Protokollkomposition und Komplexität

Zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften von der Fakultät für Informatik der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Dominique Unruh

aus Hannover

Tag der mündlichen Prüfung:	6. Dezember 2006
Erster Gutachter:	Prof. Dr. Roland Vollmar
Zweiter Gutachter:	Prof. Dr. Michael Backes

In Gedenken an
Prof. Dr. Thomas Beth

Inhaltsverzeichnis

1. Einleitung

1.1. Überblick

Eine der ältesten Fragen der exakten, d. h. die Sicherheit von Protokollen *beweisenden* Kryptologie ist die Suche nach geeigneten Definitionen der Sicherheit. Dabei unterscheiden wir zwischen anwendungsspezifischen Definitionen, wie z. B. der Definition eines sicheren Schlüsselaustauschprotokolls, sowie allgemeinen Definitionen, die in allgemeinster Weise festlegen, was wir unter einem sicheren Protokoll für eine vorgegebene Aufgabenstellung verstehen. Diese allgemeinen Sicherheitsmodelle erreichten ihren (vorläufigen) Höhepunkt mit der Einführung von *simulationsbasierten Sicherheitsmodellen mit Umgebung*, die 2001 unabhängig von Pfitzmann und Waidner [PW01] und Canetti [Can01] vorgestellt wurden. Diese Modelle haben zwei große Vorteile: Zum einen erlauben sie es, die zu implementierende Aufgabenstellung in einfacher Weise durch Entwurf einer sog. *idealen Funktionalität* zu modellieren, und zum anderen ermöglichen sie sichere Komposition: Es ist möglich, ein komplexes Protokoll zunächst unter Benutzung verschiedener Primitiven zu entwerfen, und dann diese Primitiven durch sie implementierende Protokolle zu ersetzen, was einen modularen Protokollentwurf und -beweis ermöglicht.

Seitdem waren diese Sicherheitsmodelle mit Umgebung Objekt intensiver Forschungen. So wurde unter anderem gezeigt, daß einige elementare kryptographische Aufgabenstellungen wie Commitment und Münzwurf ohne die Benutzung zusätzlicher Annahmen nicht sicher realisierbar sind [CF01]. Es kam somit die Frage auf, ob eine derart strenge Sicherheitsdefinition überhaupt notwendig ist, oder ob nicht eine schwächere existiert, die trotzdem noch Komposition ermöglicht. Diese Frage wurde von Lindell [Lin03] beantwortet, der zeigte, daß simulationsbasierte Sicherheit mit Umgebung nicht nur hinreichend, sondern auch *notwendig* für die Komposition ist.

Dabei stellte Lindell erstmalig klar heraus, daß es zwei Varianten dieser Sicherheitsdefinitionen gibt, sowie zwei Varianten der Komposition. Die Varianten der Sicherheit, die in dieser Arbeit *allgemeine* und *spezielle Sicherheit* genannt werden sollen, unterscheiden sich in der Reihenfolge der Quantoren in der Definition; ein unscheinbarer Unterschied, der aber, wie wir noch genauer sehen werden, weitreichende Konsequenzen hat. Die allgemeine Sicherheit impliziert hierbei die spezielle. Der Unterschied zwischen den Varianten der Komposition, im folgenden *einfache* und *allgemeine Komposition* genannt, ist bereits substan-

tieller. Bei der einfachen Komposition ersetzen wir *eine* Instanz einer Primitive durch ein Unterprotokoll. Bei der allgemeinen Komposition hingegen dürfen wir *mehrere* Instanzen[1] der Primitive simultan ersetzen. In vielen Fällen wird man die stärkere Form der Komposition brauchen, da z. B. wenige Protokolle mit nur *einem* Commitment auskommen.

In dieser genaueren Sprechweise ist die Situation die folgende: Spezielle Sicherheit erlaubt einfache Komponierbarkeit [PW01], allgemeine Sicherheit gibt uns allgemeine Komponierbarkeit [Can01], und spezielle Sicherheit ist sogar *notwendig* für einfache Komponierbarkeit [Lin03].

Im Lichte dieser von Lindell initiierten Untersuchung der Beziehungen ergeben sich nun in natürlicher Weise die folgenden offenen Fragen: Welcher Begriff ist notwendig für die allgemeine Komponierbarkeit, ist es die allgemeine oder die spezielle Sicherheit oder ein dazwischenliegender Begriff? Oder sind allgemeine Sicherheit und spezielle Sicherheit gar äquivalent, und somit beide sowohl notwendig als auch hinreichend für spezielle wie allgemeine Komponierbarkeit? Kurz, welches sind die Implikationen und Trennungen zwischen den folgenden vier Begriffen: spezielle Sicherheit, allgemeine Sicherheit, einfache Komponierbarkeit und allgemeine Komponierbarkeit? Diesen Fragen gehen wir in dieser Arbeit nach.

1.2. Unsere Ergebnisse

Es stellt sich heraus, daß die Frage nach den Implikationen und Trennungen zwischen den Begriffen der speziellen Sicherheit, allgemeinen Sicherheit, einfachen Komponierbarkeit und allgemeinen Komponierbarkeit von anderen Details der Sicherheitsdefinition abhängt. So unterscheiden wir zunächst perfekte, statistische und komplexitätstheoretische Sicherheit: Bei der *perfekten Sicherheit* muß die Erfolgswahrscheinlichkeit eines Angriffs exakt Null sein, bei der *statistischen Sicherheit* darf eine vernachlässigbare Wahrscheinlichkeit eines erfolgreichen Angriffs bestehen, und bei der *komplexitätstheoretischen Sicherheit* beschränken wir uns zusätzlich auf polynomiell-beschränkte Angreifer.[2]

Weiterhin kann man zwischen *Sicherheit mit* und *ohne Auxiliary input* unterscheiden. Unter einem Auxiliary input verstehen wir eine nichtuniforme Eingabe, die die sog. Umgebung (einer der Angreifer) zu Beginn des Protokollaufs erhält, und die den Angriff unterstützen soll.

Zu guter Letzt unterscheiden wir noch zwischen *Sicherheit bzgl. der Ausgabe*

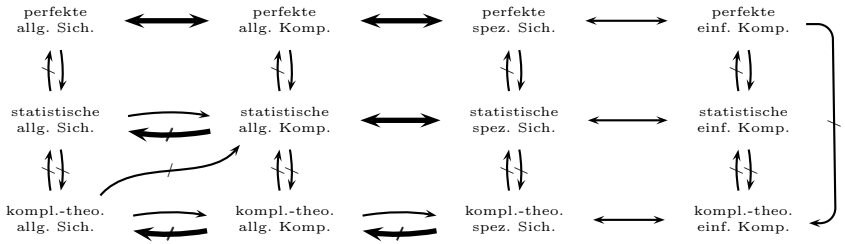
[1] Genauer: eine im Sicherheitsparameter polynomiell-beschränkte Anzahl von Instanzen.

[2] Genaugenommen kommt in der Definition der Sicherheit die „Wahrscheinlichkeit eines erfolgreichen Angriffs“ nicht explizit vor, sondern es wird von der „Ununterscheidbarkeit von realem und idealen Modell“ gesprochen. Die Vorstellung einer „Wahrscheinlichkeit eines erfolgreichen Angriffs“ soll aber an dieser Stelle genügen, bis wir in Kapitel 2 die notwendigen Details kennenlernen.

und *bzgl. der Sicht der Umgebung*. Hier handelt es sich um einen eher technischen Unterschied. Da aber beide Begriffe nicht äquivalent sind, müssen wir bei unseren Untersuchungen beide Varianten betrachten. In diesem Abschnitt stellen wir nur die Ergebnisse im Fall der Ausgabe der Umgebung vor, für die Sicherheit bzgl. der Sicht der Umgebung gelten die analogen Beziehungen, jedoch sind die Begriffe der allgemeinen und speziellen Komponierbarkeit nur bzgl. der Ausgabe der Umgebung definiert.

Wir erarbeiten für jede dieser Varianten alle Implikationen und Trennungen zwischen allgemeiner und spezieller Sicherheit und einfacher und allgemeiner Komponierbarkeit.

Im Falle der Sicherheit *ohne Auxiliary input* bzgl. der Ausgabe der Umgebung erhalten wir die in der folgenden Tabelle dargestellten Beziehungen. Die Ergebnisse dieser Arbeit sind dabei durch Fettdruck hervorgehoben. Man beachte, daß die Darstellung in dem Sinne vollständig ist, daß die Beziehung zwischen je zwei der Begriffe aus den angegebenen Pfeilen folgt.



Im Falle *mit Auxiliary input* ist unser trennendes Beispiel zwischen statistischer allgemeiner und spezieller Sicherheit nicht mehr gültig. Vielmehr fallen in diesem Fall diese Begriffe zusammen, und es ergeben sich für die Sicherheit *mit Auxiliary input* bzgl. der Ausgabe der Umgebung die folgenden Beziehungen:

Alle anderen Beziehungen zwischen den Sicherheitsbegriffen sind wie im Falle der Sicherheit ohne Auxiliary input.

Weiterhin basiert besagtes Gegenbeispiel auf Protokollen mit unbeschränkter Kommunikationskomplexität. Schränkt man sich auf Protokolle mit beschränkter Kommunikationskomplexität ein, fallen erneut statistische allgemeine und spezielle Sicherheit zusammen, und wir erhalten für Protokolle mit beschränkter Kommunikationskomplexität für die Sicherheit sowohl mit als auch ohne Auxiliary input erneut die Äquivalenzen:

Wieder sind die anderen Beziehungen nicht betroffen.

Die in diesen Diagrammen angegebenen Verhältnisse stellen die wesentlichen in dieser Arbeit aufgezeigten Beziehungen auf. Darüber hinaus ergeben sich weitere bislang unbekannte Implikationen und Trennungen zwischen den verschiedenen Varianten der Sicherheit, so zeigen wir z. B. daß die Sicherheit bzgl. der Sicht und der Ausgabe der Umgebung nicht in allen Fällen zusammenfallen. Da eine komplette Aufzählung der gezeigten Beziehungen zu unübersichtlich wäre und somit dem Überblickscharakter dieses Abschnittes zuwiderliefe, verzichten wir an dieser Stelle darauf. Wir verweisen dazu auf die kondensierte Aufstellung der in dieser Arbeit aufgezeigten Beziehungen in Anhang B.

