zwei „Steckdosen“ anbietet. Wir betrachten im Folgenden drei Fälle und geben anschliessend je eine Gegnerstrategie an, so dass die drei Fälle identisch aussehen, das Protokoll aber zu verschiedenen Outputs kommen müsste.

Fall 1: P_1 ist unehrlich, P_2 hat Input $x_2 = 1$ und P_3 hat Input $x_3 = 1$.

Fall 2: P_1 hat Input $x_1 = 0$, P_2 in unehrlich und P_3 hat Input $x_3 = 0$.

Fall 3: P_1 hat Input $x_1 = 0$, P_2 hat Input $x_2 = 1$ und P_3 ist unehrlich.

Die Strategie des Gegners ist, dass in allen drei Fällen genau das gleiche Protokoll abläuft. Hierzu verwendet er die folgende Strategie: Im Fall 1 ist P_1 korrumpiert; der Gegner hat also je eine Kommunikationsverbindung zu P_2 und zu P_3. Er startet je eine Instanz der Programme Π_1 und Π_3. Bei Π_1 verbindet er eine Steckdose mit P_2, und die andere Dose mit Π_3. Die andere Steckdose von Π_3 lässt er leer (speist keine Nachrichten ein und ignoriert Nachrichten von Π_3). Ferner gibt der Gegner beiden Programmen den Input 0. Daraus ergibt sich das linke Bild (der Gegner ist grau hinterlegt). Im Fall 2 ist P_2 korrumpiert. Der Gegner startet je eine Instanz von Π_2 und Π_3, beide mit Input 1, und verbindet eine Dose von Π_2 mit P_1, und die andere Dose mit Π_3. Die andere Steckdose von Π_3 lässt er leer. Daraus ergibt sich mittlere Bild. Im Fall 3 — P_3 ist korrumpiert — startet der Gegner zwei Instanzen von Π_3, eine mit Input 0 und eine mit Input 1. Bei der ersten Instanz verbindet er die eine Dose mit P_1 und lässt die andere Dose leer. Bei der zweiten Instanz verbindet er eine Dose mit P_2 und lässt die andere leer. Daraus ergibt sich das rechte Bild.



Fall 1 Fall 2 Fall 3

Bei genauem Hinsehen lässt sich erkennen, dass in allen drei Fällen exakt die gleiche Konstellation von Programmen zusammen kommunizieren, und folglich auch die gleichen Outputs berechnet werden (bzw. die gleichen Zufallsvariablen). Im Fall 1 muss aber $y_2 = 1$ gelten, im Fall 2 muss $y_1 = 0$ gelten, und im Fall 3 muss $y_1 = y_2$ gelten, ein Widerspruch. Egal welcher Output das Protokoll erzeugt, mit Wahrscheinlichkeit $1/3$ erzeugt es einen falschen Output, wenn der Gegner einen zufälligen Spieler P_i korrumpiert und mit der oben beschriebenen Strategie (Fall i) verfährt.

Verallgemeinerung

Im Folgenden zeigen wir, dass die Existenz eines Consensus-Protokolls für n Spieler, welches $t \geq n/3$ Gegner toleriert, die Existenz eines solchen Protokolls für $n = 3$ und $t = 1$ implizieren würde.

Widerspruchsannahme: Es gebe ein sicheres Protokoll für n Spieler das $t \geq n/3$ aktive Betrüger toleriert. Wir betrachten die n Programme $\Pi_1, \ldots, \Pi_n$, welches sicheren Consensus erlauben sollen, und konstruieren daraus ein Consensus-Protokoll für drei Spieler P_1, P_2, P_3. Hierzu lassen wir P_1 die Programme $\Pi_1, \ldots, \Pi_t$ simulieren, P_2 die Programme $\Pi_{t+1} \ldots, \Pi_{2t}$ simulieren, und P_3 die Programme $\Pi_{2t+1}, \ldots, \Pi_n$ simulieren. Jeder Spieler simuliert höchstens t Programme, und da bis zu t Programme unehrlich spielen dürfen, ist es also erlaubt, dass ein Spieler unehrlich ist (und alle seine Programme falsch spielen lässt). Nun startet jeder Spieler P_i alle seine Programme auf seinem Input x_i, wartet das Ende des Protokolls ab, und nimmt als Output y_i den Output eines der Programme. Da das Protokoll unter den Programmen gemäss Annahme Consensus erreicht, erreicht auch des Drei-Spieler-Protokoll unter den Spielern P_1, P_2 und P_3 Consensus.

2.7 Sicherheit gegen aktive Gegner: die Idee

In diesem Abschnitt werden wir schrittweise die Fähigkeiten des (bisher passiven) Gegners erweitern, bis wir einen aktiven Gegner betrachten, der beliebige t Spieler kontrollieren und auf beliebige Art vom Protokoll abweichen lassen kann. Die Betrachtungen dieses Abschnitts gelten sowohl im kryptographischen wie auch im informationstheoretischen Modell. Die einzelnen Fälle werden in späteren Abschnitten behandelt.

Wir gehen hier von einem synchronen Kommunikationsmodell aus, d.h. jede gesendete Nachricht kommt beim Empfänger innerhalb einer gegebenen maximalen Zeitspanne an. Insbesondere können Protokolle in sogenannten Runden durchgeführt werden, da zuverlässig erkannt werden kann, ob ein Wert verzögert wurde oder ob ein Spieler einen Wert nicht gesendet hat.

Wir betrachten der Reihe nach drei Stufen des fehlerhaften Verhaltens.

- **Geheime Information nicht geheim halten.** Dies kann in zwei Formen geschehen: (1) die Koeffizienten im Secret-Sharing nicht zufällig wählen, und (2) Teile der (oder die ganze) geheimen Information an einzelne oder alle anderen Spieler senden.
- **Werte nicht senden**, die gemäss Protokoll an einen, mehrere oder alle Spieler gesendet werden sollen.
- **Falsche Werte senden.** Dies ist die allgemeinste aktive Betrugsart. Die betrügerischen Spieler können koordiniert vorgehen und beliebig vom vorgeschriebenen Protokoll abweichen.

2.7.1 Geheime Information nicht geheim halten

Wir zeigen zunächst, dass das Protokoll des vorherigen Abschnitts (sicher gegen passive Gegner) sicher ist, selbst wenn die $t < n/2$ korrumpierten Spieler die Geheimhaltung zu verletzen versuchen. Die Korrektheit wird dadurch nicht gefährdet, wir müssen also nur die Geheimhaltung betrachten.

Der schlimmste Fall ist, wenn die t Spieler ihre gesamte Information verbreiten. Dies deckt den Fall ab, dass die Spieler die Koeffizienten im Secret-Sharing nicht zufällig wählen. Weil das Protokoll aber garantiert, dass beliebige t Spieler keine Information über die Inputs von anderen Spielern oder über irgendwelche Zwischenresultate erhalten, besitzen die unehrlichen Spieler gar keine relevante Information, die sie verschicken könnten. Selbstverständlich *könnte* ein ehrlicher Spieler mit Hilfe der verbreiteten Information und seiner eigenen Shares unerlaubte Zwischenresultate berechnen; würde er dies tun, dann wäre er aber unehrlich, und müsste zu den korrumpierten Spielern gezählt werden (und wir hätten $t + 1$ korrumpierte Spieler, was per Annahme ausgeschlossen ist).

2.7.2 Werte nicht senden

Wir betrachten nun zusätzlich die nächste Betrugsart, das Nichtsenden von Werten. In diesem Abschnitt gehen wir aber noch davon aus, dass keine falschen Werte verschickt werden. Diese Betrachtung, die für sich genommen vielleicht nicht sehr relevant erscheint, dient als Vorbereitung auf die Behandlung beliebiger aktiver Gegner.[10]

Wir müssen drei Fälle unterscheiden, die wir getrennt betrachten:

(1) Beim Sharen seines Inputs sendet der Dealer gewissen Spielern keinen Share.

(2) Beim Sharen des Produktshares (im Multiplikationsprotokoll) sendet ein Spieler gewissen anderen Spielern keinen Share.

(3) Bei der Rekonstruktion eines Wertes sendet ein Spieler seinen eigenen Share nicht allen anderen Spielern.

[10] Man kann solche Ausfälle aber auch im Sinne der Fehlertoleranz betrachten (z.B. Network Failure), d.h. das Protokoll ist sicher gegen (nicht notwendigerweise absichtlich) abbrechende Spieler. Allerdings reduziert sich bei jedem Ausfall die Toleranzschwelle für die Geheimhaltung um einen Spieler.

Der Fall (3) ist einfach zu lösen, da für die Rekonstruktion eines verteilten Wertes nur $t + 1$ Shares benötigt werden. Ist $n \geq 2t + 1$, so dürfen bis zu t Spieler ausfallen.

In den Fällen (1) und (2) wird zuerst das Problem publik gemacht. Hierzu informiert der Spieler, welcher *keinen* Share erhalten hat, mittels Broadcast alle anderen Spieler. Daraufhin muss der Dealer den vorhin zurückgehaltenen Share nun mittels Broadcast verschicken. Falls er dies tut, ist das Problem erledigt, da ja jetzt der beklagende Spieler seinen Share erhalten hat (und der Umstand, dass der Share nun allen bekannt ist, stört nicht, weil ja sowieso entweder der Dealer oder der beklagende Spieler unehrlich ist). Falls der Dealer sich aber weigert, den zurückgehaltenen Share per Broadcast zu verschicken, können dies alle anderen Spieler sehen, und es ist allgemein bekannt, dass der Dealer unehrlich ist. Im Fall (1) wird nun einfach ein Default-Input für den Dealer angenommen (z.B. 0, verteilt durch lauter 0-Shares). Im Fall (2) ist die Lage etwas schwieriger, weil der Share für das Multiplikationsprotokoll gebraucht wird. Es gibt verschiedene Lösungsansätze:

(i) Das gesamte Protokoll wird wiederholt, aber ohne den Betrüger (sowohl die Anzahl n der Spieler als auch die Anzahl t der maximal anwesenden Betrüger wird um eins verkleinert). Da der Fehler in einem Multiplikationsprotokoll aufgetreten ist, hat noch kein Spieler irgendwelche Information über seinen Output erhalten.

(ii) Die beiden Faktor-Shares des Dealers (und nur diese) werden rekonstruiert[11], und jeder Spieler kann den Produkt-Share berechnen, den der Dealer hätte verteilen müssen. Dann wird das Konstanten-Sharing dieses Produkt-Shares als das Sharing des Dealers verwendet. Der fehlbare Dealer wird jedoch nicht eliminiert.

(iii) Der fehlbare Dealer wird eliminiert und sämtliche relevanten Shares von ihm werden interpoliert (siehe ii). Das Protokoll wird ohne ihn fortgesetzt, wobei jeder Spieler den (die) eliminierten Spieler lokal simuliert. Wenn ein Wert an einen eliminierten Spieler geschickt werden müsste, wird dieser Wert stattdessen an alle Spieler gesendet.

(iv) Der fehlbare Dealer wird eliminiert, und alle Zwischenresultate werden unter den verbleibenden Spielern neu verteilt, diesmal aber mit dem Grad $t' = t - 1$. Dies kann erreicht werden, indem die übrigbleibenden Spieler für jedes Zwischenresultat jeweils ihren Share neu sharen (mit $n' = n - 1$ und $t' = t - 1$), und dann ein Sharing des Wertes als lineare Berechnung der Sharings der Shares berechnen (Lagrange).

Die ersten beiden Lösungen sind die in der Literatur am meisten verwendeten. Die Lösung (i) ist jedoch nicht nur unelegant, sondern funktioniert nur für sichere Funktionsevaluation, und nicht für interaktive Spezifikationen (sogenannte „on-going computations“), in welchen Spieler Output erhalten und später (abhängig von diesen Outputs) neue Inputs liefern. Die Lösung (ii) funktioniert für alle Spezifikationen, ist aber auch nicht gerade elegant (es leuchtet wenig ein, dass ein Spieler, der beim Betrug erwischt wurde, weiterhin im Protokoll stören darf). Die Lösungsansätze (iii) und (iv) sind deutlich eleganter; der letzte Ansatz ist jedoch effizienter als alle anderen Lösungen.

2.7.3 Falsche Werte senden: die theoretische Lösung

Nun betrachten wir den allgemeinsten Fall einer aktiven Betrugsstrategie, bei der ein unehrlicher Spieler beliebige falsche Werte versenden kann. Dieses Problem kann im Prinzip wie folgt durch eine Modifikation des Protokolls des vorherigen Abschnitts gelöst werden. Das Ziel der Modifikation ist es, das Senden eines falschen Wertes zu erkennen und als Nichtsenden des Wertes zu interpretieren.

- Jeder Spieler ist zu jedem ihm bekannten Wert *committet*.

[11] Ein Share eines eliminierten Spielers kann berechnet werden als gewichtete Summe der Shares von beliebigen $t + 1$ anderen Spielern (Lagrange Interpolation). Um einen Share eines eliminierten Spielers zu veröffentlichen, müssen zuerst die ehrlichen Spielern ihren entsprechenden Share unter den Spielern sharen, dann die gewichtete Summe berechnen und das resultierende Sharing rekonstruieren.

- Beim Senden eines Wertes an einen anderen Spieler muss auch das Commitment auf den neuen Spieler übertragen werden. Dies kann zum Beispiel erreicht werden, indem der Sender das Commitment zu Handen vom Empfänger öffnet, und der Empfänger sich zum erhaltenen Wert committet und beweist (z.B. mit einem allgemeinen Zero-Knowledge Beweis), dass der committete Wert gleich dem empfangenen Wert ist.
- Beim Broadcasten eines Wertes muss der Sender das Commitment zuhanden von jedem Spieler öffnen. Dabei muss sichergestellt werden (z.B. mittels Consensus), dass alle Empfänger den gleichen Wert erhalten.
- Bei jedem internen Rechenschritt eines Spielers beweist der Spieler, dass er den Rechenschritt korrekt durchgeführt hat. Mit anderen Worten, er committet sich zum Resultat des Rechenschrittes und beweist (z.B. durch Verwendung eines allgemeinen Zero-Knowledge Beweises, siehe Abschnitt 1.8), dass der committete Wert dem korrekten Resultat der committeten Inputs des Rechenschrittes ist.[12]

Je nach Modell (kryptographische oder informationstheoretische Sicherheit) wird eine andere Art der Commitments verwendet.