Die Untersuchung der eben aufgeführten Beziehungen stellt das Hauptziel dieser Arbeit dar. Auf dem Weg zu dieser vollständigen Klassifizierung erarbeiten wir jedoch zwei Hilfsmittel, die auch unabhängig von der vorliegenden Aufgabenstellung interessant sind.

Zum einen untersuchen wir sog. *Time-lock puzzles*. Es handelt sich dabei um eine erstmals in [May93] vorgestellte und von [RSW96] ausgearbeitete Idee. Ein Time-lock puzzle ist ein (interaktives) Problem, bei dem die Schwierigkeit des Problems von der das Problem stellenden Maschine relativ genau eingestellt werden kann. Wir geben die erste formale Definition eines solchen Time-lock puzzles an und untersuchen Bedingungen für dessen Existenz. Wir verwenden Time-lock puzzles in den Kapiteln 5 und 6 zur Konstruktion von trennenden Gegenbeispielen. Es ist wahrscheinlich, daß die dort erarbeiteten Beweistechniken auch jenseits der Zielsetzung dieser Arbeit bei der Konstruktion trennender Beispiele in der Kryptologie Anwendung finden können. Insbesondere die in Kapitel 6 entwickelte Methodik, ein Time-lock puzzle mit einer sicheren Funktionsauswertung zu kombinieren, könnte ein mächtiges Werkzeug sein.

Ein weiteres Hilfsmittel bei der Analyse der Beziehungen sind die *universellen Umgebungen und Simulatoren*. Wir übertragen hier das Konzept des Nash-Equilibriums aus der Spieltheorie auf die behandelten Sicherheitsmodelle. Wir untersuchen die Existenz und die Komplexität universeller Umgebungen und Simulatoren. Eine Folgerung aus ihrer Existenz ist die Tatsache, daß die Reihenfolge der Quantoren in der Sicherheitsdefinition irrelevant wird; wir folgern daraus die oben bereits aufgeführte Äquivalenz zwischen statistischer allgemeiner und spezieller Sicherheit bei Protokollen mit beschränkter Kommunikationskomplexität. Aber darüber hinaus ergibt sich, daß im Falle der statistischen Sicherheit exponentiell-beschränkte Angreifer so mächtig wie unbeschränkte Angreifer sind (wenn die betrachteten Protokolle polynomiell-beschränkt sind). Das heißt man kann o. B. d. A. annehmen, daß nur exponentielle Angreifer vorkommen.[3]

[3] Genaugenommen gilt dies für den Simulator, die Umgebung und den Angreifer. Wir ver-

1.3. Aufbau dieser Arbeit

Kapitel 1 (Einleitung). Die grundlegende Problemstellung wird erläutert (Abschnitt 1.1), dann werden die Ergebnisse der Arbeit kurz zusammengefaßt (Abschnitt 1.2) und es wird ein Überblick über den Aufbau dieser Arbeit gegeben (Abschnitt 1.3). Weiterhin werden grundlegende Definitionen eingeführt und deren Eigenschaften beschrieben (Abschnitt 1.4).

Kapitel 2 (Simulationsbasierte Sicherheit). Dieses Kapitel führt die untersuchten Sicherheitsbegriffe ein. Dazu wird zunächst ein Überblick über die Geschichte solcher Sicherheitsmodelle gegeben (Abschnitt 2.1). Dann werden die in dieser Arbeit verwandten Definitionen der Sicherheit im Detail vorgestellt (Abschnitt 2.2), und schließlich das Problem der Komposition erörtert und die bisher bekannten Ergebnisse vorgestellt (Abschnitt 2.3).

Kapitel 3 (Die Mächtigkeit zufälliger Angriffe). In diesem Kapitel wird gezeigt, wie man zufällige Angriffe, d. h. Angriffe, bei denen alle gesandten Nachrichten zufällig gewählt werden, dazu verwenden kann, die Varianten der perfekten Sicherheit als äquivalent zu beweisen. Es wird gezeigt, daß perfekte allgemeine und spezielle Sicherheit äquivalent sind (Abschnitt 3.1). Weiter wird gezeigt, daß im Gegensatz zur statistischen Sicherheit bei perfekter Sicherheit die Sicherheit bzgl. der Sicht und bzgl. der Ausgabe zusammenfallen (Abschnitt 3.2).

Kapitel 4 (Ein Kompositionstheorem für statistische Sicherheit). Hier zeigen wir, daß im Falle der statistischen Sicherheit allgemeine Komposition tatsächlich möglich ist (Abschnitte 4.1 und 4.2). Außerdem geben wir einen Beweis, daß in Anwesenheit von Auxiliary input sogar statistische allgemeine und spezielle Sicherheit zusammenfallen (Abschnitt 4.3).

Kapitel 5 (Laufzeittrennungen und Time-lock puzzles). Wir geben ein einfaches trennendes Beispiel zwischen statistischer allgemeiner und spezieller Sicherheit an (Abschnitt 5.2). Dann führen wir das Werkzeug der Time-lock puzzles ein, untersuchen Bedingungen für deren Existenz, und übertragen das trennende Gegenbeispiel mit ihrer Hilfe auf den komplexitätstheoretischen Fall (Abschnitt 5.4).

Kapitel 6 (Verdeckte Time-lock puzzles und Komposition). In diesem Kapitel zeigen wir, daß komplexitätstheoretische spezielle Sicherheit *nicht* allgemeine

wenden hier wieder die vereinfachende Sprechweise, da wir die Details der Sicherheitsdefinition noch nicht eingeführt haben.

Komposition erlaubt. Das trennende Beispiel baut auf den Time-lock puzzles aus Kapitel 5 auf, und wir zeigen eine neue Möglichkeit, diese anzuwenden.

Kapitel 7 (Spieltheorie und exponentielle Angreifer). In diesem Kapitel untersuchen wir, wie man spieltheoretische Konzepte auf unsere Problemstellung anwenden kann. Dazu geben wir zunächst eine kurze Einführung in die Spieltheorie (Abschnitt 7.1). Dann demonstrieren wir, wie man unser Sicherheitsmodell spieltheoretisch auffassen kann (Abschnitt 7.2) und wie sich das Konzept eines Nash-Equilibriums auf diesen Fall überträgt und die Existenz von „universellen Umgebungen und Simulatoren“ garantiert (Abschnitt 7.3). Schließlich geben wir Anwendungen der erarbeiteten Techniken, u. a. zeigen wir, daß für Protokolle mit beschränkter Kommunikationskomplexität spezielle und allgemeine statistische Sicherheit zusammenfallen, und daß in gewissen Fällen exponentiell-beschränkte Angreifer so mächtig wie unbeschränkte sind (Abschnitt 7.4).

Kapitel 8 (Schlußbemerkungen). Wir geben ein Fazit unserer Arbeit und präsentieren einige offene Fragen.

Anhang. In Anhang A (Einige Lemmata) geben wir einige einfache Lemmata an, die der Vollständigkeit halber erwähnt werden sollten. In Anhang B (Beziehungen zwischen Sicherheitsbegriffen) listen wir alle in dieser Arbeit gezeigten Beziehungen zwischen Sicherheitsbegriffen in kondensierter Form auf. In Anhang F (Englische Fachausdrücke) werden die englischen Entsprechungen der wichtigsten in dieser Arbeit vorkommenden Begriffe gegeben.

Weiterhin vorhanden sind: Anhang C (Eigene Arbeiten), Anhang D (Lebenslauf), Anhang E (Abbildungsverzeichnis), Anhang G (Satz- und Definitionsverzeichnis), Anhang H (Symbolverzeichnis), Anhang I (Literaturverzeichnis) und Anhang J (Index).

1.4. Notation und grundlegende Definitionen

In diesem Abschnitt spezifizieren wir die verwendete mathematische Notation und geben einige grundlegende Definitionen und deren Eigenschaften an. Die hier präsentierten Definitionen sind allgemeiner Natur, die zu dem in dieser Arbeit untersuchten Sicherheitsmodell gehörigen werden in Kapitel 2 gegeben. Es wurde versucht, die in diesem Abschnitt gegebenen Informationen über den Index in Anhang J und das Symbolverzeichnis in Anhang H zugänglich zu machen, so daß dieser Abschnitt auch zunächst übersprungen und bei Bedarf auf ihn zurückgegriffen werden kann.

Es bezeichne $\mathbb{N}$ die Menge der natürlichen Zahlen (ohne 0), und $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. Es sei Σ ein (im folgendes festes, aber nicht weiter bestimmtes) Alphabet, und Σ^* bezeichnet die Wörter über Σ, $\lambda \in \Sigma^*$ das leere Wort, und $\Sigma^+ := \Sigma^* \setminus \{\lambda\}$. Wir nehmen an, Σ sei groß genug, um alle in dieser Arbeit definierten konstanten Wörter darstellen zu können. Sprechen wir von einer Funktion ohne nähere Angabe des Urbild- oder Bildbereiches, so meinen wir eine Funktion von $\mathbb{N}$ in die nichtnegativen reellen Zahlen (oder, falls aus dem Kontext ersichtlich, nach $\mathbb{N}_0$).

Wir identifizieren Zufallsvariablen und Verteilungen miteinander. Verwenden wir also eine Zufallsvariable X an einer Stelle, an der eine Verteilung verlangt würde, so meinen wir die Randverteilung von X. Verwenden wir eine Verteilung μ, wo eine Zufallsvariable erwartet wird, so meinen wir eine Zufallsvariable mit Randverteilung μ. Wir bezeichnen eine Zufallsvariable als diskret, wenn ihr Wertebereich abzählbar ist. Betrachten wir Wahrscheinlichkeitsräume, so geben wir nicht explizit den zugehörigen Meßraum an, sondern nehmen an, daß der kanonische verwendet wird. Das heißt konkret, ist X abzählbar, so ist jede Teilmenge von X meßbar, und sind über den Mengen X_i Meßräume definiert, so nehmen wir über $X_1 \times X_2$, X_1^* und $X_1^{\mathbb{N}}$ die entsprechenden Produkträume an (vgl. [Bau02, § 9]).