- Im kryptographischen Modell wird ein kryptographisches Commitment Verfahren verwendet. Um sich zu einem Wert zu committen (ein Input oder ein Koeffizient eines Secret-Sharings), muss der Spieler das Commitment per Broadcast versenden. Sonst könnte er jedem Spieler ein anderes Commitment senden.
- Im informationstheoretischen Modell wird ein spezielles verteiltes Commitment Verfahren verwendet (siehe Abschnitt 2.9.1).

2.7.4 Falsche Werte senden: die praktische Lösung

Eine Analyse des im letzten Abschnitt skizzierten Protokolls zeigt, dass für das Commitment-Scheme die folgenden Operationen realisiert werden müssen:

1. Ein Spieler kann sich zu einem Wert commiten.
2. Ein Spieler, der zu einem Wert commitet ist, kann dieses Commitment zuhanden von einem bestimmten Empfänger öffnen.
3. Die Spieler können ein Commitment, welches für einen bestimmten Spieler gilt, auf einen anderen Spieler übertragen.
4. Die Spieler können zwei Commitments (für den gleichen Spieler) in ein neues Commitment (für den gleichen Spieler) transformieren, welches als Wert die Summe der Werte der beiden ursprünglichen Commitments enthält.
5. Die Spieler können zwei Commitments (für den gleichen Spieler) in ein neues Commitment (für den gleichen Spieler) transformieren, welches als Wert das Produkt der Werte der beiden ursprünglichen Commitments enthält.

Die ersten beiden Operationen sind inhärenter Bestandteil eines Commitment-Schemes (COMMIT und OPEN). Im MPC-Umfeld muss aber sichergestellt werden, dass sich ein Spieler gegenüber allen anderen Spielern zum gleichen Wert commitet, bzw. zuhanden von allen Spielern den gleichen Wert öffnet.

Die vierte Operation (Addition) kann über eine spezielle Eigenschaft des Commitment-Schemes erreicht werden, die sogenannte Homomorphie-Eigenschaft. Ein Commitment-Scheme heisst *homomorph* (bezüglich Addition), wenn aus zwei Commitments ein Commitment für die Summe lokal berechnet werden kann. Dies erlaubt den Spielern, ohne Interaktion das Commitment zu einer beliebigen Linearkombination bereits

[12] Es gibt (mindestens) zwei Varianten eines solchen Beweises. Entweder wird der Beweis jedem einzelnen Spieler (als Verifier) geliefert, oder er wird einmal öffentlich geliefert, wobei die Zufallswerte des Verifiers durch alle Spieler gemeinsam erzeugt werden und die Antworten des Beweisers mittels Broadcast versandt werden.

committeter Werte zu berechnen. Somit müssen die Spieler für lineare lokale Rechenoperationen keinen interaktiven Beweis durchführen.

Für die dritte Operation (Übertragung von Commitments) postulieren wir ein sogenanntes *Commitment Transfer Protokoll (CTP)*. Das CTP erlaubt, einen committeten Wert geheim von einem Spieler an einen anderen Spieler zu senden, so dass anschliessend auch der Empfänger zu diesem Wert committet ist. Für viele Commitment-Schemes ist dieses Protokoll trivial (der Sender schickt einfach den committeten Wert und die Öffnungsinformation an den Empfänger), es gibt aber Commitment-Schemes, für welche Commitment-Transfer komplizierter ist.

Für die fünfte Operation (Multiplikation von Commitments) postulieren wir ein sogenanntes *Commitment Multiplikations Protokoll (CMP)*. Das CMP erlaubt, dass sich ein Spieler beweisbar zum Produkt von zwei committeten Werten committet.

Lemma 2.6. *Gegeben sei ein homomorphes Commitment-Scheme mit CTP und CMP, sowie ein Broadcast Protokoll. Dann kann ein MPC Protokoll mit entsprechender Sicherheit konstruiert werden. Das MPC Protokoll ist informationstheoretisch (resp. kryptographisch) sicher, wenn das Commitment-Scheme und das Broadcast Protokoll informationstheoretisch (resp. kryptographisch) sicher sind. Das MPC Protokoll toleriert bis zu $t < n/2$ aktive Gegner (höchstens aber soviele, wie das Commitment-Scheme und das Broadcast Protokoll tolerieren).*

Wir beweisen das Lemma konstruktiv durch Angabe des MPC Protokolls. Zuerst konstruieren wir ein Hilfsprotokoll, welches im MPC Protokoll mehrfach verwendet werden wird. Das Hilfsprotokoll erlaubt einem Spieler, der zu einem Wert committet ist, diesen Wert mit einem linearen Secret-Sharing Scheme unter den so Spielern zu sharen, dass anschliessend jeder Spieler zu seinem Share committet ist. Dieses Subprotokoll kann generisch erzeugt werden aus dem Commitment Transfer Protokoll, ist also keine zusätzliche Annahme an das Commitment Scheme.

Commitment Sharing Protokoll:

Ausgangslage: Der Dealer ist zu einem Wert s committet.
Ziel: Jeder Spieler besitzt einen Share von s und ist zu ihm committet.

1. Der Dealer wählt die t für das Secret-Sharing notwendigen Koeffizienten zufällig und committet sich dazu.
2. Unter Verwendung der homomorphen Eigenschaft des Commitment Schemes und der Linearität des Secret-Sharing Schemes berechnen die Spieler (lokal, ohne Kommunikation) Commitments (des Dealers) zu den einzelnen Shares der n Spieler.
3. Die Commitments zu den Shares werden vom Dealer auf die entsprechenden Spieler mit dem Commitment Transfer Protokoll übertragen.

Das Protokoll für Multi-Party Computation ist im Folgenden zusammengefasst.

Multi-Party Computation:

1. Jeder Spieler, der einen Input geben soll, committet sich zum gewählten Wert und verwendet das Commitment Sharing Protokoll, um jeden Spieler zu einem Shamir-Share des Inputs zu committen.
2. Die Schaltung wird Gatter um Gatter evaluiert unter Beibehaltung der Invarianten: Jeder Wert ist mit einem Polynom vom Grad $\leq t$ unter allen Spielern verteilt und jeder Spieler ist zu seinem Share „committet".
   - 2.1. Zwei verteilte Werte werden addiert, indem sich jeder Spieler (unter Ausnutzung der homomorphen Eigenschaft, ohne Kommunikation) zur Summe der beiden Shares committet. Die Multiplikation mit einer Konstanten erfolgt analog, ebenso die Berechnung einer allgemeinen linearen Funktion.
   - 2.2. Die Multiplikation verteilter zweier Werte erfolgt mit dem Multiplikationsprotokoll aus Abschnitt 2.5.2. Dabei muss jeder Spieler seine zwei Shares multiplizieren und sich mittels Commitment Multiplikationsprotokoll zu deren Produkt committen. Anschliessend verwendet er das Commitment Sharing Protokoll, um diesen Wert zu sharen. Dann erfolgt eine lineare Berechnung gemäss Multiplikationsprotokoll (Interpolation), realisiert mittels Schritt 2.1.
3. Ein Outputwert wird für einen Spieler rekonstruiert, indem jeder Spieler das Commitment zu seinem Share gegenüber diesem Spieler öffnet. Mindestens $t+1$ Spieler sind ehrlich und deren Shares genügen. Sollen mehrere Spieler den Outputwert erhalten, so wird dieser Schritt wiederholt.

Weigert sich ein Spieler, einen der obigen Schritte (unter Punkt 1. oder 2.) korrekt auszuführen, so wird entsprechend Abschnitt 2.7.2 verfahren.

2.8 Berechenmässig sichere MPC mit aktiven Gegnern

In diesem Abschnitt beweisen wir das Theorem von Goldreich, Micali und Wigderson [GMW87] von 1987. Der Originalartikel ist schwer verständlich. Das hier präsentierte Protokoll ist konzeptionell einfacher und deutlich effizienter.

Theorem 2.7. *Eine Menge von n Spielern kann genau dann jede Funktion berechenmässig sicher gegen t aktive Gegner berechnen, wenn $t < n/2$.*

Entsprechend den Betrachtungen aus Abschnitt 2.7.4 brauchen wir zum Beweis des Theorems neben einem Broadcast Protokoll lediglich ein homomorphes Commitment Scheme sowie ein zugehöriges Commitment Transfer Protokoll und ein Commitment Multiplikationsprotokoll anzugeben, welche sicher sind gegen $t < n/2$ aktive Gegner. Ein kryptographisches Broadcast-Protokoll für $t < n$ geben wir aus Zeitgründen nicht an; es kann aber relativ einfach konstruiert werden [DS82].

Wir betrachten drei verschiedene homomorphe Commitment Schemes:

1. DL-Commitments: Gegeben sei eine zyklische Gruppe G mit primer Ordnung q und einem Generator g. Das Commitment zu einem Wert $x \in \mathbb{Z}_q$ ist $C(x) = g^x$.
2. Pedersen-Commitments: Gegeben sei eine zyklische Gruppe G mit primer Ordnung q und zwei Generatoren g und h, wobei der diskrete Logarithmus $\mathrm{DL}_g(h)$ nicht bekannt ist. Das Commitment zu einem Wert $x \in \mathbb{Z}_q$ mit Randomness $\alpha \in_R \mathbb{Z}_q$ ist $C(x, \alpha) = g^x h^\alpha$.
3. ElGamal-Commitments: Gegeben sei eine zyklische Gruppe G mit primer Ordnung q und drei Generatoren g, h und γ, wobei die diskreten Logarithmen je zweier Generatoren zueinander nicht bekannt sind. Das Commitment zu einem Wert $x \in \mathbb{Z}_q$ mit Randomness $\alpha \in_R \mathbb{Z}_q$ ist das Paar $C(x, \alpha) = (g^\alpha, \gamma^x h^\alpha)$.

DL-Commitments haben den entscheidenden Nachteil, dass sie nicht semantisch sicher sind: Falls der Gegner den committeten Wert x erraten kann (z.B., weil der Wertebereich klein ist), dann kann er diesen erratenen Wert überprüfen. Wenn z.B. ein Spieler als Input sein Gehalt eingeben muss (und sich zu diesem Input committet), dann kann der Gegner einfach alle Werte zwischen Null und einer Million durchprobieren.

Pedersen-Commitments und ElGamal-Commitments sind semantisch sicher (unter vernünftigen Annahmen). Der Hauptunterschied zwischen den beiden Varianten ist, dass Pedersen-Commitments vom Typ H sind (perfectly hiding, computationally binding), und ElGamal-Commitments vom Typ B (perfectly binding, computationally hiding). Bei der Verwendung von Pedersen-Commitments kann der Gegner also mit sehr grossen Rechenresourcen allenfalls die Correctness (und als Folge davon evtl. auch die Privacy) des Protokolls brechen, während bei Verwendung von ElGamal-Commitments die Correctness des Protokolls informationstheoretisch sicher ist, die Privacy aber nur berechenmässig sicher.

Bei allen drei Commitment-Schemes lassen sich Commitments einfach von einem Spieler auf einen anderen Spieler übertragen (Commitment Transfer Protokoll), indem die Öffnungsinformation (also x und evtl. α) dem empfangenden Spieler gesendet wird. Commitment Multiplikation erfolgt entsprechend der Beschreibung in Abschnitt 2.7.4: Der Dealer erzeugt ein neues Commitment für das Produkt, und beweist in einem interaktiven Beweis, dass er das neue Commitment als Produkt der Werte der beiden anderen Commitments öffnen kann. Für DL-Commitments und für Pedersen-Commitments wurde ein entsprechendes Beweisprotokoll in den Abschnitten 1.9.5 und 1.9.6 gegeben. Für ElGamal-Commitments kann der Beweis analog konstruiert werden.

Diese interaktiven Beweise sind als Two-Party Protokolle ausgelegt: der Verifier muss einen zufälligen Challenge wählen und dem Prover zusenden. In unserem Setting haben wir jedoch n Verifiers (bzw. $n-1$). Entweder muss der Prover das Protokoll mit jedem Verifier einzeln durchspielen (und das Gesamtprotokoll wird akzeptiert, falls alle ausser höchstens t Verifiers ihren Protokolldurchlauf akzeptieren), oder alle Verifiers generieren gemeinsam einen Challenge, simulieren also einen (ehrlichen) Verifier. Hierzu wählt jeder Verifier einen zufälligen Teil-Challenge c_i, committet sich dazu, verteilt c_i (Commitment Sharing Protokoll), und dann werden all diese Sharings rekonstruiert. Der Challenge ist dann die Summe aller Teil-Challenges.

Um die Notwendigkeit der Bedingung $t < n/2$ zu beweisen, betrachten wir zuerst den Fall mit zwei Spielern Alice und Bob und einem Betrüger ($n = 2$, $t = 1$). Um einen Widerspruch herzuleiten nehmen wir an, es existiere ein kryptographisch sicheres Protokoll zur Berechnung des ANDs der beiden Inputs x_A und x_B, wobei beide Spieler den Output $y = x_A \wedge x_B$ erfahren sollen. Wir nehmen das kürzeste Protokoll, welches (gemäss Annahme) sicher sein soll, und betrachten die letzte Nachricht in diesem Protokoll. OBdA nehmen wir an, dass diese Nachricht von Alice an Bob geschickt wurde. Offensichtlich kannte Alice vor dem Absenden dieser Nachricht bereits den Output y. Da das Protokoll nach Weglassen der letzten Nachricht jedoch nicht sicher ist (wir betrachten ja das kürzeste sichere Protokoll), kann Bob vor Erhalt der letzten Nachricht den Output y noch nicht kennen. Also könnte eine unehrliche Alice einfach die letzte Nachricht nicht schicken; damit würde Alice den Output kennen, Bob jedoch nicht, was der Sicherheitsanforderung widerspricht. Der Fall $n > 2$ und $t \geq n/2$ kann gleich wie im passiven Modell auf den Fall $n = 2$ und $t = 1$ reduziert werden.

2.9 Informationstheoretisch sichere MPC mit aktiven Gegnern

In diesem Abschnitt beweisen wir das folgende Theorem von Ben-Or, Goldwasser und Wigderson [BGW88] von 1988.

Theorem 2.8. *Eine Menge von n Spielern kann genau dann jede Funktion informationstheoretisch sicher gegen t aktive Gegner berechnen, wenn $t < n/3$.*

Das hier behandelte Protokoll enthält eine Reihe von Verbesserungen und Vereinfachungen gegenüber dem Protokoll aus [BGW88]. Da Broadcast ein Spezialfall einer Multi-Party Computation ist, folgt die Notwendigkeit der Bedingung $t < n/3$ aus der entsprechenden Bedingung für Broadcast.