Die Menge aller polynomiell-beschränkten Funktionen bezeichnen wir mit $POLY$, d. h. $POLY := \{g : g \geq 0 \text{ und } \exists c > 0 : g(k) \in O(k^c)\}$. Weiter bezeichnet $POLY(f)$ die in f polynomiell-beschränkten Funktionen, also $POLY(f) := \{g : g \geq 0 \text{ und } \exists c > 0 : g(k) \in O(f(k)^c)\}$. Darüber hinaus sei $POLY(F) := \bigcup_{f \in F} POLY(f)$ für eine Menge F von Funktionen. Die Menge aller exponentiell-beschränkten Funktionen bezeichnen wir als EXP, also d. h. $EXP := \{g : g \geq 0 \text{ und } \exists c > 0 : g(k) \in O(2^{k^c})\}$. Entsprechend sind $EXP(f) := \{g : g \geq 0 \text{ und } \exists c > 0 : g(k) \in O(2^{f(k)^c})\}$ und $EXP(F) := \bigcup_{f \in F} EXP(f)$.

Unter einer interaktiven Turing-Maschine (ITM) verstehen wir eine probabilistische Turing-Maschine mit Ein-/Ausgabe, Arbeits- und Kommunikationsbändern. Bei der ersten Aktivierung erhält die ITM ihre Eingaben auf den Eingabebändern, über die Kommunikationsbänder kann sie Nachrichten zu anderen ITMs schicken (welche dann aktiviert werden), und wenn die ITM terminiert, ist der Inhalt ihrer Ausgabebänder ihre Ausgabe. Der Inhalt der Arbeitsbänder bleibt über mehrere Aktivierungen erhalten. (Für Details vgl. z. B. [Gol01, Abschnitt 4.2.1.1].) Wird eine Maschine A mit der Eingabe x gestartet, so schreiben wir dies kurz $A(x)$. Mit $\langle A(x), B(y)\rangle$ bezeichnen wir die Ausgabe der ITM B nach einer Interaktion der ITMs A und B bei Eingaben x bzw. y. Eine polynomiell-beschränkte ITM (PITM) ist eine ITM, deren Laufzeit (über alle Aktivierungen summiert) in der Länge ihrer ersten Eingabe polynomiell-beschränkt ist. D. h. wenn A eine PITM ist, darf $A(1^k, x)$ in k polynomiell-

beschränkte Laufzeit haben, aber nicht notwendigerweise in $|x|$.[4] Analog nennen wir auch eine normale Turing-Maschine polynomiell-beschränkt, wenn ihre Laufzeit in der Länge ihres *ersten* Arguments polynomiell-beschränkt ist.

Wir geben nun an, was es für uns bedeutet, wenn eine Funktion sehr klein ist oder sehr groß ist:

Definition 1.1 (Vernachlässigbar, überwältigend)
Eine Funktion μ heißt *vernachlässigbar*, wenn für jedes $c > 0$ gilt: Für hinreichend großes $k \in \mathbb{N}$ ist $\mu(k) < k^{-c}$.

Eine Funktion μ heißt *überwältigend*, wenn $1 - \mu$ vernachlässigbar ist.

Eine vernachlässigbare Funktion ist also eine, die schneller fällt als das Inverse jedes Polynoms. Die Bezeichnung „überwältigend“ macht Sinn, wenn von Wahrscheinlichkeiten gesprochen wird, eine überwältigende Wahrscheinlichkeit ist eine, die nur vernachlässigbar weit von 1 (sicher) entfernt ist.

Wir werden oft Wahrscheinlichkeitsverteilungen vergleichen wollen (z. B. die Verteilungen der Ausgabe einer Maschine in verschiedenen Situationen). In solch einer Situation wollen wir nicht nur zwischen gleich und ungleich unterscheiden, sondern auch von ähnlichen Verteilungen sprechen können. Daher brauchen wir ein Abstandsmaß zwischen Verteilungen:

Definition 1.2 (Statistische Ununterscheidbarkeit)
Es seien X und Y Zufallsvariablen mit Werten in M. Dann ist der *statistische Abstand* $\Delta(X;Y)$ definiert durch

$$\Delta(X;Y) := \max_T \big|P(X \in T) - P(Y \in T)\big|,$$

wobei T über alle meßbaren Teilmengen von M geht (im Falle von abzählbarem M sind dies alle Teilmengen).

Es seien $\{X_{k,z}\}_{k,z}$ und $\{Y_{k,z}\}_{k,z}$ über $k \in \mathbb{N}$ und $z \in Z$ indizierte Familien von Zufallsvariablen. Dann heißen $\{X_{k,z}\}_{k,z}$ und $\{Y_{k,z}\}_{k,z}$ *statistisch ununterscheidbar*, wenn eine vernachlässigbare Funktion μ existiert, so daß für alle $k \in \mathbb{N}$ und alle $z \in Z$ gilt:

$$\Delta(X_{k,z}; Y_{k,z}) \leq \mu(k).$$

[4] Üblicher ist die Konvention, eine ITM als polynomiell-beschränkt zu bezeichnen, wenn ihre Laufzeit polynomiell in der Länge *aller* Eingaben beschränkt ist. Die hier gewählte Definition erlaubt es aber, einige Sachverhalte in dieser Arbeit übersichtlicher darzustellen.

Da wir oft Tupel von Zufallsvariablen vergleichen, lassen wir im Argument von Δ oft die die Tupel umschließenden Klammern weg. Wir schreiben also z. B. verkürzend $\Delta(X, Y;\ X', Y')$ für den statistischen Abstand $\Delta\big((X, Y); (X', Y')\big)$ zwischen (X, Y) und (X', Y').

Einige wichtige Eigenschaften der statistischen Ununterscheidbarkeit, die wir im Laufe der folgenden Kapitel brauchen werden, gibt das folgende Lemma:

Lemma 1.3 (Eigenschaften der statistischen Ununterscheidbarkeit)

(i) Δ is wohldefiniert, d. h. für zwei Zufallsvariablen X und Y mit Werten in M wird das Maximum in Definition 1.2 angenommen.

(ii) Δ ist eine Metrik, d. h. für beliebige Zufallsvariable X, Y, Z gilt: $\Delta(X; Y) \geq 0$, $\Delta(X; Y) = 0$ gdw. X und Y die gleiche Verteilung haben, und $\Delta(X; Z) \leq \Delta(X; Y) + \Delta(Y; Z)$.

(iii) Sind X und Y diskrete Zufallsvariablen mit Werten in M, so ist

$$\Delta(X; Y) = \frac{1}{2} \sum_{a \in M} \big|P(X = a) - P(Y = a)\big|.$$

(iv) Statistische Ununterscheidbarkeit ist transitiv, d. h. sind $\{X_{k,z}\}_{k,z}$ und $\{Y_{k,z}\}_{k,z}$ statistisch ununterscheidbar, sowie $\{Y_{k,z}\}_{k,z}$ und $\{Z_{k,z}\}_{k,z}$, so sind auch $\{X_{k,z}\}_{k,z}$ und $\{Z_{k,z}\}_{k,z}$ statistisch ununterscheidbar.

(v) Weniger Information führt zu weniger Unterscheidbarkeit, d. h. für eine meßbare Funktion f und Zufallsvariablen X und Y gilt

$$\Delta(f(X);\ f(Y)) \leq \Delta(X; Y).$$

Ist f injektiv (und ihr Inverses meßbar), so gilt Gleichheit.

(vi) Unabhängige Zusatzinformation ändert den statistischen Abstand nicht, d. h. für Zufallsvariablen X, Y und Z, wobei X und Z stochastisch unabhängig, und Y und Z stochastisch unabhängig seien, ist

$$\Delta(X;\ Y) = \Delta(X, Z;\ Y, Z).$$

(Fortsetzung nächste Seite)

(Fortsetzung)

(vii) Für Zufallsvariablen X und Y und jede Menge C gilt: Ist $P(X \in C) = P(Y \in C) > 0$, so ist

$$\Delta(X|X \in C;\ Y|Y \in C) \leq \Delta(X;Y)/P(X \in C).$$

(Hierbei bedeutet $X|X \in C$ die Zufallsvariable X im nach $X \in C$ konditionierten Wahrscheinlichkeitsraum, und $Y|Y \in C$ analog.)

(viii) Für Zufallsvariablen X, Y und Z, wobei Z diskret sei, gilt

$$\Delta(X,Z;\ Y,Z) = \sum_z P(Z=z)\Delta(X|Z=z;\ Y|Z=z).$$

(ix) Es seien X und Y Folgen von Zufallsvariablen. Es sei $X_{\leq n} := X_1, \ldots, X_n$ das Präfix der Länge n von X und $Y_{\leq n}$ analog. Dann konvergiert

$$\Delta(X_{\leq n};\ Y_{\leq n}) \stackrel{n\to\infty}{\longrightarrow} \Delta(X;\ Y).$$

(x) Es sei X eine Folge von Zufallsvariablen, und X_n konvergiere fast sicher in der diskreten Topologie.[5] Dann konvergiert

$$\Delta(X_n;\ \lim X_\mu) \stackrel{n\to\infty}{\longrightarrow} 0.$$

Beweis: Zunächst zeigen wir (i), also die Existenz des Maximums

$$\max_T \big|P(X \in T) - P(Y \in T)\big| \tag{1.1}$$

wobei T über alle meßbaren Teilmengen von M geht.

Es seien hierfür μ_X und μ_Y die Randverteilungen von X und Y.

Da $\mu_X - \mu_Y$ ein endliches signiertes Maß[6] ist, existieren nach [Bau92, Satz 18.1 (Hahn-Zerlegung)] zwei Maße μ^+ und μ^- und zwei disjunkte meßbare Mengen $T \cup T^{\complement} = M$, so daß $\mu_X - \mu_Y = \mu^+ - \mu^-$ und $\mu^+(T^{\complement}) = \mu^-(T) = 0$. Für jede meßbare Menge $T' \subseteq M$ gilt nun

$$\begin{aligned} &P(X \in T') - P(Y \in T') = \mu^+(T') - \mu^-(T') \leq \mu^+(T') \\ &= \mu^+(T' \cap T) \leq \mu^+(T) = \mu^+(T) + \mu^-(T) = P(X \in T) - P(Y \in T). \end{aligned} \tag{1.2}$$

[5] Eine Folge konvergiert in der diskreten Topologie, wenn sie stationär wird. Eine Folge konvergiert fast sicher, wenn sie mit Wahrscheinlichkeit 1 konvergiert.

[6] Ein signiertes Maß ist analog zu einem Maß definiert, nur darf ein signiertes Maß auch negative Werte annehmen, vgl. [Bau92, § 18].

Es sei $\tilde{T} := T'$ falls $P(X \in T') - P(Y \in T') \geq 0$, und $\tilde{T} := M \setminus T'$ falls $P(X \in T') - P(Y \in T') < 0$. Dann erhalten wir für jedes meßbare $T' \subseteq M$

$$\big|P(X \in T') - P(Y \in T')\big| = P(X \in \tilde{T}) - P(Y \in \tilde{T}) \overset{(1.2)}{\leq} \big|P(X \in T) - P(Y \in T)\big|.$$

Somit wird das Maximum (1.1) von T errreicht.

Wir beweisen nun (ii), also daß Δ eine Metrik ist. $\Delta(X;Y) \geq 0$ folgt direkt aus der Definition von Δ. Die Dreiecksungleichung folgt aus

$$\begin{aligned}\Delta(X;Z) &= \max_T |P(X \in T) - P(Z \in T)| \\ &\leq \max_T \big(|P(X \in T) - P(Y \in T)| + |P(Y \in T) - P(Z \in T)|\big) \\ &\leq \max_T |P(X \in T) - P(Y \in T)| + \max_T |P(Y \in T) - P(Z \in T)| \\ &= \Delta(X;Y) + \Delta(Y;Z).\end{aligned}$$

Haben X und Y die gleiche Verteilung, so ist trivialerweise $\Delta(X;Y) = 0$. Umgekehrt gilt, falls $\Delta(X;Y) = 0$, daß für jede Menge T $P(X \in T) = P(Y \in T)$, und somit haben X und Y die gleiche Verteilung.

Wir zeigen nun (iii). Es sei $\Delta'(X;Y) := \frac{1}{2}\sum_{a \in M} |P(X = a) - P(Y = a)|$ und T_0 die Menge der a mit $P(X = a) \geq P(Y = a)$. Dann ist

$$\begin{aligned}&2(P(X \in T_0) - P(Y \in T_0)) \\ &\quad = \big(P(X \in T_0) - P(Y \in T_0)\big) + \big(P(Y \notin T_0) - P(X \notin T_0)\big) \\ &\quad = \sum_{a \in T_0} \big(P(X = a) - P(Y = a)\big) + \sum_{a \notin T_0} \big(P(Y = a) - P(X = a)\big) \\ &\quad = \sum_{a \in T_0} \big|P(X = a) - P(Y = a)\big| + \sum_{a \notin T_0} \big|P(X = a) - P(Y = a)\big| \\ &\quad = 2\Delta'(X;Y) > 0,\end{aligned}$$

also $\Delta'(X;Y) = |P(X \in T_0) - P(Y \in T_0)|$. Damit ist bereits $\Delta(X;Y) \geq \Delta'(X;Y)$.

Weiter gilt

$$\begin{aligned}\Delta(X;Y) &= \max_T \Big(\tfrac{1}{2}|P(X \in T) - P(Y \in T)| + \tfrac{1}{2}|P(X \notin T) - P(Y \notin T)|\Big) \\ &\leq \max_T \Big(\tfrac{1}{2}\sum_{a \in T} |P(X = a) - P(Y = a)| + \tfrac{1}{2}\sum_{a \notin T} |P(X = a) - P(Y = a)|\Big) \\ &= \tfrac{1}{2}\sum_{a \in M} |P(X = a) - P(Y = a)| = \Delta'(X;Y).\end{aligned}$$

Somit ist $\Delta(X;Y) = \Delta'(X;Y)$.

Aussage (iv) folgt direkt aus der Dreiecksungleichung für Δ und der Tatsache, daß die Summe zweier vernachlässigbarer Funktionen wieder vernachlässigbar ist.

Aussage (v) ergibt sich durch

$$\begin{aligned}\Delta(f(X);f(Y)) &= \max_T |P(X \in f^{-1}(T)) - P(Y \in f^{-1}(T))| \\ &\leq \max_T |P(X \in T) - P(Y \in T)| = \Delta(X;Y).\end{aligned}$$

Im Falle einer injektiven Funktion f haben wir

$$\Delta(X;\, Y) = \Delta(f^{-1} \circ f(X);\ f^{-1} \circ f(Y)) \leq \Delta(f(X);\, f(Y)) \leq \Delta(X;\, Y).$$

Wir beweisen nun (vi). Es seien μ_X, μ_Y und μ_Z die Randverteilungen von X, Y und Z, und $\Omega_X = \Omega_Y$ und Ω_Z die zugehörigen Meßräume.

Da $\mu_X - \mu_Y$ ein endliches signiertes Maß ist, existieren nach [Bau92, Satz 18.1 (Hahn-Zerlegung)] zwei Maße μ^+ und μ^- und zwei disjunkte meßbare Mengen $\Omega^+ \cup \Omega^- = \Omega_X$, so daß $\mu_X - \mu_Y = \mu^+ - \mu^-$ und $\mu^+(\Omega_X \setminus \Omega^+) = \mu^-(\Omega_X \setminus \Omega^-) = 0$.

Für eine meßbare Menge $T \subseteq \Omega_X \times \Omega_Z$ sei $T_x := \{z : (x,z) \in T\}$. Für $\tilde{T} := \Omega^+ \times \Omega_Z$ ist dann

$$\tilde{T}_x = \begin{cases} \Omega_Z, & \text{falls } x \in \Omega^+, \\ \varnothing, & \text{sonst} \end{cases}$$

Dann ist für jedes meßbare $T \subseteq \Omega_X \times \Omega_Z$:

$$\begin{aligned}&\mu_X \times \mu_Z(T) - \mu_Y \times \mu_Z(T) \\ &\quad \overset{(*)}{=} \int \mu_Z(T_x) d\mu_X - \int \mu_Z(T_x) d\mu_Y \\ &\quad = \int_{\Omega^+} \mu_Z(T_x) d\mu^+ - \int_{\Omega^-} \mu_Z(T_x) d\mu^- \\ &\quad \leq \int_{\Omega^+} \mu_Z(\tilde{T}_x) d\mu^+ - \int_{\Omega^-} \mu_Z(\tilde{T}_x) d\mu^- \\ &\quad = \int \mu_Z(\tilde{T}_x) d\mu_X - \int \mu_Z(\tilde{T}_x) d\mu_Y \\ &\quad \overset{(*)}{=} \mu_X \times \mu_Z(\tilde{T}) - \mu_Y \times \mu_Z(\tilde{T}) \\ &\quad = (\mu_X - \mu_Y)(\Omega^+) \cdot \mu_Z(\Omega_Z) \\ &\quad = (\mu_X - \mu_Y)(\Omega^+) \ \leq \ \Delta(X;Y).\end{aligned}$$

Hierbei ist immer x die Laufvariable der Integrale. Die Gleichheiten $(*)$ folgen mit einem Spezialfall des Satzes von Fubini, siehe [Bau92, Satz 23.3].

Damit ist

$$\Delta(X,Z;\ Y,Z) = \max_T\big(\mu_X \times \mu_Z(T) - \mu_Y \times \mu_Z(T)\big) \le \Delta(X;Y).$$

Nach (v) ist aber auch $\Delta(X;Y) \le \Delta(X;Z;\ Y;Z)$, es folgt (vi).

Um (vii) zu zeigen, rechnen wir

$$\begin{aligned}
&\Delta(X|X\in C;\ Y|Y\in C)\\
&\quad= \max_T\big|P(X\in T|X\in C) - P(Y\in T|Y\in C)\big|\\
&\quad= \max_T\left|\frac{P(X\in T\cap C)}{P(X\in C)} - \frac{P(Y\in T\cap C)}{P(Y\in C)}\right|\\
&\quad= \frac{\max_T|P(X\in T\cap C) - P(Y\in T\cap C)|}{P(X\in C)}\\
&\quad\le \frac{\max_T|P(X\in T) - P(Y\in T)|}{P(X\in C)} = \frac{\Delta(X;Y)}{P(X\in C)}.
\end{aligned}$$

Wir zeigen nun (viii). Da wir statt T auch dessen Komplement verwenden können, ist $\Delta(X;Y) = \max_T P(X\in T) - P(Y\in T)$. Definieren wir $T_z := \{a : (a,z)\in T\}$, so erhalten wir

$$\begin{aligned}
\Delta(X,Z;\ Y,Z) &= \max_T P\big((X,Z)\in T\big) - P\big((Y,Z)\in T\big)\\
&= \max_T \sum_z P(X\in T_z \wedge Z = z) - P(Y\in T_z \wedge Z = z)\\
&\overset{(*)}{\le} \sum_z \max_{T_z}\Big(P(X\in T_z \wedge Z = z) - P(Y\in T_z \wedge Z = z)\Big)\\
&= \sum_z P(Z=z)\ \max_{T_z}\Big(P(X\in T_z|Z=z) - P(Y\in T_z|Z=z)\Big)\\
&= \sum_z P(Z=z)\ \Delta(X|Z=z;\ Y|Z=z).
\end{aligned}$$

Da wir zu jeder Familie von Mengen T'_z eine Menge $T := \{(a,z) : a\in T_z\}$ konstruieren können, so daß $T_z = T'_z$, gilt in $(*)$ sogar Gleichheit, und (viii) folgt.

Wir zeigen nun (ix).

Es sei Ω der Wertebereich von X und Y (d. h. eine Menge von Folgen). Dann ist durch $\mu(A) := P(X \in A) - P(Y \in A)$ ein endliches signiertes Maß auf Ω definiert. Es existiert dann eine disjunkte Zerlegung $\Omega = \Omega^+ \cup \Omega^-$, so daß μ auf jeder Teilmenge von Ω^+ positiv, und auf jeder Teilmenge von Ω^- negativ ist (die sog. Hahn-Zerlegung, vgl. [Bau92, Satz 18.1]). Damit können wir zwei Maße μ^+ und μ^- als $\mu^+(A) := \mu(A \cap \Omega^+)$ und $\mu^-(A) := -\mu(A \cap \Omega^-)$ definieren. Beide Maße sind endlich, und $\mu = \mu^+ - \mu^-$. Da μ auf jede Teilmenge von Ω^- negativ ist, ist $\mu(A)$ maximal für $A = \Omega^+$. Wir definieren außerdem $|\mu| := \mu^+ + \mu^-$. Auch $|\mu|$ ist ein endliches Maß, und es gilt $|\mu|(A) \geq |\mu(A)|$ für alle A.