Wir wollen wieder das in Abschnitt 2.7.4 beschriebene Protokollmuster verwenden und müssen also ein homomorphes Commitment-Scheme finden, das informationstheoretisch sicher für Beweiser und Verifier

ist. Zudem müssen wir ein Commitment Transfer Protokoll und ein Commitment Multiplikationsprotokoll konstruieren.

Aus früheren Betrachtungen wissen wir, dass Bit Commitments, die für beide Spieler informationstheoretisch sicher sind, nicht existieren können. Der Trick aus diesem Dilemma ist, das Commitment auf alle Spieler zu verteilen und Sicherheit nur bei weniger als $n/3$ aktiven Gegnern zu verlangen. Es spricht ja nichts dagegen, dass beim Committen und beim Öffnen alle Spieler beteiligt sind.

2.9.1 Informationstheoretisch sichere verteilte Commitments

Commitments verstehen wir hier allgemeiner als bisher. Ein Commitment-Verfahren für n Spieler besteht aus zwei Protokollen, COMMIT und OPEN.

- Das Protokoll COMMIT erlaubt einem Dealer D (einer der Spieler), sich zu einem Wert zu verpflichten.
- Das Protokoll OPEN erlaubt ihm später, den Wert wieder zu öffnen.

Beide Protokolle können damit enden, dass die Spieler die Ausführung des Protokolls ablehnen (und damit D als Betrüger entlarven). Wir gehen im Folgenden immer davon aus, dass ein Gegner auftritt, der nicht mehr als $t < n/3$ Spieler aktiv korrumpieren kann. Die Anforderungen an ein Commitment Verfahren sind:

(i) *Konsistenz.* Wenn ein ehrlicher Spieler akzeptiert respektive ablehnt, so tun dies alle ehrlichen Spieler. Dies gilt für das COMMIT und das OPEN-Protokoll. Wenn D ehrlich ist, so akzeptieren die ehrlichen Spieler.

(ii) *Eindeutigkeit.* Wenn das COMMIT-Protokoll akzeptiert wird, dann ist D eindeutig auf einen Wert verpflichtet, d.h. es gibt genau einen Wert, für den D das OPEN-Protokoll bestehen kann.

(iii) *Privacy.* Wenn D ehrlich ist und sich zu einem Wert a verpflichtet, dann erhält der Gegner im COMMIT-Protokoll keine Information über a, die er nicht anderweitig besitzt.

2.9.2 Commitments durch ein Polynom vom Grad $\leq t$

Als verteiltes Commitment Scheme verwenden wir eine Variante von Shamir's Secret-Sharing Scheme (siehe Abschnitt 2.4.4): Um sich zu einem Wert s zu committen, wählt der Dealer ein zufälliges Polynom $g(x)$ vom Grad höchstens t und sendet jedem Spieler P_i den Commit-Share $s_i = g(\alpha_i)$. Ausserdem muss der Dealer beweisen, dass die verschickten Commit-Shares $s_1, \ldots, s_n$ wirklich auf einem Polynom vom Grad t liegen. Um einen committeten Wert zu öffnen, verteilt der Dealer das Polynom $g(x)$ per Broadcast, und alle Spieler prüfen, dass ihr Commit-Share auf dem Polynom liegt. Wir betrachten zuerst das Protokoll zum Öffnen:

Verteilte Commitments: OPEN**-Protokoll:**

Ausgangslage: Ein Dealer D ist mit einem Polynom $g(x)$ zu einem Wert s committet, und jeder (ehrliche) Spieler P_i hält den Commit-Share s_i.

1. D broadcastet das Polynom $g(x)$ (d.h. die $t+1$ Koeffizienten).
2. Jeder Spieler P_i prüft, ob sein gespeicherter Commit-Share auf dem Polynom liegt, d.h., ob $s_i \stackrel{?}{=} g(\alpha_i)$. Falls nicht, klagt er den Dealer mittels Broadcast an.
3. Falls höchstens t Spieler den Dealer anklagen wird das Öffnen akzeptiert als $s = g(0)$. Sonst hat offenbar der Dealer betrogen, und das OPEN-Protokoll ist fehlgeschlagen.

Es ist klar, dass ein ehrlicher Dealer immer das Secret s öffnen kann (es gibt höchstens t unehrliche Spieler, welche ihn anklagen). Wir müssen aber auch zeigen, dass ein unehrlicher Spieler nicht einen anderen Wert als $s = g(0)$ öffnen kann. Dazu betrachten wir den Fall, dass der Dealer ein falsches Polynom $g'(x) \neq g(x)$ broadcastet (welches aber ebenfalls Grad t hat). Dieses Polynom kann höchstens t Commit-Shares identisch mit $g(x)$ haben, das heisst, mindestens $n - t$ Spieler stellen fest, dass ihr Commit-Share nicht auf $g'(x)$ liegt. Von diesen Spieler werden höchstens t (unehrliche) diese Inkonsistenz nicht melden, es werden also mindestens $n-2t$ Spieler den Dealer anklagen. Für $t < n/3$ sind somit mehr als t anklagende Spieler vorhanden, und das OPEN-Protokoll schlägt fehl.

Wir wenden uns nun dem Commitment Protokoll zu:

Verteilte Commitments: COMMIT**-Protokoll:**

1. *Verteilung:* Der Dealer D wählt ein zufälliges Polynom $f(x, y)$ in zwei Variablen vom Grad höchstens t, wobei $f(0, 0) = s$,
$$f(x,y) = \sum_{i=0}^{t} \sum_{j=0}^{t} f_{ij}\ x^i y^j, \quad \text{mit } f_{00} = s,\ \ f_{ij} \in_R GF(q),$$
und sendet jedem Spieler P_i (für $i = 1, \ldots, n$) die Polynome $h_i(x) := f(x, \alpha_i)$ und $k_i(y) := f(\alpha_i, y)$ zu.
2. *Konsistenzchecks:* Für $1 \leq i, j \leq n$: Das Paar (P_i, P_j) prüft, ob $h_i(\alpha_j) = k_j(\alpha_i)$. Dazu sendet P_i den Wert $h_i(\alpha_j)$ an den Spieler P_j, und P_j vergleicht den erhaltenen Wert mit $k_j(\alpha_i)$. Jeder Spieler P_j reklamiert alle Koordinaten (i, j) mit nicht übereinstimmenden Werten per Broadcast. Der Dealer muss solche Reklamationen beantworten, indem er den korrekten Wert (also $f(\alpha_i, \alpha_j)$) per Broadcast verschickt.
3. *Anklagen:* Falls der Dealer in Schritt 2 Werte broadcastet, welche nicht konsistent sind mit den beiden Polynomen $h_i(x)$ und $k_i(y)$ eines Spielers P_i, dann klagt dieser den Dealer per Broadcast an. Der Dealer muss auf solche Anklagen antworten, indem er die beiden Polynome $h_i(x)$ und $k_i(y)$ von P_i per Broadcast verschickt. Falls die gebroadcasteten Polynome inkonsistent sind mit den Polynomen eines anderen Spielers P_j (also $h_i(\alpha_j) \neq k_j(\alpha_i)$ oder $k_i(\alpha_j) \neq h_j(\alpha_i)$), dann klagt auch P_j den Dealer an, und der Dealer muss auch dessen Polynome $h_j(x)$ und $k_j(y)$ mittels Broadcast bekanntmachen. Dieses Prozedere wird solange wiederholt, bis kein Spieler mehr den Dealer anklagt.
4. *Commit-Share bestimmen:* Falls im Schritt 3 mehr als t Anklagen eingegangen sind (oder der Dealer einige der Anklagen nicht beantwortet hat, oder sich die Antworten gegenseitig widersprechen), dann wird der Dealer disqualifiziert. Sonst berechnet jeder Spieler P_i als Output vom Protokoll seinen Commit-Share $s_i = k_i(0)$. Diejenigen Spieler, die den Dealer in Schritt 3 angeklagt haben, berechnen $s_i = k_i(0)$ als Output, wobei $k_i(x)$ das Polynom ist, das der Dealer als Antwort auf die Anklage gebroadcastet hat. Der Dealer D nimmt das Polynom $g(x) := f(x, 0)$ als sein Output.

Um die Sicherheit dieses Protokolls zu analysieren, brauchen wir zuerst ein paar Grundlagen über Polynome in zwei Variablen: Ein Polynom $f(x, y)$ mit Grad t ist eindeutig bestimmt durch die Polynomwerte an den Stellen (α_i, α_j), wobei α_i und α_j aus einer festen Menge von $t+1$ Körperelementen stammen (verallgemeinerte Lagrange-Interpolation. Andererseits geben die Polynomwerte an bis zu t Stellen (also insgesamt t^2 Werte) keine Information über $f(0, 0)$. Etwas formaler: Für eine beliebige Menge $\mathcal{I} \subseteq \{1, \ldots, n\}$ definieren die Polynomwerte an den Stellen (α_i, α_j) mit $i, j \in \mathcal{I}$ das Polynom (und damit auch das Secret $f(0, 0)$) eindeutig falls $|\mathcal{I}| > t$. Falls $|\mathcal{I}| \leq t$ geben die Werte keine Information über das Secret $f(0, 0)$. Details hierzu werden in der Vorlesung gegeben.

Privacy. Die Polynome $h_i(x)$ und $k_i(y)$ von bis zu t (unehrlichen) Spieler geben gemeinsam keine Information über das Secret; jedes Secret ist konsistent mit diesen Polynomen. Später im Protokoll werden immer nur Werte veröffentlicht, welche sowieso ein unehrlicher Spieler schon kennt, was die Privacy nicht weiter tangiert.

Korrektheit. Falls der Dealer ehrlich ist, dann ist die Korrektheit des Protokolls offensichtlich. Es bleibt zu zeigen, dass die Commit-Shares der ehrlichen Spieler auch dann konsistent sind, wenn der Dealer unehrlich ist. Nach dem Schritt 3 sind die Polynome aller ehrlichen Spieler *paarweise* konsistent (wobei die Spieler, welche den Dealer angeklagt haben, die vom Dealer als Antwort gebroadcasteten Polynome verwenden). Wir betrachten das Polynom $f'(x, y)$, welches durch die Polynome $h_i(x)$ und $k_i(y)$ der $t+1$ ehrlichen Spieler mit den kleinsten Indizes definiert ist. Das Polynom $h_j(x)$ von irgendeinem anderen ehrlichen Spieler ist ja konsistent mit allen Polynomen $k_i(y)$ der $t + 1$ ersten Spieler, daraus folgt direkt, dass $h_j(x)$ ebenfalls in $f'(x, y)$ liegt. Analoges gilt für $k_j(y)$ aller anderen ehrlichen Spieler. Dies beweist, dass alle Polynome der ehrlichen Spieler auf einem wohldefinierten Polynom $f'(x, y)$ vom Grad $\leq t$ liegen. Insbesondere liegen die Commit-Shares $k_i(0)$ auf einem Polynom $g(x) = f'(x, 0)$ vom Grad $\leq t$ und definieren somit eindeutig das Secret.

2.9.3 Commitment Transfer Protokoll

Ein Protokoll zum Transferieren eines verteiten Commitments an einen anderen Spieler wird in den Übungen diskutiert.

2.9.4 Commitment Multiplikations Protokoll

Als letzten Schritt müssen wir zeigen, wie ein Spieler D sich zum Produkt $c = ab$ zweier committeter Werte a und b committen kann. Das folgende Protokoll ist vollkommen generisch und funktioniert für ein beliebiges Commitment Scheme (auch für berechenmässige Verfahren), verlangt aber dass $t < n/3$.

Commitment Multiplikations Protokoll:

Ausgangslage: Ein Spieler D ist zu zwei Werten a und b committet.

Ziel: D ist zum Produkt $c = ab$ der beiden Werte committet.

1. D berechnet $c = ab$ und committet sich zu c.
2. D führt für die Werte a und b je das Commitment Sharing Protokoll aus für das $(t + 1)$-aus-n Secret-Sharing Scheme mit einem Polynom vom Grad t, resultierend in Shares a_i und b_i für Spieler P_i, zu denen P_i committet ist.
3. D führt das Commitment Sharing Protokoll aus für den Wert c, wobei er ein Polynom vom Grad $2t$ (statt t) wählt, und zwar so, dass das Polynom gerade das Produkt der beiden Polynomen aus Schritt 2 ist (und somit für $1 \leq i \leq n$ der Share c_i des Spielers P_i gleich $a_i b_i$ ist). P_i ist zu c_i committet.
4. Jeder Spieler prüft, ob $c_i = a_i b_i$. Falls die Bedingung nicht erfüllt ist, so öffnet er die Commitments zu a_i, b_i und c_i und beweist damit, dass D falsch gespielt hat. Erbringt kein Spieler einen solchen Beweis, so wird das Commitment für c aus Schritt 1 akzeptiert, sonst hat sich offenbar D geweigert, sich zum Produkt zu committen, und das Protokoll schlägt fehl.

Die Analyse dieses Protokolls ist eine Übungsaufgabe.

2.A Verifiable Secret-Sharing (VSS)

In der Literatur werden aktiv sichere MPC Protokolle oft basierend auf Verifiable Secret-Sharing Schemes (VSS) konstruiert. Unser MPC Protokoll braucht kein VSS, und deshalb wurde diese Primitive bisher nicht eingeführt. Vollständigkeitshalber zeigen wir im Folgenden, wie VSS basierend auf dem homomorphen Commitment Scheme konstruiert werden kann.

Ein VSS ist ein Paar von zwei Protokollen SHARE und RECONSTRUCT, welche es erlauben, ein Secret unter einer Menge von Spielern so zu sharen (SHARE), dass beliebige t Spieler keine Information über das Secret erhalten, aber alle Spieler zusammen das Secret rekonstruieren können (RECONSTRUCT). In beiden Protokollen dürfen sich bis zu t Spieler beliebig unehrlich verhalten.

Ein VSS ist also ein Secret-Sharing Scheme, welches robust ist gegen t *aktive* Gegner. Alternativ kann ein VSS auch gesehen werden als ein verteiltes Commitment Scheme, in welchem die Commitments selbst dann geöffnet werden können, wenn der Dealer nicht mitspielt (aber genügend andere Spieler).

Das vorher diskutierte COMMIT-Protokoll für das verteilte Commitment Scheme ist in der Tat sogar ein SHARE-Protokoll eines VSS: Es erreicht, dass die Spieler je einen Share erhalten (nämlich $s_i = k_i(0)$), und jeder Spieler zu diesem Share committet ist (nämlich mit dem Polynom $k_i(x)$, wobei Spieler P_j den Wert $k_j(\alpha_i) = h_i(\alpha_j)$ auf diesem Polynom kennt). Zur Rekonstruktion eines so verteilten Wertes muss zuerst jeder Spieler das Commitment seines Shares s_i öffnen (mit dem OPEN-Protokoll), dann wird das Secret aus den korrekten Shares mit Lagrange's Formel interpoliert.

2.B Ein effizienteres Commitment Sharing Protokoll

Aus der obigen Betrachtung folgt auch sofort ein effizienteres Commitment Sharing Protokoll für verteilte Commitments. Diese Protokoll funktioniert jedoch nur für Sharings vom Grad t, kann also im Commitment Multiplikations Protokoll nicht zum Sharen von c (mit Grad $2t$) verwendet werden.

Commitment Sharing Protokoll:

Ausgangslage: Der Dealer D ist mittels eines Polynoms $g(x)$ vom Grad $\leq t$ zu einem Wert s committet.

Ziel: Jeder Spieler besitzt einen korrekten Share von s und ist zu ihm committet.

D führt das VSS-Protokoll durch, wobei er $g(x)$ als Polynom $f(x, 0)$ verwendet. Er wählt also nur noch $t(t + 1)$ (statt $(t + 1)^2 - 1$) zufällige Koeffizienten. Das VSS-Protokoll wird nur insofern modifiziert, als jeder Spieler P_i zusätzlich auch prüft, ob $f(\alpha_i, 0) = g(\alpha_i)$ und, falls dies nicht der Fall ist, D anklagt.

Kapitel 3

Voting

3.1 Einleitung

Voting gehört zu den wohl wichtigsten Anwendungen von verteilten Protokollen. Ein Wahlprotokoll garantiert die Korrektheit des Ergebnisses, während die Stimmen der Wähler geheim bleiben. Ein Wahlprotokoll kann natürlich als Spezialfall einer allgemeinen MPC betrachtet werden. Die zu berechnende Funktion ist die Summe der gültigen Stimmen.

Historisch gesehen sind Wahlprotokolle jedoch nicht aus MPC Protokollen entstanden. Verschiedene MPC Protokolle wurden sogar von Wahlprotokollen inspiriert. Der erste Vorschlag für elektronische Wahlen wurde 1981 von David Chaum publiziert [Cha81] (also ein Jahr früher als Yao's Publikation zu MPC). Chaum's Protokoll ist jedoch sehr ineffizient und in der Praxis nicht relevant. Effiziente Protokolle wurden in den letzten 10 Jahren entwickelt. Das effizienteste publizierte Protokoll ist dasjenige von Cramer, Gennaro und Schoenmakers [CGS97], welches in dieser Vorlesung besprochen wird.

Aus Effizienzgründen wird die effektive Berechnung (Aufaddierung der Stimmen) nicht wie bei MPC unter allen Wählern verteilt; stattdessen wird eine Menge von Autoritäten bemüht, welche sich um die Auszählung kümmern. Damit wird erreicht, dass die Komplexität einer Wahl bloss linear in der Anzahl Wähler steigt (aber typischerweise mindestens quadratisch in der Anzahl Autoritäten).

Die Sicherheitsanforderungen an ein Wahlprotokoll sind analog den Sicherheitsanforderungen an ein MPC Protokoll definiert:

- **Correctness**: Das Wahlergebnis entspricht der Summe aller abgegebenen Stimmen.
- **Privacy**: Die einzelnen Stimmen der Wähler bleiben geheim.
- **Robustness**: Das Protokoll kann nicht abgebrochen werden.

Dabei sind noch diverse Subtilitäten zu beachten. Zum Beispiel darf eine unehrliche Wählerin nicht eine Stimme eines ehrlichen Wählers kopieren (oder gerade die gegenteilige Stimme abgeben). Auch muss definiert sein, was mit ungültigen Stimmen geschehen soll. Formal gesehen muss die Sicherheit eines Wahlprotokolls mit dem Simulationsansatz definiert werden: In der Spezifikation können alle Wahlberechtigten einen Wert (z.B. „0" oder „1") an die Trusted Party senden, welche dann die gültigen Werte addiert und das Resultat allen zusendet. Ein Wahlprotokoll ist sicher, wenn die unehrlichen Entitäten im Protokoll nichts erreichen können, was sie nicht auch in dieser Spezifikation erreichen könnten.

Wir betrachten ein Modell mit N Autoritäten $A_1, \ldots, A_N$ und M Wählern. Wir nehmen an, dass alle Entitäten (Autoritäten und Wähler) über unsichere (aber synchrone) Kanäle miteinander kommunizieren können. Ausserdem wird ein sogenanntes Bulletin-Board (BB) vorausgesetzt. Ein BB hat die Eigenschaft, dass jedermann beliebig lesen und schreiben kann, aber niemand kann löschen. Bulletin-Boards können realisiert werden unter der Annahme, dass mehr als die Hälfte der Autoritäten ehrlich sind; dies wird jedoch in dieser Vorlesung nicht weiter betrachtet.

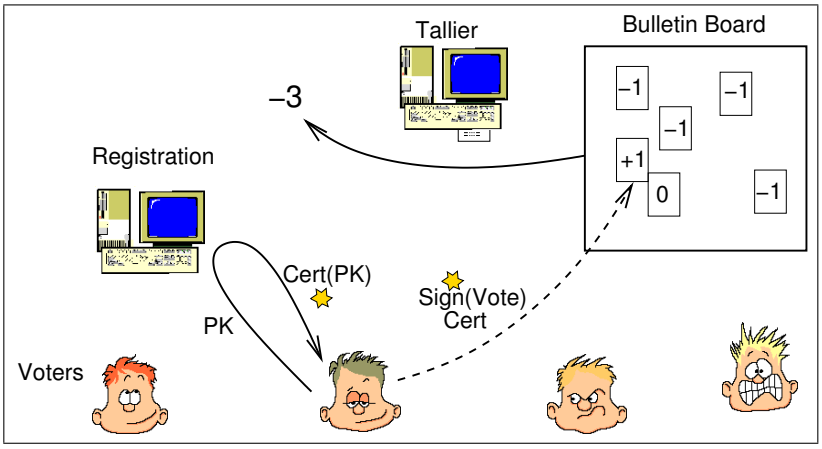
Weiter nehmen wir an, dass eine Public-Key Infrastructure (PKI) aufgesetzt ist. Das heisst, alle beteiligten Entitäten kennen den Public-Key von jeder anderen Entität. Damit können wir Nachrichten verschlüsseln (für die Geheimhaltung) und signieren (für die Authentisierung).

Wir verlangen, dass mehr als die Hälfte der Autoritäten ehrlich ist. Über die Wähler werden keine Annahmen gemacht; beliebig viele von ihnen dürfen sich unehrlich verhalten. Sämtliche betrachteten Protokolle beruhen auf kryptographischen Annahmen. Folglich dürfen unehrliche Spieler „nur“ über beschränkte Computerressourcen verfügen.

3.2 Typen von Wahlprotokollen

In der Literatur wurden drei Typen von Wahlprotokollen vorgeschlagen. Wir zeigen im Folgenden, wie diese Wahlprotokolle aufgebaut sind, und welche Vor- und Nachteile die einzelnen Protokolltypen haben. Die Reihenfolge der Aufzählung ist didaktisch und nicht historisch motiviert.

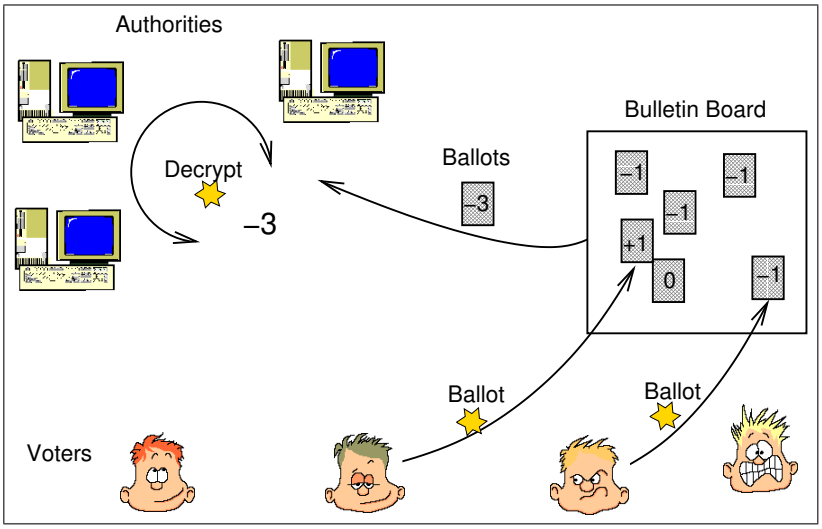
3.2.1 Wahlprotokolle mit anonymen Kanälen



Jede Wählerin registriert sich zuerst bei einer Registration Authority (RA). Hierzu wählt sie zufällig ein temporäres Secret-Key/Public-Key Paar, und lässt sich den PK von der RA *blind* unterschreiben. Die RA zertifiziert für jede Wahlberechtigte nur einen einzigen PK. Die Wählerin signiert dann seine Stimme mit dem SK, und sendet die unterschriebene Stimme (im Klartext), seinen PK und das Zertifikat der RA über einen *anonymen Kanal* auf das Bulletin-Board. Bei diesem Wahlverfahren ist die Auszählung trivial, weil alle Stimmen im Klartext auf dem BB liegen.

Ein Problem des oben skizzierten Protokolls ist, dass Wähler schon vor dem Abgeben der eigenen Stimme Informationen über den aktuellen Stand des Ergebnisses erhalten. Es gibt zwei verschiedene Ansätze, wie dieses Problem gelöst werden kann: (1) Das Bulletin-Board (welches ja sowieso von den Autoritäten simuliert werden muss) lässt keine Lesezugriffe vor dem Wahlende zu. (2) Die Autoritäten generieren ein zufälliges Secret-Key/Public-Key Paar und publizieren den Public-Key zu Beginn der Wahl. Die Wähler können nun ihre Stimme verschlüsselt auf das BB posten. Zum Wahlende publizieren die Autoritäten den Secret-Key, so dass jedermann alle Stimmen entschlüsseln kann.

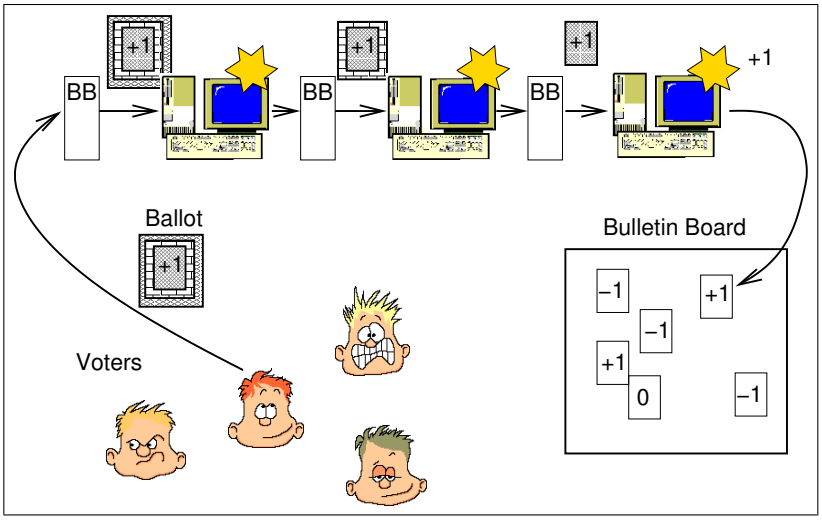
3.2.2 Wahlprotokolle mit homomorpher Verschlüsselung



Eine homomorphe Verschlüsselungsfunktion erlaubt die Addition zweier verschlüsselter Nachrichten, ohne dass der geheime Schlüssel bekannt sein muss. Mit einer solchen Verschlüsselungsfunktion kann man ganz einfach ein Wahlprotokoll zusammenschrauben: Die Autoritäten generieren gemeinsam (in einer MPC) ein SK/PK Paar für diese homomorphe Verschlüsselungsfunktion, und publizieren den PK, behalten aber den SK geshart unter sich. Jede Wählerin verschlüsselt nun ihre Stimme mit dem publizierten PK und schickt sie unterschrieben auf das BB. Am Ende der Wahl addieren die Autoritäten die verschlüsselten Stimmen (unter Ausnutzung des Homomorphismus) und entschlüsseln die Summe (mit MPC).

Bei diesem Wahlverfahren werden die einzelnen Stimmen nie im Klartext bekannt. Es muss also irgendwie sichergestellt werden, dass die Wähler nur gültige Stimmen abgeben können. Dies wird erreicht, indem jeder Wähler zusätzlich zur verschlüsselten Stimme auch einen (zero-knowledge) Gültigkeits-Beweis auf das BB senden muss. Ein effizienter Gültigkeits-Beweis kann für homomorphe Verschlüsselungsfunktionen mittels der Techniken aus Abschnitt 1.9 konstruiert werden.

3.2.3 Wahlprotokolle mit Mixern



Ein Mixer ist ein Server (eine Autorität), welche verschlüsselte Stimmen zufällig permutiert. Das Wahlprotokoll für N Mixer funktioniert so, dass die Wählerin ihre Stimme zuerst mit dem Public-Key des letzten

Mixers A_N verschlüsselt, dann dieses Chiffrat mit dem PK des zweitletzten Mixers A_{N-1} verschlüsselt, und so weiter. Diese mehrfach verschlüsselte Stimme schickt er dann dem ersten Mixer A_1. Dieser nimmt von jeder Wählerin ein Chiffrat entgegen, entschlüsselt alle Chiffrate mit seinem SK (und erhält Chiffrate für A_2), und schickt diese Menge von Chiffraten an A_2 *in zufälliger Reihenfolge*. Der zweite Mixer A_2 macht dasselbe, usw., bis schliesslich der letzte Mixer die Stimmen im Klartext entschlüsseln kann. Diese Stimmen werden dann aufaddiert und publiziert.

Damit die Mixer nicht Stimmen ändern können, muss jeder Mixer beweisen, dass die Menge der ausgegebenen Chiffrate die gleichen Stimmen enthält wie die Menge der erhaltenen Chiffrate (aber einmal weniger verschlüsselt). Diese Beweise sind im Allgemeinen sehr gross und relativ ineffizient.

3.2.4 Vergleich

Die betrachteten Protokolltypen unterscheiden sich sowohl in ihren Anforderungen als auch von den Zielen. Wir listen verschiedene Unterscheidungskriterien auf und beurteilen die Protokollvarianten danach. Man sollte jedoch nicht ausser Acht lassen, dass in der Praxis oft „nicht-kryptographische“ und sogar „nicht-technische“ Kriterien relevant für die Entscheidung für oder gegen ein bestimmtes Protokoll sind.