Für eine Menge T von endlichen Folgen der Länge n sei $\hat{T} := \{x \in \Omega : x_{\leq n} \in T\}$, d. h. die $\hat{T}$ sind Mengen von Folgen, die sich durch Betrachten eines endlichen Präfixes erkennen lassen. Es sei $\ell(\hat{T}) := n$. Nach Definition des Produktmeßraums wird die σ-Algebra über Ω gerade von den $\hat{T}$ erzeugt. Weiterhin bilden die $\hat{T}$ eine Algebra $\hat{\mathcal{A}}$.[7] Also existiert nach [Bau92, Satz 5.7 (Approximationseig.)] eine Folge von $\hat{T}_n \in \hat{\mathcal{A}}$, so daß

$$|\mu|(\Omega^+ \triangle \hat{T}_n) \overset{n\to\infty}{\longrightarrow} 0,$$

wobei $A \triangle B$ die symmetrische Differenz $A \setminus B \cup B \setminus A$ bezeichnet. In anderen Worten, die $\hat{T}_n$ approximieren Ω^+ beliebig gut (bzgl. $|\mu|$). Wir können die $\hat{T}_n$ dabei so wählen, daß $\ell(\hat{T}_n) \leq n$, indem wir Elemente in der Folge $\hat{T}_n$ wiederholen. Weiterhin können wir sogar $\ell(\hat{T}_n) = n$ erreichen, da ein $\hat{T}_n$ mit $\ell(\hat{T}_n) < n$ auch als $\hat{T}_n$ mit $\ell(\hat{T}) = n$ aufgefaßt werden kann (man wählt einfach das $\hat{T}_n$ erzeugende T größer).

Es ist aber

$$|\mu|(\Omega^+ \triangle \hat{T}_n) = \mu^+(\Omega^+ \setminus \hat{T}_n) + \mu^+(\hat{T}_n \setminus \Omega^+) + \mu^-(\Omega^+ \setminus \hat{T}_n) + \mu^-(\hat{T}_n \setminus \Omega^+)$$

und da die Summanden auf der rechten Seite alle nichtnegativ sind, streben sie alle gegen 0. Damit streben auch $\mu(\Omega^+ \setminus \hat{T}_n)$ und $\mu(\hat{T}_n \setminus \Omega^+)$ gegen 0. Es folgt somit

$$\lim \mu(\hat{T}_n) = \lim\big(\mu(\Omega^+) - \mu(\Omega^+ \setminus \hat{T}_n) + \mu(\hat{T}_n \setminus \Omega^+) = \mu(\Omega^+)$$

Damit ist

$$\begin{aligned}
\Delta(X_{\leq n}; Y_{\leq n}) &\overset{(*)}{=} \max_T P(X_{\leq n} \in T) - P(Y_{\leq n} \in T) \\
&\overset{(**)}{=} \max_{\ell(\hat{T})=n} P(X \in \hat{T}) - P(Y \in \hat{T}) \\
&= \max_{\ell(\hat{T})=n} \mu(\hat{T}) \geq \mu(\hat{T}_n) \\
&\longrightarrow \mu(\Omega^+) \overset{(***)}{=} \Delta(X; Y)
\end{aligned}$$

[7] Eine Algebra ist ein unter Komplement und *endlichem* Schnitt abgeschlossenes Mengensystem, im Gegensatz zu einer σ-Algebra, die unter Komplement und *abzählbarem* Schnitt abgeschlossen sein muß.

wobei $(*)$ daher rührt, daß ein T mit negativer Differenz der Wahrscheinlichkeiten durch sein Komplement ersetzt werden kann, und $(**)$ rührt daher, daß die $\hat{T}$ mit $\ell(\hat{T}) = n$ nach Konstruktion nur Präfixe der Länge n berücksichtigen. Die Gleichheit $(***)$ benutzt, daß μ auf Ω^+ maximal ist. Damit ist $\lim \Delta(X_{\leq n}; Y_{\leq n}) \geq \Delta(X;Y)$. Nach (v) ist $\Delta(X_{\leq n}; Y_{\leq n}) \leq \Delta(X;Y)$, es folgt die Behauptung (ix).

Nun zeigen wir (x).

Für eine Folge f bezeichne $s(f)$ das kleinste s, so daß für alle $i \geq s$ gilt: $f_i = f_s$. Gibt es kein solches s (konvergiert f also nicht in der diskreten Topologie), so sei $z(f) = \bot$.

Für $n \geq s(f) \neq \bot$ ist $f_n = \lim f_n$, also gilt bei festem i für hinreichend großes n unter der Bedingung $s(X) = i$ immer $X_n = \lim X_\mu$, und damit

$$\lim_n \ \Delta(X_n | s(X) = i;\ \lim X_\mu | s(X) = i) = 0. \tag{1.3}$$

Wir rechnen

$$\begin{aligned}
&\lim_n \Delta(X_n; \lim_\mu X_\mu) \\
&\quad \overset{(v)}{\leq} \lim_n \Delta(X_n, s(X); \lim_\mu X_\mu, s(X)) \\
&\quad \overset{(viii)}{=} \lim_n \sum_{i \in \mathbb{N} \cup \{\bot\}} P(s(X) = i)\ \Delta(X_n | s(X) = i;\ \lim_\mu X_\mu | s(X) = i) \\
&\quad \overset{(*)}{=} \lim_n \sum_{i \in \mathbb{N}} P(s(X) = i)\ \Delta(X_n | s(X) = i;\ \lim_\mu X_\mu | s(X) = i) \\
&\quad = \sum_{i \in \mathbb{N}} P(s(X) = i)\ \lim_n \Delta(X_n | s(X) = i;\ \lim_\mu X_\mu | s(X) = i) \\
&\quad \overset{(1.3)}{=} 0.
\end{aligned}$$

Dabei nutzt $(*)$, daß $P(s(X) = \bot) = 0$, da X fast sicher konvergiert. Man beachte, daß in dieser Formel $\lim_n$ einen Grenzwert in der euklidischen Topologie, $\lim_\mu$ hingegen einen in der diskreten Topologie bezeichnet.

Da $\Delta(X_n; \lim_\mu X_\mu) \geq 0$, folgt $\lim_n \Delta(X_n; \lim_\mu X_\mu) = 0$. □

Die statistische Ununterscheidbarkeit aus Definition 1.2 gibt uns die Möglichkeit, davon zu sprechen, wann zwei Verteilungen ununterscheidbar sind. Allerdings nimmt diese Definition keine Rücksicht auf die Komplexität einer solchen Unterscheidung. Es existieren Verteilungen, die zwar sehr verschieden sind (evtl. sogar disjunkten Träger haben), bei denen es dennoch sehr aufwendig ist, zu berechnen, welche Verteilung vorliegt. In vielen Fällen wollen wir solche Verteilungen

als ununterscheidbar betrachten, dies ermöglicht der Begriff der *komplexitätstheoretischen Ununterscheidbarkeit*, der in der folgenden Definition vorgestellt wird:

Definition 1.4 (Komplexitätstheoretische Ununterscheidbarkeit) Es seien $\{X_{k,z}\}_{k,z}$ und $\{Y_{k,z}\}_{k,z}$ über $k \in \mathbb{N}$ und $z \in Z$ mit $Z \subseteq \Sigma^*$ indizierte Familien von Zufallsvariablen mit Werten in Σ^*. Dann heißen $\{X_{k,z}\}_{k,z}$ und $\{Y_{k,z}\}_{k,z}$ *komplexitätstheoretisch ununterscheidbar*, wenn für jeden polynomiell-beschränkten probabilistischen Algorithmus D (dem *Unterscheider*) eine vernachlässigbare Funktion μ existiert, so daß für alle $k \in \mathbb{N}$ und alle $z \in Z$ gilt:

$$\Big|P\big(D(1^k, z, X_{k,z}) = 1\big) - P\big(D(1^k, z, Y_{k,z}) = 1\big)\Big| \leq \mu(k).$$

Lemma 1.5 (Eigenschaften der komplexitätstheoretischen Ununterscheidbarkeit)

(i) Komplexitätstheoretische Ununterscheidbarkeit ist transitiv, d. h. sind $\{X_{k,z}\}_{k,z}$ und $\{Y_{k,z}\}_{k,z}$ komplexitätstheoretisch ununterscheidbar, sowie $\{Y_{k,z}\}_{k,z}$ und $\{Z_{k,z}\}_{k,z}$, so sind auch $\{X_{k,z}\}_{k,z}$ und $\{Z_{k,z}\}_{k,z}$ komplexitätstheoretisch ununterscheidbar.

(ii) Statistische Ununterscheidbarkeit impliziert komplexitätstheoretische, d. h. sind $\{X_{k,z}\}_{k,z}$ und $\{Y_{k,z}\}_{k,z}$ statistisch ununterscheidbar, so sind sie auch komplexitätstheoretisch ununterscheidbar.

(iii) Bei Familien von Zufallsvariablen mit endlichem Träger fallen statistische und komplexitätstheoretische Ununterscheidbarkeit zusammen: Wir sagen, eine Familie von Zufallsvariablen $\{X_{k,z}\}_{k,z}$ habe endlichen Träger, wenn ein M existiert, so daß $P(X_{k,z} \in M) = 1$ für alle k, z. Haben also $\{X_{k,z}\}_{k,z}$ und $\{Y_{k,z}\}_{k,z}$ endlichen Träger, so sind $\{X_{k,z}\}_{k,z}$ und $\{Y_{k,z}\}_{k,z}$ genau dann statistisch ununterscheidbar, wenn sie komplexitätstheoretisch ununterscheidbar sind.