- *Verifizierbarkeit*: Bei Unstimmigkeiten können Teile der Wahl oder auch die gesamte Wahl „geöffnet“ werden. Wenn zum Beispiel mehr Leute behaupten, sie hätten „ja“ gestimmt, als insgesamt Ja-Stimmen gezählt wurden, dann lassen sich die Stimmen dieser Leute öffnen. Es zeigt sich, dass Protokolle mit homomorpher Verschlüsselung sehr einfach verifizierbar sind (die Zuordnung der verschlüsselten Stimmen zu den Wählern ist bekannt), solche mit Mixern nur aufwendig verifizierbar sind (die in Frage stehenden Stimmen müssen durch die Mixer verfolgt werden), solche mit anonymen Kanälen oft gar nicht verifizierbar sind.
- *Vote-and-go*: Die Wähler können „in einem Rutsch“ wählen, müssen also nicht mehrfach zu verschiedenen Zeiten aktiv sein. Protokolle mit homomorpher Verschlüsselung erlauben vote-and-go, Mix-Net Protokolle zum Teil auch. Bei Protokollen mit anonymen Kanälen müssen die Wähler mindestens in zwei verschiedenen Phasen aktiv sein (Registrierung und Wählen).
- *Effizientes Auszählen*: Das Wahlergebnis kann kurze Zeit nach dem Wahlende publiziert werden. Das bedeutet, dass entweder die Auszählung sehr effizient realisierbar sein muss (bei Protokollen mit anonymen Kanälen), oder aber inkrementell erfolgen kann (bei Protokollen mit homomorpher Verschlüsselung). Bei Mix-Net Verfahren ist die Auszählung aufwendig und nicht inkrementell durchführbar.
- *„Wilde Kandidaten“*: Bei Wahlverfahren mit homomorpher Verschlüsselung können einzig Zahlen (bzw. Gruppenelemente) aufaddiert werden — die abgegebenen Stimmen werden nie sichtbar. Bei den anderen Wahltypen kann auch „wild“ gestimmt werden, zum Beispiel mit Kandidatennamen, oder mit beliebigem Text.

3.3 Gruppenhomomorphismen

Im Weiteren betrachten wir ausschliesslich Wahlverfahren basierend auf homomorpher Verschlüsselung. Wir werden an verschiedenen Stellen Beweise brauchen, dass der Prover ein Urbild zu einem gegebenen Funktionswert kennt, wobei die auszuwertende Funktion homomorph ist. Wir repetieren die nötigen Techniken aus Abschnitt 1.9, und erweitern sie wo immer nötig.

Sei $(G, \oplus)$ eine additive Gruppe und $(H, \otimes)$ eine multiplikative Gruppe, wobei die Subtraktion in G mit $\ominus$ und die Division in H mit $\oslash$ geschrieben wird. Eine Funktion $f : G \to H$ heisst *homomorph*, falls für alle $x_1, x_2 \in G$ gilt, dass $f(x_1 \oplus x_2) = f(x_1) \otimes f(x_2)$. Wir nehmen weiter an, dass entweder $|G| = q$ für eine Primzahl q, oder aber G die Produktgruppe ist von mehreren Gruppen $G_1, G_2, \ldots$ mit $|G_i| = q$ für alle i. Das gleiche soll für H gelten. Daraus folgt, dass für jedes Element x in G gilt dass $qx = 0$, und für jedes Element y in H gilt dass $y^q = 1$. Diese Annahmen sind stärker als diejenigen aus Abschnitt 1.9, dafür werden aber einige Beweise etwas einfacher.

3.3.1 Wissen eines Urbilds von y

Für jede homomorphe Funktion $f : G \to H$ existiert ein effizienter (sicherer) Beweis von Wissen eines Urbilds. Der Prover kann den Verifier überzeugen, dass er einen Input $x \in G$ kennt, so dass $f(x) = y$ für ein bekanntes, vorgegebenes $y \in H$. Dabei soll ein ehrlicher Verifier, der den Challenge c zufällig wählt, keine Information über x erfahren (d.h., das Protokoll ist honest-verifier zero-knowledge).

Prover		**Verifier**
$k \in_R G, t = f(k)$	$\xrightarrow{t}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_q$
$r = k \oplus cx$	$\xrightarrow{r}$	$f(r) \stackrel{?}{=} t \otimes y^c$

Es folgt direkt aus der homomorphen Eigenschaft von f, dass der Beweis eines ehrlichen Beweisers von einem ehrlichen Verifier immer akzeptiert wird. Es bleibt aber zu zeigen, dass ein unehrlicher Beweiser den Verifier nicht (bzw. nur mit sehr kleiner Wahrscheinlichkeit) überzeugen kann, wenn er x nicht kennt (Wissensextraktor), und dass der ehrliche (!) Verifier keine Information über das x des ehrlichen Provers erhält (honest-verifier zero-knowledge Eigenschaft).

Der Wissensextraktor muss aus zwei akzeptierenden Konversationen mit der gleichen ersten Nachricht aber verschiedenem Challenge den Wert x berechnen. Gegeben sind also die beiden Tupel (t, c, r) und (t, c', r') mit $c \neq c'$, wobei sowohl $f(r) = t \otimes y^c$ als auch $f(r') = t \otimes y^{c'}$ erfüllt sind. Durch die homomorphe Eigenschaft von f folgt dass

$$f(r \ominus r') = f(r) \oslash f(r') = y^{c-c'}.$$

Wir berechnen nun zwei Koeffizienten a und b so dass $aq + b(c - c') = 1$ (Extended Euklid), und folgern

$$\begin{aligned} y &= y^{aq+b(c-c')} = \big(y^q\big)^a \otimes \big(y^{c-c'}\big)^b \\ &= 1^a \otimes \big(f(r \ominus r')\big)^b = f\big(b(r \ominus r')\big). \end{aligned}$$

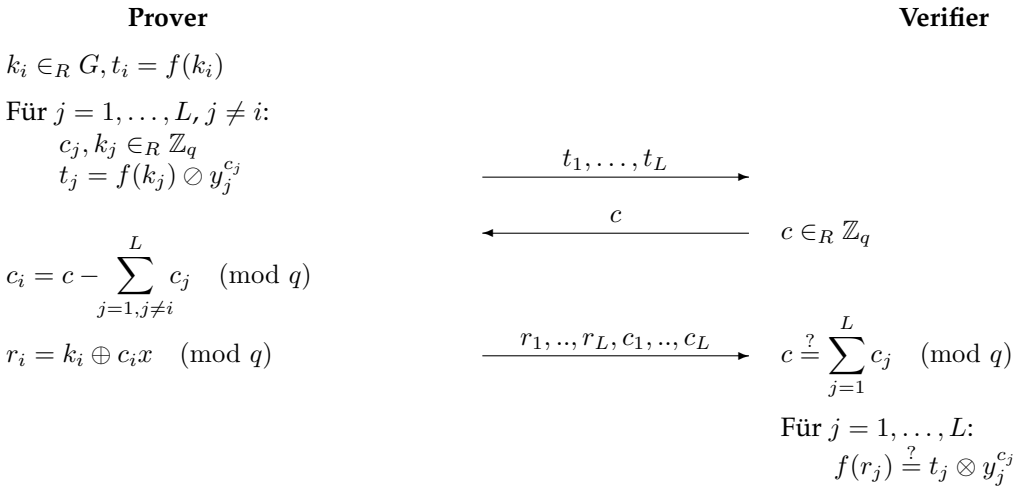
Also ist $x = b(r \ominus r')$ ein Urbild von y.

Der Simulator schliesslich soll eine Konversation mit der richtigen Wahrscheinlichkeitsverteilung generieren. Wir betrachten nur den Fall, dass der Verifier ehrlich ist. Dann können einfach zuerst r und c zufällig (uniform) gewählt werden, und dann wird t entsprechend berechnet.

3.3.2 Wissen eines Urbilds von einem der $y_1, \ldots, y_L$

Wir haben oben gesehen, wie man für ein beliebiges y Wissen eines Urbilds x mit $f(x) = y$ beweisen kann. Wir betrachten nun ein etwas komplizierteres Protokoll zum Beweis von Wissen von mindestens einem Urbild x von einem von mehreren Werten $y_1, \ldots, y_L$, also in einem gewissen Sinn die L-fache OR-Verknüpfung des obigen Beweises. Bekannt sind also L Werte $y_1, \ldots, y_L \in H$, und der Prover soll beweisen, dass er ein $x \in G$ und ein $i \in \{1, \ldots, L\}$ kennt, so dass $f(x) = y_i$, ohne x oder i bekannt zu geben.

Dazu werden L parallele Instanzen des obigen Protokolls durchgeführt (eine Instanz für jedes $y_1, \ldots, y_L$), aber der Verifier stellt nur einen einzigen Challenge c für alle Instanzen, und der Prover darf diesen Challenge aufteilen in L Sub-Challenges $c_1, \ldots, c_L$, welche sich zu c summieren. Diese Idee ermöglicht dem Prover, in allen ausser einer Protokollinstanz den Simulator zu verwenden (und sich damit auf alle c_j mit $j \neq i$ vorzubereiten), und bei Erhalt des Challenges c vom Verifier nur noch das c_i entsprechend zu wählen. Das gesamte Protokoll ist im Folgenden gegeben:



Es ist einfach zu sehen, dass dieser Beweis sicher ist, sofern der Grundbeweis (Abschnitt 3.3.1) sicher ist. Der Wissensextraktor muss aus zwei Konversationen mit gleicher erster Nachricht und verschiedenen Challenges das x finden. Da die beiden Challenges verschieden sind, muss folglich in der dritten Nachricht (mindestens) eines der c_j in den beiden Konversationen verschieden sein. Das bedeutet aber, dass es mindestens eine Protokollinstanz gibt (für den Beweis von Wissen eines Urbilds), in welchem die erste Nachricht gleich ist, aber der Challenge verschieden, und wir können den Wissensextraktor dieser Instanz verwenden, um x zu berechnen. Der Simulator lässt sich einfach bauen, indem zuerst der Challenge c zufällig gewählt wird, dann ein zufälliges Splitting gewählt wird, und für jeden der Sub-Challenges mithilfe des Simulators des Basisprotokolls eine akzeptierende Konversation generiert wird. Diese Konversationen der L Basisprotokolle können nun zu einer Konversation des OR-Protokolls zusammengefasst werden.

3.3.3 Nicht-interaktive Beweise

Schliesslich brauchen wir noch einen Mechanismus, um interaktive (honest-verifier zero-knowledge) Beweise in nicht-interaktive Beweise umzuwandeln. Hierzu wird die sogenannte Fiat-Shamir Heuristik bemüht: Der Challenge des (nicht mehr existierenden) ehrlichen Verifiers wird bestimmt als Output einer Hashfunktion H, angewendet auf die erste Nachricht. Ein nicht-interaktiver Beweis des Protokolls aus Abschnitt 3.3.1 ist also ein Paar (t, r), und die Überprüfung des Beweises erfolgt mit dem Prädikat

$$f(r) \stackrel{?}{=} t \otimes y^{H(t)}.$$

Dieser nicht-interaktive Beweis ist sicher im sogenannten *Random Oracle Model*, also unter der (theoretischen) Annahme, dass sich die Hashfunktion H wie eine zufällig gewählte Funktion verhält und folglich der Challenge im Protokoll zufällig und gleichverteilt ist.

3.4 Homomorphe Verschlüsselung

Ein homomorphes Verschlüsselungsverfahren besitzt die Eigenschaft, dass zwei Chiffrate so kombiniert werden können, dass sich daraus ein Chiffrat der Summe ergibt. Im nächsten Abschnitt betrachten wir eine Variante der ElGamal Verschlüsselungsfunktion, welche diese Eigenschaft hat. Dieses Verschlüsselungssystem hat jedoch den gravierenden Nachteil, dass die Entschlüsselung nur für kleine Werte (Klartexte) effizient ist. Im Kontext von sicheren Wahlen kann diese Einschränkung oft akzeptiert werden, für viele andere Anwendungen (und auch für bestimmte Typen von Wahlen) macht diese Einschränkung das ElGamal System ungeeignet. Kürzlich wurde ein anderes homomorphes Verschlüsselungssystem publiziert, welches diesen Nachteil bei der Entschlüsselung nicht hat. Dieses System [Pai99] ist jedoch mathematisch ziemlich anspruchsvoll und würde den Rahmen dieser Vorlesung sprengen.

Im Kontext von sicheren Wahlen brauchen wir ein Schlüsselgenerierungsprotokoll, welches ein Secret-Key/Public-Key Paar so generiert, dass der Secret-Key unter den Autoritäten geshart ist, und der Public-Key allgemein bekannt wird. Weiter brauchen wir ein Entschlüsselungsprotokoll, welches ein gegebenes Chiffrat entschlüsselt, ohne dass hierzu der Secret-Key rekonstruiert werden muss. Beide diese Protokolle können natürlich mit allgemeinen MPC Protokollen realisiert werden; wir werden aber eine einfachere und effizientere Implementierung betrachten.

3.4.1 Homomorphe Ver- und Entschlüsselungsfunktion

Sei also G eine kommutative Gruppe der Ordnung $|G| = q$, wobei q eine genügend grosse Primzahl ist, und seien g und γ unabhängige Generatoren von G, also $G = \langle g \rangle = \langle \gamma \rangle$. G kann zum Beispiel konstruiert werden als Subgruppe von $\mathbb{Z}_p^*$, wobei p eine grosse Primzahl ist und $q|(p-1)$. Der Secret-Key z ist ein zufälliges Element in $\mathbb{Z}_q$, und der Public-Key ist $Z = g^z$. Die Verschlüsselung eines Wertes v mit Zufälligkeit α ist wie folgt definiert:

$$E_Z : \mathbb{Z}_q \times \mathbb{Z}_q \to G \times G, \quad (v, \alpha) \mapsto (g^\alpha, \gamma^v Z^\alpha).$$

Wir werden meistens einfach $E(v, \alpha)$ schreiben, und nehmen an, dass der Public-Key Z aus dem Kontext gegeben ist. Um ein Chiffrat $e = (x, y)$ zu entschlüsseln, berechnen wir

$$\frac{y}{x^z} = \frac{\gamma^v Z^\alpha}{g^{\alpha z}} = \frac{\gamma^v g^{z\alpha}}{g^{\alpha z}} = \gamma^v,$$

und berechnen davon den diskreten Logarithmus zur Basis γ. Die Berechnung des diskreten Logarithmus ist effizient für genügend kleine v. Für beliebige Werte v kann ein Chiffrat aber nicht (effizient) entschlüsselt werden.

Man kann sich einfach davon überzeugen, dass diese Verschlüsselungsfunktion homomorph ist für die Addition. Um zwei verschlüsselte Klartexte zu addieren, werden einfach die beiden Chiffrate (komponentenweise) multipliziert. Wir schreiben die Gruppenoperation in $G \times G$ (also die komponentenweise Multiplikation) als $\otimes$.

$$\begin{aligned} E(v_1, \alpha_1) \otimes E(v_2, \alpha_2) &= (g^{\alpha_1} \cdot g^{\alpha_2}, \gamma^{v_1} Z^{\alpha_1} \cdot \gamma^{v_2} Z^{\alpha_2}) \\ &= (g^{\alpha_1 + \alpha_2}, \gamma^{v_1 + v_2} Z^{\alpha_1 + \alpha_2}) \\ &= E(v_1 + v_2, \alpha_1 + \alpha_2). \end{aligned}$$

Diese Verschlüsselungsfunktion ist ein Gruppenhomomorphismus gemäss der Definition in Abschnitt 3.3. Insbesondere sind Definitions- und Wertebereich je das Produkt von Gruppen mit je q Elementen (und folglich gilt $q(v, \alpha) = (0, 0)$ für alle v, α, und $e^q = 1$ für alle e).

3.4.2 Sicherheit der Verschlüsselung

Diese Verschlüsselungsfunktion ist semantisch sicher, das heisst, selbst wenn für ein Chiffrat e zwei Klartextkandidaten v_1 und v_2 bekannt sind, kann nicht festgestellt werden, welcher der beiden Kandidaten tatsächlich in e steckt (angenommen, das Diffie-Hellman Unterscheidungsproblem ist schwierig, siehe Übung).

3.4.3 Verteilte Schlüsselgenerierung

Im Setup müssen die Autoritäten ein zufälliges Secret-Key/Public-Key Paar so erzeugen, dass der Secret-Key unter den Autoritäten geshart ist, der Public-Key aber bekannt wird. Als Secret-Sharing Scheme verwenden wir Shamir's Sharing (Abschnitt 2.4.4) mit DL-Commitments, wobei der Grad des Sharings auf $t = \lfloor (N-1)/2 \rfloor$ gesetzt wird. Damit ist garantiert, dass die (weniger als $N/2$) unehrlichen Autoritäten den Secret-Key nicht rekonstruieren können, die Shares der (mehr als $N/2$) ehrlichen Autoritäten den Secret-Key jedoch eindeutig definieren.

Der Secret-Key z soll also geshart werden durch die Shares $z_1, \ldots, z_N$, und die Autoritäten sind committet zu ihren Shares durch $Z_1 = g^{z_1}, \ldots, Z_N = g^{z_N}$. Der Public-Key ist dann $Z = g^z$. Für das Polynom-Sharing nehmen wir einfachheitshalber an, dass die Auswertungsstellen der i-ten Autorität $\alpha_i = i$ ist (für $i = 1, \ldots, N$).

Ein entsprechendes Sharing kann als MPC berechnet werden (Abschnitt 2.8), wobei DL-Commitments verwendet werden: Jeder Spieler A_i wählt als Input einen zufälligen Wert $\widehat{z}_i$, und Ziel der Berechnung ist ein Sharing des Secret-Keys $z = \widehat{z}_1 + \ldots + \widehat{z}_N$, wobei die Spieler zu ihrem Share committet sind. Das Protokoll braucht keine Multiplikation und ist sehr effizient. Wir geben vollständigkeitshalber das fertige Protokoll an:

1. Jede Autorität A_i wählt ein zufälliges $\widehat{z}_i \in \mathbb{Z}_q$ und broadcastet das zugehörige Commitment $\widehat{Z}_i = g^{\widehat{z}_i}$.
2. Der Secret-Key z ist definiert als $z = \widehat{z}_1 + \ldots + \widehat{z}_N$, und der Public-Key ist definiert als $Z = \widehat{Z}_1 \cdot \ldots \cdot \widehat{Z}_N$. Die Autoritäten müssen sich nun Shares zu z berechnen:
   - 2.1 Jede Autorität A_i shart ihr $\widehat{z}_i$ mittels dem Commitment Sharing Protokoll aus Abschnitt 2.7.3: Dazu wählt A_i ein zufälliges Polynom $f_i(x)$ mit $f_i(0) = \widehat{z}_i$ vom Grad t und committet sich zu den Koeffizienten. Der Share für A_j ist also $z_{ij} = f_i(j)$, und ein Commitment Z_{ij} zu z_{ij} ist dank der homomorphen Eigenschaft des Commitment Schemes implizit definiert. A_i sendet an jede Autorität A_j den Share z_{ij}, und die Empfängerin A_j überprüft, dass der empfangene Wert zu Z_{ij} passt. Sonst klagt sie A_i mittels Broadcast an, worauf A_i den Share z_{ij} per Broadcast an A_j senden muss.
   - 2.2 Jede Autorität A_j berechnet ihren Share z_j von z als Summe der erhaltenen Shares: $z_j = z_{1j} + \ldots + z_{Nj}$. Das entsprechende Commitment ist dann $Z_j = Z_{1j} \cdot \ldots \cdot Z_{Nj}$. Diese Shares liegen alle auf dem Polynom $f(x) = f_1(x) + \ldots + f_N(x)$, und der Secret Key ist $z = f(0)$.

3.4.4 Verteilte Entschlüsselung

Ziel dieses Abschnitts ist die Konstruktion eines Protokolls, welches ein Chiffrat e entschlüsselt, ohne dass die Autoritäten den Secret-Key rekonstruieren müssen. Um ein Chiffrat $e = (x, y)$ zu entschlüsseln, muss x^z berechnet werden. Der Secret-Key z ist jedoch geshart unter den Autoritäten. Wir verwenden nun den Umstand, dass das Secret-Sharing Scheme linear ist, das heisst, z lässt sich schreiben als eine gewichtete Summe der Shares $z_1, \ldots, z_N$:

$$z = \sum_{i=1}^{N} w_i z_i, \text{ für geeignete } w_i\text{-s.}$$

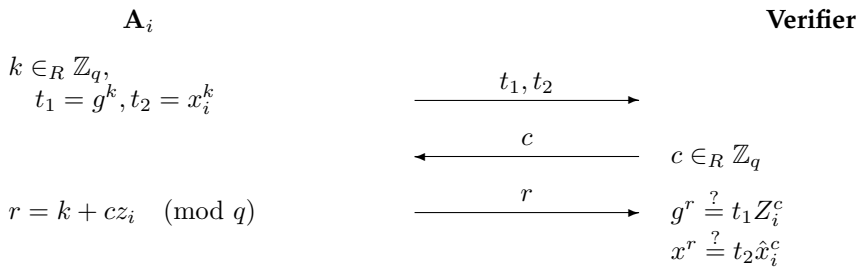
Folglich lässt sich x^z wie folgt berechnen:

$$x^z = x^{\sum w_i z_i} = \prod_{i=1}^{N} \left(x^{z_i}\right)^{w_i}.$$

Es genügt also, wenn jede Autorität A_i den Wert $\hat{x}_i = x^{z_i}$ berechnet und publiziert (und natürlich beweist, dass $\hat{x}_i = x^{z_i}$). Dieser Beweis ist ein Beweis, dass zwei diskrete Logarithmen gleich sind, also dass $\mathrm{DL}_g Z_i = \mathrm{DL}_x \hat{x}_i$, und kann entsprechend dem Abschnitt 3.3.1 konstruiert werden, wobei der Gruppenhomomorphismus gegeben ist durch

$$f : \zeta \mapsto (g^\zeta, x^\zeta),$$

und die Autorität beweist, dass sie ein Urbild ζ kennt (nämlich $\zeta = z_i$), so dass $f(\zeta) = (Z_i, \hat{x}_i)$. Das fertige Protokoll sieht dann wie folgt aus:



Dieses Protokoll wird nun nicht-interaktiv gemacht (siehe Abschnitt 3.3.3). Der Beweis der Korrektheit von $\hat{x}_i$ ist also das Tupel (t_1, t_2, r), und die Überprüfung des Beweises erfolgt, indem die beiden Prädikate des Verifiers oben ausgewertet werden für $c = H(t_1, t_2)$.

3.5 Wahlprotokoll mit homomorpher Verschlüsselung

In diesem Abschnitt kombinieren wir nun die Techniken der letzten Abschnitte, und erhalten damit das Wahlprotokoll von Cramer, Gennaro und Schoenmakers [CGS97]. Dabei handelt es sich um das effizienteste bekannte Wahlprotokoll, welches auf homomorpher Verschlüsselung basiert.

3.5.1 Codierung der Stimmen

Bei Wahlprotokollen mit homomorpher Verschlüsselung wird immer die *Summe* der abgegebenen Stimmen berechnet. Bei einfachen Ja/Nein-Abstimmungen kann „Ja“ als 1 und „Nein“ als 0 codiert werden. Aus der Summe der Stimmen und der Anzahl abgegebener Stimmen lässt sich dann die Anzahl „Ja“-Stimmen und die Anzahl „Nein“-Stimmen berechnen. Falls auch Stimmenthaltung erlaubt sein soll (also die Abgabe einer leeren Stimme), könnte zum Beispiel „Ja“ als 1, „Nein“ als -1, und „leer“ als 0 codiert werden. Aus der Summe kann dann die Differenz der „Ja“- und „Nein“-Stimmen berechnet werden. Die absolute Anzahl „Ja“-Stimmen lässt sich aber nicht bestimmen.

Um beliebige Kandidatenwahlen durchzuführen ist ein spezielleres Encoding vonnöten. Wir betrachten eine 1-out-of-L Wahl. Das heisst, es gibt L Kandidaten (wobei ein Kandidat eine Person oder eine Auswahlmöglichkeit sein kann), und jede Wählerin kann genau einem Kandidaten seine Stimme geben. Wenn eine Wählerin nun für den i-ten Kandidaten Stimmen will (für $1 \leq i \leq L$) setzt sie ihre Stimme auf M^{i-1}, wobei M die Anzahl wahlberechtigter Personen ist (oder ein upper-bound davon). Damit lässt sich aus der Summe der abgegebenen Stimmen einfach die Anzahl Stimmen für jeden Kandidaten eruieren. Natürlich muss $q \geq M^L$ gelten, damit beim Auszählen kein „Überlauf“ vorkommt.

Wir bezeichnen die Menge der gültigen Stimmen mit $\mathcal{V} = \{v_1, \ldots, v_L\}$. Es ist klar, dass für jedes Setting (mit endlich vielen Kandidaten) eine Menge $\mathcal{V}$ angegeben werden kann, selbst wenn Wähler mehrere Stimmen abgeben dürfen oder irgendwelche lustigen Bedingungen erfüllt sein müssen (z.B. darf jeder Wähler drei Stimmen abgeben, darf jedoch dem gleichen Kandidaten höchstens zwei Stimmen geben, ausser Hansli, dem darf man alle drei Stimmen geben). Bei vielen Kandidaten und insbesondere bei komplizierten Bedingungen wird die Menge $\mathcal{V}$ jedoch sehr gross.

3.5.2 Setup

In der Setup-Phase generieren die Autoritäten gemeinsam ein Secret-Key/Public-Key Paar für die homomorphe Verschlüsselungsfunktion, wobei der Secret-Key unter den Autoritäten geshart ist (siehe Abschnitt 3.4.3). Selbstverständlich müssen sich die Autoritäten auch auf das Thema der Wahl und die Menge $\mathcal{V}$ der gültigen Stimmen einigen, sowie auf Termine und so weiter.

3.5.3 Stimmabgabe

Bei der Stimmabgabe wählt jede Wählerin ihre Stimme $v \in \mathbb{Z}_q$ und verschlüsselt sie als $e = E_Z(v, \alpha)$ für ein zufälliges Element $\alpha \in \mathbb{Z}_q$. Nun muss die Wählerin einen Beweis konstruieren, dass die Stimme in e auch wirklich gültig ist, also dass $v \in \mathcal{V}$. Ein solcher Beweis wird im Anschluss gezeigt. Dann unterschreibt die Wählerin die verschlüsselte Stimme und den Gültigkeitsbeweis und sendet das gesamte Packet auf das Bulletin-Board.

Ein Beweis, dass die verschlüsselte Stimme e gültig ist, wird mit Hilfe der Techniken aus Abschnitt 3.3 konstruiert. Für die Menge $\mathcal{V} = \{v_1, \ldots, v_L\}$ von gültigen Stimmen heisst das, dass der Wähler beweisen muss, dass er ein $i \in \{1, \ldots, L\}$ und ein α kennt, so dass $e = E(v_i, \alpha)$.

Als erstes betrachten wir die Funktion

$$f : \mathbb{Z}_q \to G \times G, \quad \alpha \mapsto E(0, \alpha).$$

Es ist einfach zu sehen, dass diese Funktion f ein Gruppenhomomorphismus gemäss der Definition in Abschnitt 3.3 ist.

Die Wählerin muss nun beweisen, dass sie bezüglich f ein Urbild von $e \oslash E(v_1, 0)$ kennt, oder ein Urbild von $e \oslash E(v_2, 0)$, usw. Wissen (bzw. Existenz) eines Urbildes α von $e \oslash E(v_i, 0)$ mit $f(\alpha) = e \oslash E(v_i, 0)$ bedeutet, dass $e = E(v_i, \alpha)$, also dass e ein Chiffrat einer gültigen Stimme ist.

Ein genereller Beweis von Wissen eines Urbilds α von einem aus vielen Bildwerten wurde in Abschnitt 3.3.2 gegeben und kann einfach auf die konkrete Situation angepasst werden (siehe Übung).

3.5.4 Auszählen

Die Autoritäten addieren nun diejenigen verschlüsselten Stimmen (unter Ausnutzung des Homomorphismus), welche von einer wahlberechtigten Entität unterschrieben sind, und zu welchen der Gültigkeitsbeweis korrekt ist. Daraus ergibt sich die verschlüsselte Summe e_T. Diese wird nun entschlüsselt, wie in Abschnitt 3.4.4 beschrieben, und das Resultat ist die Summe T. Daraus lassen sich nun gemäss der Codierung der Stimmen die Summen der Kandidaten berechnen.