Beweis: Wir zeigen zunächst (i). Wir nehmen an, $\{X_{k,z}\}_{k,z}$ und $\{Z_{k,z}\}_{k,z}$ seien nicht komplexitätstheoretisch ununterscheidbar. Dann existieren ein polynomi-

ell-beschränkter probabilistischer Algorithmus D und eine Folge $z_k \in Z$, so daß

$$\begin{aligned}\big|P(D(k, z_k, X_{k,z_k}) = 1) - P(D(k, z_k, Y_{k,z_k}) = 1)\big| \\ + \big|P(D(k, z_k, Y_{k,z_k}) = 1) - P(D(k, z_k, Z_{k,z_k}) = 1)\big| \\ \geq \big|P(D(k, z_k, X_{k,z_k}) = 1) - P(D(k, z_k, Z_{k,z_k}) = 1)\big|\end{aligned}$$

nicht vernachlässigbar ist. Damit ist aber auch einer der Summanden nicht vernachlässigbar, und somit $\{X_{k,z}\}_{k,z}$ und $\{Y_{k,z}\}_{k,z}$ komplexitätstheoretisch unterscheidbar oder $\{Y_{k,z}\}_{k,z}$ und $\{Z_{k,z}\}_{k,z}$ komplexitätstheoretisch unterscheidbar.

Nun beweisen wir (ii). Wir nehmen an $\{X_{k,z}\}$ und $\{Y_{k,z}\}$ seien komplexitätstheoretisch unterscheidbar. Dann existiert ein polynomieller probabilistischer Algorithmus D, so daß $\{D(k, z, X_{k,z})\}$ und $\{D(k, z, Y_{k,z})\}$ statistisch unterscheidbar sind. Aber $D(k, z, X_{k,z})$ kann als $d(k, z, X_{k,z}, R)$ mit einer deterministischen Funktion d und einer von $X_{k,z}$ unabhängigen Zufallsvariable R (dem Zufallsband von D) dargestellt werden, und $D(k, z, Y_{k,z})$ analog; wir haben somit mit Lemma 1.3:

$$\begin{aligned}\Delta(D(k, z, X_{k,z});\ D(k, z, Y_{k,z})) \\ \overset{1.3\,(v)}{\leq} \Delta(X_{k,z}, R;\ Y_{k,z}, R) \\ \overset{1.3\,(vi)}{\leq} \Delta(X_{k,z};\ Y_{k,z}).\end{aligned}$$

Da $\{X_{k,z}\}$ und $\{Y_{k,z}\}$ nach Annahme statistisch ununterscheidbar sind, müssen es somit auch $D(k, z, X_{k,z})$ und $D(k, z, Y_{k,z})$ sein, es liegt somit ein Widerspruch vor, und (ii) ist bewiesen.

Wir beweisen nun (iii). Wegen (ii) genügt es zu zeigen, daß aus der komplexitätstheoretischen Ununterscheidbarkeit die statistische folgt. Wir nehmen dazu an, $\{X_{k,z}\}$ und $\{Y_{k,z}\}$ seien statistisch unterscheidbar. Dann existieren $z_k \in Z$, Mengen $T_k \subseteq M$, ein Polynom p und eine unendliche Menge $K \subseteq \mathbb{N}$, so daß für alle $k \in K$:

$$\big|P(X_{k,z_k} \in T_k) - P(Y_{k,z_k} \in T_k)\big| = \Delta(X_{k,z_k};\, Y_{k,z_k}) > \frac{1}{p(k)}.$$

Da nur endlich viele verschiedene Teilmengen T_k der endlichen Menge M existieren, gibt es eine Menge $T \subseteq M$ und eine unendliche Menge $K' \subseteq K$, so daß $T = T_k$ für alle $k \in K'$. Da T endlich ist, existiert ein polynomiell-beschränkter probabilistischer Algorithmus D, so daß $D(k, z, x) = 1$ gdw. $x \in T$. Somit ist auch

$$\big|P(D(k, z_k, X_{k,z_k}) = 1) - P(D(k, z_k, X_{k,z_k}) = 1)\big| > \frac{1}{p(k)}$$

für alle $k \in K'$. Da K' unendlich ist, folgt damit die komplexitätstheoretische Unterscheidbarkeit von $\{X_{k,z}\}$ und $\{Y_{k,z}\}$. □

Nun präsentieren wir noch vier gängige kryptographische Komplexitätsannahmen. Die erste ist die Existenz von Einweg-Funktionen, eine der schwächsten in der Kryptographie verwandten Annahmen, die aber trotzdem bereits überraschend viele Konstruktionen ermöglicht.

Definition 1.6 (Einweg-Funktion)
Eine Funktion f ist eine *Einweg-Funktion*, wenn folgende Bedingungen gegeben sind:

- Die Funktion f ist effizient berechenbar (in deterministischer polynomieller Zeit).
- Die Funktion f ist schwer zu invertieren, d. h. für jeden nichtuniformen polynomiell-beschränkten probabilistischen Algorithmus A ist

$$P\Big(x \leftarrow U_k,\ x' \leftarrow A(1^k, f(x)) : f(x) = f(x')\Big)$$

vernachlässigbar.

Mehr Details finden sich in [Gol01, Abschnitt 2.2] unter dem Namen der *non-uniformly strong one-way functions*.

Definition 1.7 (Kollisionsresistente Hashfunktionen)
Eine über den Sicherheitsparameter k parametrisierte Funktion $f_k : \Sigma^* \to \Sigma^{l(k)}$ ist eine *kollisionsresistente Hashfunktion*, wenn folgende Bedingungen gegeben sind:

- Der Wert $f_k(x)$ ist effizient berechenbar (in deterministischer polynomieller Zeit in k und der Länge der Eingabe x).
- Es ist schwer, eine Kollision zu finden, d. h. für jeden *uniformen* polynomiell-beschränkten probabilistischen Algorithmus A ist

$$P\Big((x, x') \leftarrow A(1^k) : x \neq x' \text{ und } f_k(x) = f_k(x')\Big)$$

vernachlässigbar in k.

Man beachte, daß (im Gegensatz zu den anderen Komplexitätsannahmen in diesem Abschnitt) hier der Angreifer *uniform* ist. Dies liegt daran, daß ein nichtuniformer Angreifer diese Annahme trivialerweise brechen kann: Da f_k beschränkte

Länge $l(k)$ aber einen unbeschränkten Urbildbereich hat, existiert immer eine Kollision, und der Auxiliary input kann diese enthalten.

Wir geben noch zwei weitere Komplexitätsannahmen an. Diese sind hier nur der Vollständigkeit halber gegeben, da wir diese Annahmen nicht direkt verwenden werden. Sie kommen jedoch als Voraussetzung von zwei Sätzen vor, die wir aus der Literatur übernehmen (Sätze 5.6 und 6.7). Wir skizzieren die Definitionen nur grob und verweisen für Details auf die Literatur.

Definition 1.8 (Kollisionsresistente Familien von Hashfunktionen (Skizze))
Eine *kollisionsresistente Familie von Hashfunktionen* ist eine Familie $\{h_i\}_{i \in I}$ von Funktionen zusammen mit einem effizienten Algorithmus I (dem Indexgenerator), so daß gilt:

- Die Funktion h_i ist effizient berechenbar.
- Für jeden nichtuniformen polynomiell-beschränkten probabilistischen Algorithmus A (dem Angreifer) gilt, daß wenn i vom Instanzgenerator I gewählt wurde, A nur mit vernachlässigbarer Wahrscheinlichkeit eine Kollision (x, x') ausgibt, d. h. nur mit vernachlässigbarer Wahrscheinlichkeit ist $h_i(x) = h_i(x')$, aber $x \neq x'$.

Details zur Definition von kollisionsresistenten Familien von Hashfunktionen finden sich z. B. in [Gol04] unter dem Namen *collision-free hashing functions*. Man beachte aber, daß dort der uniforme Fall betrachtet wird, d. h. der Angreifer ist dort ein uniformer Algorithmus, wohingegen wir nichtuniforme Angreifer zulassen.

Definition 1.9 (Trapdoor permutations (Skizze))
Eine *Familie von Trapdoor permutations* ist eine Familie $\{f_{pk}\}_{pk \in I}$ zusammen mit einem polynomiell-beschränkten probabilistischen Algorithmus G, so daß die folgenden Eigenschaften gelten:

- Die Funktion f_{pk} ist effizient berechenbar.
- Für von G gewähltes pk ist f_{pk} schwer zu invertieren (vgl. Definition 1.6).
- Der Algorithmus G generiert zusätzlich zu pk (dem Public-Key) noch einen Secret-Key sk. Unter Kenntnis des Secret-Keys sk ist f_{pk} effizient invertierbar.

(Fortsetzung nächste Seite)

(Fortsetzung)

Wir sprechen von *Enhanced trapdoor permutations*, wenn die zusätzliche Eigenschaft erfüllt ist, daß es effizient möglich ist, ein (ungefähr gleichverteiltes) Bild von f_{pk} so zu wählen, daß – selbst wenn der bei dieser Wahl verwendete Zufall dem Angreifer bekannt ist – es schwierig ist, f_{pk} an dieser Stelle zu invertieren.

Details zur Definition von Trapdoor permutations finden sich z. B. in [Gol01, Abschnitt 2.4.4] und zu Enhanced trapdoor permutations in [Gol04, Appendix C.1]. Man beachte aber, daß dort der uniforme Fall betrachtet wird, wohingegen wir nichtuniforme Angreifer zulassen.

2. Simulationsbasierte Sicherheit

In diesem Kapitel geben wir einen kurzen Überblick über die Geschichte der simulationsbasierten Sicherheitsmodelle, um danach die Definitionen und Eigenschaften der in unserer Arbeit verwendeten Sicherheitsmodelle vorzustellen.

2.1. Die Geschichte der simulationsbasierten Sicherheitsmodelle

2.1.1. Zero-Knowledge

Der wohl erste Sicherheitsbegriff, der auf der Idee der Simulation basiert, ist der Begriff des *Zero-Knowledge-Beweises* [GMR85]. Unter einem Zero-Knowledge-Beweis (im folgenden kurz *ZK-Beweis*) versteht man ein Protokoll, das zwei sich scheinbar widersprechende Eigenschaften vereint: (a) eine Partei P (*Prover* genannt) überzeugt eine andere Partei V (den *Verifier*) von der Wahrheit einer Aussage, für die P einen Beweis kennt, und (b) der Verifier V erfährt außer der Korrektheit der Aussage nichts. Insbesondere lernt der Verifier nicht den dem Prover bekannten Beweis, und ist nach Protokollende nicht einmal in der Lage, eine dritte Partei von der Korrektheit der Aussage zu überzeugen. Überraschenderweise lassen sich solche Protokolle für sehr allgemeine Klassen von Aussagen konstruieren [GMW86, BOGG$^+$90].

Die Eigenschaft (a), ein Beweis zu sein, ist einfach zu definieren. Vereinfacht gesagt wird verlangt, daß selbst bei unehrlichem Prover die Wahrscheinlichkeit dafür vernachlässigbar klein ist, daß der Verifier die Aussage akzeptiert, obwohl sie falsch ist (die sog. *Soundness*-Bedingung). Weiter fordert man noch, daß beim Zusammenspiel von ehrlichem Prover und Verifier eine korrekte Aussage mit hoher Wahrscheinlichkeit akzeptiert wird (die sog. *Completeness*- oder *Vollständigkeits*-Bedingung).

Schwieriger ist es, die Bedingung (b) zu fassen, daß der Verifier nichts lernt (die Zero-Knowledge-Eigenschaft). Der naive Ansatz wäre, das Wissen des Verifiers informationstheoretisch zu fassen, und z. B. zu verlangen, daß die Information, die der Verifier über die Eingaben des Provers hat, im Verlauf des Protokolls nur in vernachlässigbarem Maße ansteigt. Diese Definition hat aber den großen Nachteil, daß sie keinen komplexitätstheoretischen Wissensgewinn berücksichtigt. Dies sei an einem Beispiel erläutert: Wir nehmen an, der Verifier kenne das Chiffrat eines geheimen, dem Prover bekannten Textes p, aber nicht den Text

selbst. Wenn nun im Verlauf des Protokolls p offengelegt wird, hat der Verifier im komplexitätstheoretischen Sinne klar etwas gelernt. Da aber informationstheoretisch das Chiffrat bereits alle Information über den Klartext p enthält (unter der Annahme, daß der verwendete öffentliche Schlüssel bekannt ist), steigt die Information über p nicht an. Unser Ansatz fängt also diesen Fall nicht ab.

Daher fand man einen Ansatz, der sich grob in folgendem Satz zusammenfassen läßt: „Der Verifier lernt nichts, wenn er alles, was er im Laufe des Protokolls erfährt, sich auch selbst hätte ausdenken können." Formaler fassen wir das wie folgt: Für jeden (auch einen unehrlichen) polynomiell-beschränkten Verifier gibt es eine polynomiell-beschränkte Maschine, den sog. *Simulator*, die folgendes leistet: Die Ausgabe des Simulators ist, als Wahrscheinlichkeitsverteilung aufgefaßt, gleich oder zumindest ununterscheidbar von einer Aufzeichnung der Kommunikation zwischen Prover und Verifier. Dies faßt obige intuitive Definition von „nichts lernen", da die Existenz des Simulators zeigt, daß der Verifier tatsächlich nichts erfährt, was man sich nicht auch ohne die dem Prover vorliegenden geheimen Informationen hätte ausdenken können.

Die im vorangegangenen Abschnitt beschriebene und erstmals in [GMR85] vorgestellte simulationsbasierte Definition von ZK-Beweisen stellte sich als ein mächtiges Werkzeug in der Protokollentwicklung heraus. Wenn in einem Protokoll garantiert werden soll, daß die von einer Partei gesandten Daten gewissen Bedingungen genügen, aber ein Offenlegen der zum Beweis nötigen geheimen Daten nicht in Frage kommt, bietet sich der ZK-Beweis als universelles (aber leider oft nicht sehr effizientes) Mittel an. So wurde zum Beispiel in [GMW87] ein Verfahren zur sicheren Auswertung von beliebigen Funktionen vorgestellt, in dem ZK-Beweise eine zentrale Rolle spielen.

Da der ZK-Beweis in den meisten Fällen als Teil eines größeren Protokolls verwendet wird, und da in diesem Fall oft nicht nur ein, sondern viele Beweise geführt werden, ist man natürlich an der folgenden Eigenschaft interessiert: Führt man mehrere ZK-Beweise durch (etwa mehrere Beweise der gleichen Tatsache an verschiedene Verifier, oder Beweise verschiedener Aussagen an den gleichen Verifier), so sollten die Verifier immer noch nichts lernen. In anderen Worten sollte das resultierende Protokoll (die *Komposition* der einzelnen ZK-Beweise) als ganzes immer noch die ZK-Eigenschaft haben. Obwohl es selbstverständlich scheint, daß der Verifier, wenn er in jedem einzelnen Protokoll nichts lernt, er auch insgesamt nichts lernt, hat sich herausgestellt, daß dies keineswegs selbstverständlich ist. Wir unterscheiden hier zwischen zwei Typen der Komposition: *sequentielle* und *parallele*. Unter sequentieller Komposition verstehen wir, daß die verschiedenen ZK-Beweise nacheinander ausgeführt werden, d. h. der nächste Beweis wird erst begonnen, wenn die letzte Nachricht des vorausgegangenen gesandt wurde. Dahingegen werden bei der parallelen die ZK-Beweise gleichzeitig ausgeführt.

In [GK96b] wurde gezeigt, daß die ursprüngliche Definition von ZK nicht

einmal sequentielle Komposition zuläßt. Allerdings war zu dieser Zeit bereits Abhilfe bekannt: In [GO94] war gezeigt worden, daß eine leicht strengere Definition von ZK ausreicht, um sequentielle Komposition zu garantieren. Diese Definition führt einen sogenannten *Auxiliary input* ein. Dies ist eine zusätzliche Eingabe, die sowohl dem Verifier als auch dem Simulator gegeben wird. Ist die Simulation nun auch noch unter Berücksichtigung dieser Zusatzeingabe vom realen Protokollauf ununterscheidbar, so ist die strengere Definition erfüllt, und sequentielle Komposition ist möglich. Erfreulicherweise sind fast alle bekannten ZK-Beweise auch noch bezüglich dieser stärkeren Forderung sicher.

Schwieriger ist der Fall der parallelen Komposition. [GK96b] zeigten, daß beide Definitionen (sowohl mit als auch ohne Auxiliary input) im allgemeinen keine parallele Komposition erlauben. Anders als im Falle der sequentiellen Komposition gab es nicht nur künstliche Gegenbeispiele, vielmehr waren viele der damals bekannten ZK-Protokolle nicht parallel komponierbar. Darüber hinaus zeigten [GK96b] die Unmöglichkeit gewisser natürlicher Klassen parallel komponierbarer ZK-Beweise. Vereinfacht dargestellt, ist der Hauptgrund dafür, warum ein ZK-Protokoll nicht parallel komponiert, der folgende: Der Verifier kann bei mehreren parallelen Protokollausführungen seine Nachrichten für *jede* Protokollinstanz in Abhängigkeit der Nachrichten des Provers in *allen* Protokollinstanzen wählen. Dies führt zu einem Protokollauf, bei dem die Nachrichten in einer komplexen Relation stehen, die vom Simulator nicht effizient reproduziert werden kann. Diese Verzahnung der Protokolläufe muß ein parallel komponierbares ZK-Protokoll [FS90, GK96a] verhindern. In [GK96a] wird dies z. B. dadurch erreicht, daß der Verifier gezwungen wird, sich schon zu Protokollbeginn auf seine Eingaben festzulegen, diese aber erst später dem Prover gegenüber aufdeckt (dies wird durch sog. *Commitment*-Verfahren ermöglicht).

Noch schwieriger wird die Situation, wenn wir sog. *nebenläufige* Komposition zulassen. Hierbei handelt es sich eine Verallgemeinerung der parallelen Komposition, bei der wir nicht mehr annehmen, daß die einzelnen Protokollinstanzen synchronisiert ablaufen, sondern erlauben, daß die Instanzen in beliebiger Weise verzahnt werden. (So kann z. B. die erste Nachricht der einen Protokollinstanz gesandt werden, dann die komplette zweite Protokollinstanz durchgeführt, und dann erst die erste beendet werden.) Diese zusätzliche Allgemeinheit führt dazu, daß noch größere Klassen von ZK-Protokollen nicht nebenläufig komponieren [KPR98], und erst in [DNS98, RK99] wurden Lösungen für dieses Szenario vorgestellt.

Für einen weitergehenden Überblick über Zero-Knowledge und die damit verbundenen Probleme, Techniken und Ergebnisse empfehlen wir dem geneigten Leser [Gol02] oder [Gol01, Kapitel 4].

2.1.2. Sichere Funktionsauswertungen

Obgleich ursprünglich der simulationsbasierte Ansatz bei Zero-Knowledge-Beweisen dazu diente, die Geheimhaltung zu modellieren, während die Korrektheit (in Falle von ZK-Beweisen die Eigenschaft, in der Tat ein Beweis zu sein) getrennt definiert wurde, stellte es sich heraus, daß dieser Ansatz weitaus allgemeinere Anwendung finden kann. Das wichtigste Beispiel ist hier die *sichere Funktionsauswertung*. Hierunter versteht man ein Protokoll, bei dem jede beteiligte Partei eine private (geheime) Eingabe x_i besitzt, und die Parteien gemeinsam den Funktionswert $f(x_1, \ldots, x_n)$ berechnen wollen, wobei f eine feste, allen bekannte (und möglicherweise probabilistische) Funktion ist. Hierbei soll garantiert sein, daß selbst wenn einige Parteien unehrliches Verhalten zeigen (*korrumpiert* sind), das Ergebnis der Funktionsauswertung korrekt ist, und daß keine Partei mehr über die Eingaben der anderen Parteien lernt, als sie aus ihren eigenen Eingaben und dem Ergebnis der Funktionsauswertung erschließen könnte.