Wie bereits erwähnt ist diese Entschlüsselung nur für kleine Summen effizient. Tatsächlich ist die Berechnungskomplexität des Entschlüsselungsprotokolls in $\mathcal{O}(T)$. Bei Abstimmungen mit Codierung 1="Ja" und 0="Nein" ist dies kein Problem, weil ja $T \leq M$ gilt. Bei der vorgeschlagenen Codierung für L Kandidaten kann aber nur $T \leq M^L$ garantiert werden, und die Entschlüsselung ist nur für genügend kleine L möglich. Sonst kann statt der ElGamal Verschlüsselung die Verschlüsselungsfunktion von Paillier [Pai99] verwendet werden.

3.6 Receipt-Freeness

Ein bisher nicht betrachtetes Problem ist *Stimmenkauf* [BT94]. Mit Stimmenkauf wird die Koppelung zwischen dem Wahlverhalten der Wählerin und dem Erhalt einer Belohnung (z.B. Geld) bezeichnet. Selbstverständlich kann jemand der Wählerin Geld geben in der Hoffnung, dass sie dann wie gewünscht stimmt; dies ist einfach eine kapitalistische Form der Propaganda. Beim Stimmenkauf hingegen erfolgt die Bezahlung nur unter der Bedingung, dass die Wählerin wie verlangt stimmt. Stimmenkauf kann verhindert werden, indem verunmöglicht wird, dass die Wählerin beweisen kann, wie sie gestimmt hat. Folglich kann dann die Wählerin auch entgegen der Abmachung stimmen, ohne dass der Stimmenkäufer dies feststellen kann.

Ein Wahlprotokoll, in welchem die Wählerin den Inhalt ihrer eigenen Stimme nicht beweisen kann, heisst *receipt-free* ("Quittungs-frei"). Receipt-freeness kann als auferzwungene Privacy interpretiert werden: Privacy bedeutet, dass die Wählerin ihre Stimme geheim halten *kann*, während in einem receipt-free Protokoll die Wählerin ihre Stimme geheim halten *muss*. Selbst wenn die Wählerin möchte, kann sie die Privacy ihrer eigenen Stimme nicht verifizierbar aufheben.

Receipt-freeness verhindert neben Stimmenkauf auch Erpressung. Falls eine Wählerin gezwungen wird, eine bestimmte Stimme abzugeben, kann sie in einem receipt-free Protokoll trotzdem anders stimmen, ohne dass der Erpresser dies unterscheiden kann.

3.6.1 Definitionen

Leider gibt es bis heute keine allgemein akzeptierte Definition von Receipt-freeness. Das grundlegende Verständnis ist sicherlich, dass die Wählerin nicht jemanden überzeugen kann, wie sie gestimmt hat. Es bleiben aber viele Freiheitsgrade in der Definition. Wir geben im Folgenden eine Liste der wichtigsten zu berücksichtigenden Fragen und eine Auswahl von Antworten. Die Liste ist natürlich durch verschiedene in der Literatur verwendete Definitionen motiviert.

a) *Ehrlichkeit der Wählerin*: (1) Die Wählerin hält sich während der gesamten Protokolldauer ehrlich an das vorgegebene Protokoll. (2) Die Wählerin hält sich an das Protokoll, ausser dass sie keine Daten löscht, selbst wenn das Protokoll dies verlangt. (3) Die Wählerin darf jederzeit beliebig vom Protokoll abweichen.

   In der Literatur ist auch eine Mischform beliebt, nämlich dass sich die Wählerin in einer Initialisierungsphase (zum Beispiel bei der Registrierung) vollkommen an das Protokoll halten muss (inklusive Löschen von Information); in der eigentlichen Wahlphase darf sie vom Protokoll abweichen.

b) *Interaktion mit dem Stimmenkäufer*: (1) Die Wählerin kontaktiert den Stimmenkäufer erstmals nach der eigentlichen Wahlphase, und versucht dann, ihre Stimme zu beweisen. (2) Die Wählerin kann schon während der Stimmabgabe mit dem Stimmenkäufer interagieren, und laufend erhaltene Daten bekanntgeben und neue Instruktionen erhalten.

c) *Beweiskraft des Receipts*: (1) Ein Bit-String ist ein Receipt für eine Stimme v, falls gilt, dass er von einer v-stimmenden Wählerin erzeugt werden kann (mit überwältigender Wahrscheinlichkeit), und von einer nicht-v-stimmenden Wählerin nicht erzeugt werden kann (oder nur mit vernachlässigbarer Wahrscheinlichkeit). (2) Ein Bit-String ist selbst dann ein Receipt für v, falls eine v-stimmende Wählerin eine substanziell bessere Wahrscheinlichkeit hat, den String zu erzeugen, also eine nicht-v-stimmende Wählerin.

d) *Kooperation der Autoritäten*: (1) Autoritäten dürfen nicht mit dem Stimmenkäufer kooperieren; der Receipt muss also ohne Hilfe von Autoritäten überprüfbar sein. (2) Ein gewisse Anzahl Autoritäten dürfen beliebig mit dem Stimmenkäufer kooperieren, falls die Wählerin weiss, welche Autoritäten dies sind. (3) Eine gewisse Anzahl Autoritäten dürfen nach Belieben mit dem Stimmenkäufer kooperieren.

Für (a1) ist Receipt-freeness trivial: Die Wählerin wird einfach aufgefordert, alle relevanten Informationen nach der Stimmabgabe sofort zu löschen. In der Praxis ist (a1) aber wenig relevant, weil die Wählerin ja einen Receipt erzeugen *will*; warum sollte sie sich dann an das Protokoll halten?

Die Unterscheidung, ob die Wählerin während der Stimmabgabe mit dem Stimmenkäufer interagieren darf (b2) oder nicht (b1), ist in der Praxis oft irrelevant; wenn es nämlich eine interaktive Strategie gibt für den Stimmenkäufer, um schliesslich zur Überzeugung zu gelangen, dass die Wählerin wie verlangt gestimmt hat, dann kann die Wählerin diese Strategie auch selber anwenden (ohne Interaktion mit dem Stimmenkäufer). Einzig die Wahl von Zufallsbits durch den Stimmenkäufer ist etwas heikel; hier kann unter vernünftigen Annahmen die Wählerin aber einfach den Output einer sicheren Hashfunktion verwenden (Fiat-Shamir Heuristik).

Das dritte Kriterium (c) sollte genauer mittels Simulation definiert werden: Wir betrachten ein vorgegebenes Wahlprotokoll und eine beliebige Wählerin, und bezeichnen die lokale View der Wählerin zu Beginn das Wahlvorgangs mit V, und das durch den Stimmvorgang entstandene Transkript aller öffentlicher Kanäle (inklusive Bulletin-Board) mit T. In V sind insbesondere die abzugebende Stimme v und der Secret-Key z der Wählerin enthalten, oft aber auch für die Wahlprozedur notwendige Zufallsbits. Das Transkript T enthält oft nur chiffrierte Werte, weil die Kommunikation typischerweise verschlüsselt vonstatten geht.

Die Bedingung (c) verlangt nun, dass sich aus jeder lokalen View V mit Stimme v für jede beliebige gültige Stimme v' eine View V' berechnen lässt, so dass die Wahrscheinlichkeitsverteilung der beim Wählen erzeugten Transkripte T und T' mit überwältigender (c2) oder wenigstens mit nicht-vernachlässigbarer (c1) Wahrscheinlichkeit vom Stimmenkäufer nicht unterschieden werden können. Selbstverständlich müssen die beiden Views V und V' konsistent sein mit allenfalls vorhandenen publiken Informationen (z.B. mit dem Public-Key der Wählerin).

Um Stimmenkauf zu verhindern mag (c1) ausreichen, weil dann *jede* Wählerin (unabhängig davon, wie sie gestimmt hat), mit nicht-vernachlässigbarer Wahrscheinlichkeit trotzdem Geld vom Stimmenkäufer erhält. Im Kontext von Erpressung muss jedoch auf jeden Fall (c2) verlangt werden, damit die Wählerin, die dem Erpresser nicht gehorcht, nur mit vernachlässigbarer Wahrscheinlichkeit entdeckt werden kann.

Wenn Autoritäten mit dem Stimmenkäufer kooperieren dürfen, ohne dass die Wählerin dies weiss (d3), kann höchstens noch (c1) erreicht werden. Wir werden diese Behauptung hier aber nicht beweisen.

3.6.2 Receipt-Freeness im Standard-Modell

Receipt-freeness kann im Standard-Modell (mit paarweisen unsicheren Kanälen und einem Bulletin-Board, wobei die Wähler keine Daten löschen) nicht erreicht werden. Dies soll im Folgenden gezeigt werden.

Dazu betrachten wir ein (beliebiges) vorgegebenes Wahlprotokoll im Standard-Modell. Dieses Protokoll definiert, wie eine (ehrliche) Wählerin ihre Stimme in das Protokoll eingeben muss, zum Beispiel als (interaktive) Turing Maschine für die Wählerin. Diese Turing Maschine nimmt als Input die Stimme der Wählerin, eine Menge von Zufallsbits, und allenfalls eine Menge von geheimen Bits (z.B. den Secret Key der Wählerin).

Es ist klar, dass das Transkript der Kommunikation der Turing Maschine mit den Autoritäten von der Stimme der Wählerin abhängen muss. Für zwei verschiedene Stimmen können die beiden erzeugten Transkripte auf keinen Fall gleich sein; sonst wäre es für die Autoritäten unmöglich, am Schluss ein garantiert korrektes Resultat zu publizieren. Also muss das Transkript die Stimme der Wählerin eindeutig bestimmen.

Daraus folgt aber unmittelbar, dass der Input der Turing Maschine ein Receipt für die abgegebene Stimme ist, welcher jedermann überprüfen kann, der *alle* Kommunikationskanäle der Wählerin abhören kann. Der Stimmenkäufer verwendet einfach die gleiche Turing Maschine wie die Wählerin, und kann überprüfen, ob er das gleiche Transkript erhält wie jenes, welches er zwischen der richtigen Wählerin und den Autoritäten abgehört hat.

Wie schon früher erwähnt, kann Receipt-freeness trivialerweise erreicht werden, wenn die Wählerin Daten löscht. Interessanterweise genügt es dazu, dass die Wählerin nach der Initialisierungsphase Daten löscht; im eigentlichen Wahlvorgang braucht sie dann nichts mehr zu löschen.

3.6.3 Zusätzliche Anforderungen

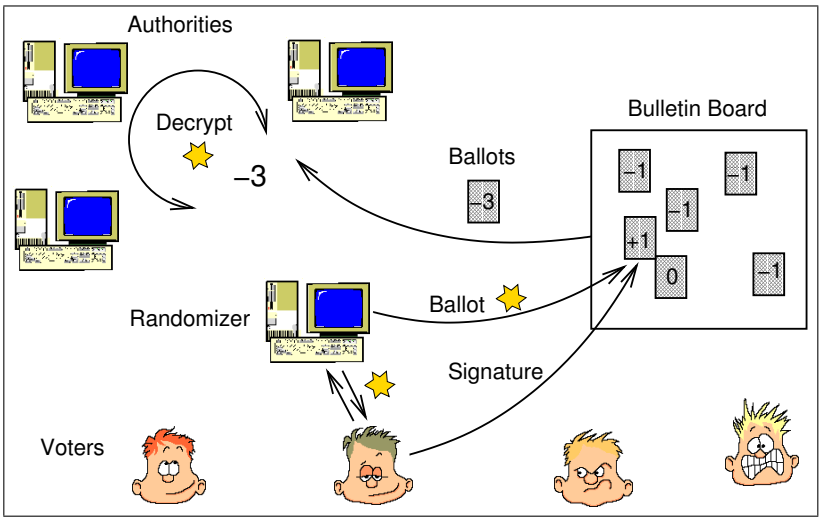
Wir haben im letzten Abschnitt gesehen, dass Receipt-freeness im Standard-Model nicht erreicht werden kann. Es stellt sich unmittelbar die Frage, welche zusätzlichen Anforderungen an das Modell gestellt werden müssen, damit Receipt-freeness erreichbar wird.

Die minimal notwendigen Annahmen für die Existenz von receipt-free Protokollen sind nicht bekannt. Die minimale Annahme, für welche receipt-free Wahlprotokolle *bekannt* sind, ist die Annahme von physikalisch abhörsicheren (aber nicht notwendigerweise authentischen) unidirektionalen Kanälen von den Autoritäten zu allen Wählern [HS00]. Ein solcher Kanal kann von einem Wahlkäufer unmöglich abgehört werden. Weil die Abhörsicherheit physikalisch und nicht kryptographisch ist, kann der Wahlkäufer auch keine verschlüsselten Daten lesen. Selbst wenn die Wählerin dem Wahlkäufer sagt, welche Daten sie über diesen Kanal empfangen hat, kann der Wahlkäufer diese Behauptung nicht verifizieren.

Wir werden in diesem Kapitel jedoch eine etwas stärkere Annahme treffen, nämlich die Existenz von physikalisch sicheren authentischen bidirektionalen Kanälen zwischen den Autoritäten und jeder Wählerin. Diese Voraussetzung ist zwar stärker als notwendig, aber dafür wird das Protokoll einfacher und effizienter.

Eine weitere Einschränkung ist die Annahme, dass die Autoritäten nicht beliebig mit dem Stimmenkäufer kooperieren dürfen. Solche Kooperationen können nur toleriert werden, falls jede Wählerin mindestens eine Autorität kennt, welche nicht mit dem Stimmenkäufer kooperiert. Wir werden im Folgenden jedoch Kooperationen zwischen Autoritäten und Stimmenkäufer ausschliessen.

3.7 Wahlprotokoll mit einem Randomizer



Ein receipt-free Wahlprotoll unterscheidet sich von einem normalen Wahlprotokoll nur in der Stimmabgabe. Es muss nämlich gewährleistet werden, dass die Wählerin ihre eigene Stimme nicht beweisen kann. Dies wird typischerweise erreicht, indem die Wählerin ihre Stimme nicht selber erzeugt, sondern die Autoritäten dabei mithelfen. Als Grundidee könnte man sich die Stimmengenerierung vorstellen als MPC Protokoll zwischen der Wählerin und den Autoritäten, wobei die Wählerin ihre eigene Stimme als Input gibt, und die Autoritäten die Zufälligkeit, welche zur Berechnung des Chiffrats notwendig ist. Dieser Ansatz würde aber mit klassischen MPC Protokollen nicht funktionieren, weil da nämlich die Wählerin ihren Input für das MPC Protokoll beweisen kann.

Stattdessen verwenden wir die homomorphe Eigenschaft der Verschlüsselungsfunktion: Die Wählerin schickt ihre verschlüsselte Stimme über einen physikalisch sicheren Kanal einer speziellen Autorität, dem Randomizer. Dieser empfängt die Stimme und randomisiert sie, indem er eine zufällige Verschlüsselung von 0 zum Chiffrat addiert. Dann beweist der Randomizer der Wählerin, dass dieses neue Chiffrat wirklich die gleiche Stimme enthält. Daraufhin schickt der Randomizer das randomisierte Chiffrat auf das Bulletin-Board, und die Wählerin sendet eine dazu passende Signatur.

Was noch fehlt, ist ein Gültigkeitsbeweis für das randomisierte Chiffrat, analog dem Gültigkeitsbeweis aus Abschnitt 3.5.3. Diesen Gültigkeitsbeweis kann weder die Wählerin noch der Randomizer generieren, sie beide können ja das randomisierte Chiffrat nicht öffnen. Also müssen die Wählerin und der Randomizer diesen Beweis in einem interaktiven Protokoll erzeugen.

Wir brauchen also zwei Beweise: Erstens muss der Randomizer der Wählerin beweisen, dass das randomisierte Chiffrat die gleiche Stimme enthält wie das ursprüngliche Chiffrat, und zweitens müssen die Wählerin und der Randomizer gemeinsam einen nicht-interkativen Beweis erzeugen, dass die randomisierte Stimme gültig ist. Beide Beweise dürfen natürlich die Receipt-freeness des Protokolls nicht beeinträchtigen.

3.7.1 Randomisierungsbeweis

Die Wählerin hat ein Chiffrat e mit einer (gültigen) Stimme generiert. Nun soll der Randomizer ein randomisiertes Chiffrat e^* mit der gleichen Stimme erzeugen und beweisen, ohne dass die Wählerin anschlies-

send den Inhalt der randomisierten Stimme jemand anderem beweisen kann.

Es scheint paradox, dass der Randomizer der Wählerin beweist, dass e^* und e die gleiche Stimme enthalten, und anschliessend die Wählerin trotzdem e^* nicht beweisen kann (sie kann ja e öffnen). Die Grundidee des Beweises ist, dass der Randomizer einen nicht-übertragbaren Beweis an die Wählerin sendet, einen sogenannten Designated-Verifier Beweis. Ein solcher Beweis beweist nicht die eigentlich Aussage (also dass e^* eine Randomisierung von e ist), sondern eine OR-Verknüpfung der eigentlichen Aussage und der Aussage, dass der Beweiser den Secret-Key der Wählerin kennt. Der Randomizer beweist also die folgende Aussage: „Entweder ist e^* eine Randomisierung von e, oder ich kenne den Secret-Key der Wählerin". Dieser Beweis wird die Wählerin überzeugen, dass e^* wirklich eine Randomisierung von e ist; wenn sie aber den Beweis (zum Beispiel an einen Stimmenkäufer) weitergibt, verliert er jeden Wert, weil die Wählerin selbstverständlich ihren eigenen Secret-Key kennt.

Als erstes konstruieren wir ein Protokoll, mit welchem der Randomizer der Wählerin beweisen kann, dass e^* eine Randomisierung von e ist (dieses Protokoll ist aber noch nicht receipt-free). Hierzu verwenden wir wieder den Gruppenhomomorphismus $f : \alpha \mapsto E(0, \alpha)$, und der Randomizer beweist Wissen eines Urbilds von $e^* \oslash e$. Das resultierende Protokoll ist gerade das Schnorr-Protokoll.

Randomizer		**Wählerin**
kennt α mit $e^* = e \otimes E(0, \alpha)$		kennt e, e^*
$k \in_R \mathbb{Z}_q, t = E(0, k)$	$\xrightarrow{t}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_q$
$r = k + c\alpha \pmod q$	$\xrightarrow{r}$	$E(0, r) \stackrel{?}{=} t \otimes (e^* \oslash e)^c$

Als nächstes konstruieren wir ein Protokoll, mit welchem die Wählerin einem beliebigen Verifier beweisen kann, dass sie ihren eigenen Secret-Key z_v, passend zu ihrem Public-Key $Z_v = g^{z_v}$ kennt. Hierzu verwenden wir den Gruppenhomomorphismus $f : \zeta \mapsto g^\zeta$, und die Wählerin beweist Wissen eines Urbilds von Z_v:

Wählerin		**Verifier**
kennt z_v mit $g^{z_v} = Z_v$		kennt Z_v
$k \in_R \mathbb{Z}_q, t = g^k$	$\xrightarrow{t}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_q$
$r = k + cz_v \pmod q$	$\xrightarrow{r}$	$g^r \stackrel{?}{=} t \cdot Z_v^c$

Und zu guter letzt konstruieren wir nun die OR-Verknüpfung der beiden obigen Protokolle. Dies ist dann ein Protokoll, welches der Randomizer gegenüber der Wählerin nur besteht, wenn e^* eine Randomisierung von e ist, aber die Wählerin gegenüber dem Stimmenkäufer immer besteht. Wir zeigen die Variante für den Randomizer, in welcher das Protokoll zum Beweis von Wissen des Secret-Keys der Wählerin simuliert wird:

Randomizer		**Wählerin**
kennt α mit $e^* = e \otimes E(0,\alpha)$		kennt e, e^*, Z_v
$k_1 \in_R \mathbb{Z}_q, t_1 = E(0,k_1)$, $c_2, k_2 \in_R \mathbb{Z}_q, t_2 = g^{k_2}/Z_v^{c_2}$	$\xrightarrow{t_1, t_2}$	
	$\xleftarrow{c}$	$c \in_R \mathbb{Z}_q$
$c_1 = c - c_2 \pmod q$, $r_1 = k_1 + c_1\alpha \pmod q$	$\xrightarrow{r_1, r_2, c_1, c_2}$	$c_1 + c_2 \stackrel{?}{=} c \pmod q$, $E(0,r_1) \stackrel{?}{=} t_1 \otimes (e^* \oslash e)^{c_1}$, $g^{r_2} \stackrel{?}{=} t_2 \cdot Z_v^{c_2}$

Ein nicht-interaktiver Beweis besteht aus einem Tupel (r_1, r_2, c_1, c_2), welches das folgende Prädikat erfüllt:

$$c_1 + c_2 \stackrel{?}{=} H\big(E(0,r_1) \oslash (e^* \oslash e)^{c_1}, g^{r_2}/Z_v^{c_2}\big) \pmod q.$$

Es ist klar, dass dieses Tupel ein Beweis ist, dass e^* eine Randomisierung von e ist, falls es vom Randomizer an die Wählerin geschickt wird, aber völlig wertlos ist, wenn es von der Wählerin an den Stimmenkäufer geschickt wird.

3.7.2 Gültigkeitsbeweis

Wir haben in der Übung gesehen, wie ein (nicht-interaktiver) Gültigkeitsbeweis für eine Stimme e konstruiert werden kann. Wir müssen nun diesen Beweis so modifizieren, dass es ein Beweis für e^* ist, wobei die Wählerin diesen Beweis nicht verwenden können darf, um die Stimme e^* zu beweisen.

Die Idee des Beweises ist, dass der Randomizer mit der Wählerin den interaktiven Gültigkeitsbeweis für e durchläuft, und parallel dazu einen Gültigkeitsbeweis für e^* mit einem beliebigen Verifier ausführt (dieser zweite Beweis wird dann mit der Fiat-Shamir Heuristik nicht-interaktiv gemacht). Selbstverständlich kann der Randomizer den Challenge des Verifiers nicht beantworten (er kennt ja den Inhalt von e^* gar nicht), er kann aber der Wählerin einen geschickt gewählten Challenge senden, so dass sich aus deren Antwort gerade seine Antwort an den Verifier berechnen lässt.

Wir betrachten eine akzeptierende Konversation zwischen der Wählerin und dem Randomizer eines Gültigkeitsbeweises für e. Diese Konversation sieht aus Sicht des Randomizers wie folgt aus:

Wählerin		**Randomizer**
	$\xrightarrow{t_1, \ldots, t_L}$	
	$\xleftarrow{c}$	$c = \ldots$
	$\xrightarrow{r_1, \ldots, r_L, c_1, \ldots, c_L}$	$c \stackrel{?}{=} \sum_{j=1}^{L} c_j \pmod q$, $\forall j \in \{1, \ldots, L\}$: $E(0,r_j) \stackrel{?}{=} t_j \otimes \big(e \oslash E(v_j, 0)\big)^{c_j}$

Nun soll der Randomizer aus dieser Konversation eine neue Konversation generieren, welche als Beweis für die Gültigkeit von $e^* = e \otimes E(0,\alpha)$ akzeptiert wird. Diese neue Konversation soll völlig zufällig und unabhängig von der Konversation mit der Wählerin sein.

Als erstes randomisiert der Randomizer in der obigen Konversation die Nachrichten der Wählerin. Das heisst, er berechnet sich aus der Konversation mit der Wählerin eine neue, zufällige Konversation, welche

die Gültigkeit von e beweist. Hierzu wählt der Randomizer $2L$ zufällige Werte

$$c'_1, \ldots, c'_L, r'_1, \ldots, r'_L \in_R \mathbb{Z}_q, \quad \text{wobei } c'_1 + \ldots + c'_L = 0.$$

Ziel ist nun, Werte $t'_1, \ldots, t'_L$ zu finden, so dass die Konversation

$$\big(t_1 \otimes t'_1, \ldots, t_L \otimes t'_L\big), c, \big(r_1 + r'_1, \ldots, r_L + r'_L, c_1 + c'_1, \ldots, c_L + c'_L\big)$$

akzeptierend für e ist. Es ist einfach verifizierbar, dass dies der Fall ist für

$$t'_j = E(c'_j v_j, r'_j) \oslash e^{c'_j} \text{ für } j = 1, \ldots, L.$$

Nun haben wir also einen Beweis für die Gültigkeit von e, der völlig unabhängig ist vom Gültigkeitsbeweis der Wählerin. Wir müssen allerdings einen Beweis für die Gültigkeit von $e^* = e \otimes E(0, \alpha)$ konstruieren. Der obige Beweis lässt sich aber einfach dahingehend adaptieren, indem einfach alle Vorkommen von e ersetzt werden durch e^*, und die dritte Nachricht des Beweises entsprechend angepasst wird. Der Gültigkeitsbeweis für e^* ist gegeben durch die Konversation

$$\big(t''_1, \ldots, t''_L\big), c, \big(r''_1, \ldots, r''_L, c''_1, \ldots, c''_L\big),$$

wobei für $j = 1, \ldots, L$

$$\begin{aligned} t''_j &= t_j \otimes E(c'_j v_j, r'_j) \oslash e^{c'_j}, \\ c''_j &= c_j + c'_j \pmod q, \\ r''_j &= r_j + r'_j + \alpha c''_j \pmod q. \end{aligned}$$

Das Protokoll, mit welchem die Wählerin und der Randomizer gemeinsam einen Gültigkeitsbeweis für e^* erzeugen, sieht aus Sicht der Wählerin also aus wie ein normaler interaktiver Gültigkeitsbeweis. Der Randomizer berechnet aber nach der ersten Nachricht der Wählerin die Werte $t''_1, \ldots, t''_L$, und bestimmt den Challenge c als Hashwert dieser Werte. Aus der Antwort der Wählerin auf diesen Challenge kann dann der Randomizer die Werte $r''_1, \ldots, r''_L$ berechnen, und das Tupel

$$(t''_1, \ldots, t''_L, c''_1, \ldots, c''_L, r''_1, \ldots, r''_L)$$

ist dann ein nicht-interaktiver Gültigkeitsbeweis des Chiffrats e^*. Der Randomizer muss nun diesen Beweis zusammen mit dem Chiffrat e^* auf das Bulletin-Board senden, und die Wählerin sendet eine Signatur zu e^* auf das BB, um diese Stimme als ihre Stimme anzuerkennen.

Yupi!

Literaturverzeichnis

[Bea91] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, pp. 75–122, 1991.

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, Oct. 1988.

[BFM88] M. Blum, P. Feldman, and S. Micali. Non-Interactive Zero Knowledge and Its Applications. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pp. 103–112, 1988.

[BGP89] P. Berman, J. A. Garay, and K. J. Perry. Towards optimal distributed consensus (extended abstract). In *Proc. 21st ACM Symposium on the Theory of Computing (STOC)*, pp. 410–415, 1989.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th Annual Symposium on the Theory of Computing (STOC)*, pp. 1–10, 1988.

[BT94] J. Cohen Benaloh and D. Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proc. 26th ACM Symposium on the Theory of Computing (STOC)*, pp. 544–553. ACM, 1994.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th Annual Symposium on the Theory of Computing (STOC)*, pp. 11–19, 1988.

[CDM00] R. Cramer, I. Damgård, and U. Maurer. General secure multi-party computation from any linear secret sharing scheme. In *Advances in Cryptology – EUROCRYPT 2000*, B. Preneel (Ed.), Lecture Notes in Computer Science, Berlin: Springer-Verlag, vol. 1807, pp. 321–339 , 2000.

[CGS97] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European Transactions on Telecommunications*, 8:481–489, September 1997.

[Cha81] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[DS82] D. Dolev and H. R. Strong. Polynomial algorithms for multiple processor agreement. *Proc. 14th ACM Symposium on Theory of Computing (STOC)*, pp. 401–407, 1982.

[FS86] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problem. In *Advances in Cryptology – CRYPTO' 86*, Lecture Notes in Computer Science, Berlin: Springer-Verlag, vol. 263, pp. 186–194, 1987.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *Proc. 17th ACM Symposium on Theory of Computing (STOC)*, pp. 291–304, 1985.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, February 1989.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symposium on the Theory of Computing (STOC)*, pp. 218–229, 1987.

[HM97] M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multi-party computation. *Journal of Cryptology*, vol. 13, no. 1, pp. 31–60, 2000.

[HS00] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology — EUROCRYPT '00*, volume 1807 of *Lecture Notes in Computer Science*, 2000.

[Lyn96] N. A. Lynch. Distributed Algorithms. *Morgan Kaufmann series in data management systems*, Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 1996.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pp. 223–238, 1999.

[Ped91] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pp. 129–140. Springer-Verlag, 1991.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[RB89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *Proc. 21st ACM Symposium on the Theory of Computing (STOC)*, pp. 73–85, 1989.

[Sch89] C.P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, vol. 4, 1991, pp. 161–174.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[vT88] H.C.A. van Tilborg. An introduction to cryptology. *Kluwer Academic Publishers*, 1988.

[Yao82] A. C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pp. 160–164. IEEE, 1982.

[Weg] I. Wegener. Theoretische Informatik. Eine algorithmenorientierte Einführung. *Teubner Verlag*, 1993, 235 Seiten, 2. Auflage 1999.