Die frühen Definitionen von sicheren Funktionsauswertungen (mit [Yao82] beginnend) forderten, daß ein sicheres Protokoll zwei Eigenschaften erfüllt: *Korrektheit* und *Geheimhaltung*. Unter Korrektheit verstand man in etwa das folgende: Wenn die Funktionsauswertung bei Eingaben x_i das Ergebnis $\tilde{f}$ liefert, so existieren Eingaben $\tilde{x}_i$, so daß $\tilde{f} = f(\tilde{x}_1, \ldots, \tilde{x}_n)$, wobei für unkorrumpierte Parteien $\tilde{x}_i = x_i$ sein muß, d. h. die korrumpierten Parteien dürfen ihre Eingaben frei wählen, während die unkorrumpierten die ihnen vorgegebenen verwenden. Unter der Geheimhaltung wiederum versteht man in etwa, daß keine (Gruppe von) unkorrumpierten Parteien etwas über die Eingabe der ehrlichen Parteien lernt, was sie nicht auch bei passender Wahl der $\tilde{x}_i$ aus $f(\tilde{x}_1, \ldots, \tilde{x}_n)$ hätten erschließen können, wobei wieder $\tilde{x}_i = x_i$ für die unkorrumpierten Parteien. (Die Ausführung des Protokolls bezeichnen wir oft als *reale Funktionsauswertung*, während die Berechnung von $f(\tilde{x}_1, \ldots, \tilde{x}_n)$ die *ideale Funktionsauswertung* genannt wird.)

Obwohl diese beiden Eigenschaften die intuitiven Forderungen an die Sicherheit des Protokolls widerzuspiegeln scheinen, stellte es sich heraus, daß eine separate Definition von Korrektheit und Geheimhaltung nicht in allen Fällen in einem Sicherheitsbegriff resultiert, der unserer Intuition von Sicherheit entspricht. So wurde z. B. von [MR91] das folgende Beispiel gefunden:

Wir betrachten ein Zwei-Parteien-Protokoll zur Auswertung der logischen Konjunktion, d. h. zwei Parteien Alice und Bob bekommen je ein Bit a bzw. b als Eingabe, und am Ende soll $f(a, b) := a \wedge b$ berechnet werden. Das Protokoll verfährt wie folgt: Alice sendet ihr Bit an Bob, und Bob berechnet $a \wedge b$ und gibt das Ergebnis aus. Alice bricht das Protokoll dann und nur dann ab, wenn $a = 0$ und das von Bob verkündete Ergebnis 1 ist. Dieses Protokoll ist offensichtlich intuitiv unsicher, denn Bob erfährt immer Alices Eingabe *und* kann das

Endergebnis frei wählen (wenn $a = 1$). Doch formal erfüllt dieses Protokoll die Korrektheitsbedingung, denn das Protokollergebnis 0 kann Bob durch Eingabe $b = 0$ auch bei einer idealen Funktionsauswertung erzwingen, und das Ergebnis 1 bei $a = 1$ durch Eingabe $b = 1$. Weiterhin ist formal auch die Geheimhaltungsbedingung gegeben, denn wenn Bob ideal $b = 1$ setzt, so erfährt er immer Alices Eingabe, da $f(a, 1) = a$.

Wir erkennen an diesem Beispiel, daß die Forderung nach sowohl Korrektheit als auch Geheimhaltung nicht hinreichend ist. Das Problem liegt darin, daß wir zugelassen haben, daß die korrumpierten Parteien (hier Bob) durch eine geeignete ideale Eingabe das reale Funktionsergebnis ideal imitieren können (hier $b =$ gewünschtes Ergebnis), und durch eine geeignete ideale Eingabe (hier $b = 1$) den realen Informationsgewinn auch ideal erreichen können. Nur ist durch die getrennte Definition nicht garantiert, daß beide Phänomene durch die *gleiche* Eingabe realisiert werden. Es ist also notwendig, eine vereinheitlichte Definition von Korrektheit und Geheimhaltung zu finden.

Hier zeigt der Simulationsansatz seine Stärke. In [GL91, MR91, Bea92, Can95, Can00] wurden verschiedene Sicherheitsbegriffe für sichere Funktionsauswertungen vorgestellt, die die Idee der Simulation verwenden, um eine einheitliche Sicht auf Korrektheit und Geheimhaltung zu erlauben. Der Ansatz, wie er in [Bea92, Can95, Can00] verfolgt wird, ist vereinfacht der folgende: Wir betrachten zwei Spiele, das *reale* und das *ideale Modell*. Im realen Modell wird das zu untersuchende Protokoll π ausgeführt, wobei ein Angreifer eine Anzahl von Parteien korrumpieren und die Ein- und Ausgaben der korrumpierten Parteien lernen und modifizieren darf. Außerdem – und das ist der wesentliche Unterschied zum idealen Modell – darf der Angreifer das Verhalten der korrumpierten Parteien kontrollieren, d. h. er erhält alle an sie gesandten Nachrichten und darf in ihrem Namen Nachrichten verschicken. Nach Protokollende gibt der Angreifer eine Ausgabe, die o. B. d. A. alle Informationen enthält, die der Angreifer im Protokollverlauf gelernt hat.

Im idealen Modell hingegen gibt es statt eines Angreifers einen Simulator, der zwar auch Parteien korrumpieren und ihre Ein-/Ausgabe kontrollieren darf, aber die Ausgaben der ehrlichen Parteien werden durch eine ideale (d. h. vom Simulator nicht beeinflußbare) Auswertung der Funktion f bestimmt, die als Argumente die Eingaben der ehrlichen Parteien und die vom Simulator modifizierten Eingaben der korrumpierten Parteien erhält. Auch der Simulator gibt eine Ausgabe, bei der er versucht, die Ausgabe des Angreifers so genau wie möglich nachzuahmen.

Wir sagen dann, das Protokoll π sei eine sichere Funktionsauswertung von f, wenn es für jeden Angreifer einen Simulator gibt, so daß für beliebige Partei-Eingaben die Ausgabe der ehrlichen Parteien und des Angreifers im realen Modell (als *eine* Zufallsvariable aufgefaßt) ununterscheidbar ist von der Ausgabe des Simulators und der ehrlichen Parteien im idealen Modell.

Wir erkennen nun, daß dieser Sicherheitsbegriff sowohl Korrektheit als auch Geheimhaltung impliziert. Die Korrektheit ergibt sich daraus, daß der Simulator es schaffen muß, daß die ehrlichen Parteiausgaben im idealen Modell denen im realen entsprechen. Da der Simulator aber ideal keine inkorrekten Ausgaben erzwingen kann, ist ihm das nur möglich, wenn auch im realen bereits Korrektheit gegeben ist. Analog muß der Simulator die Ausgabe des Angreifers nachahmen, dazu muß er in gewissem Sinne „ebensoviel wissen wie der Angreifer." Wieder ist dies nur möglich, wenn der Angreifer nichts lernt, was man nicht auch im Idealfall lernen kann. Aber wir haben mehr als nur jeweils für sich Korrektheit und Geheimhaltung, da nun *ein* Simulator *gleichzeitig* die Ausgaben der unkorrumpierten Parteien und das Wissen des Angreifers simulieren muß. Das schließt das obige Beispiel (und andere in [MR91] genannte) aus, da dort ein Simulator für die Ausgaben der unkorrumpierten Parteien und einer für die Ausgabe des Angreifers verschiedene Werte für b wählen müßten.

Die Erklärungen der letzten Absätze illustrieren das zentrale Paradigma simulationsbasierter Sicherheitsmodelle: *Weil im idealen Modell nichts „Schlimmes" passieren kann (per Definition), und weil im idealen Modell alles passiert, was im realen passiert, kann folglich auch im realen Modell nicht „Schlimmes" passieren.* Diese Grundmotivation zieht sich, mit verschiedenen konkreten Interpretationen von „schlimm", durch alle simulationsbasierten Sicherheitsdefinitionen.

Bereits [MR91, Bea92] haben erkannt, daß solche simulationsbasierten Definitionen von sicheren Funktionsauswertungen sequentielle Komposition ermöglichen. Dabei versteht man unter sequentieller Komposition von Funktionsauswertungen natürlich nicht einfach die Hintereinanderausführung verschiedener unabhängiger Berechnungen, sondern erlaubt, daß die Eingaben späterer Berechnungen von den Ausgaben (d. h. Ergebnissen) der vorausgegangenen abhängen. Die parallele Komponierbarkeit jedoch ist—je nach Modell—durch Gegenbeispiele widerlegbar oder zumindest unbewiesen.

Interessanterweise ergeben sich manche Varianten von ZK-Beweisen nun als Spezialfälle der sicheren Funktionsauswertung. Die zugehörige Funktion ist eine, bei der der Prover einen Beweis für die Behauptung eingibt, und die Ausgabe der Funktion ist ein Bit, welches angibt, ob der Beweis korrekt ist. Eine sichere Funktionsauswertung für diese Funktion ist ein Beweis wegen der Korrektheit, und hat die Zero-Knowledge-Eigenschaft wegen der simulationsbasierten Definition der Geheimhaltung.

2.1.3. Reaktive Funktionalitäten

Im vorangegangenen Abschnitt haben wir gesehen, daß der simulationsbasierte Ansatz eine vorzügliche (wenn nicht sogar die einzige praktikable) Möglichkeit darstellt, um allgemein sichere Funktionsauswertungen zu definieren. Eine solche allgemeine Definition ist aus dem Grunde sehr wichtig, daß ansonsten für

jede Anwendung eine spezielle Definition gefunden werden müßte. Diese könnte zwar—je nach Anwendung—einfacher ausfallen, doch bergen anwendungsspezifische Definitionen eine gewisse Gefahr, denn es ist, selbst für den erfahrenen Kryptologen, nicht immer einfach, genau zu erkennen, inwiefern eine Definition tatsächlich den intuitiven und praktischen Anforderungen an die Sicherheit entspricht. Dies erkennt man z. B. an der im vorangegangen Abschnitt diskutierten Kritik am Modell von [Yao82], oder an der langen Geschichte der Sicherheitsbegriffe für Public-Key-Kryptosysteme (vgl. [BDPR98] für einen Überblick), in der immer striktere Definitionen eingeführt wurden, um neue Angriffsszenarien abzufangen. Diese Problematik besteht natürlich nicht nur bei anwendungsspezifischen Sicherheitsbegriffen, sondern ebenso bei allgemeinen, doch ist zu hoffen, daß im letzteren Falle deren Probleme besser erforscht werden, einfach da sich mehr Forscher mit diesen allgemeinen Modellen beschäftigen.

Ein weiterer Vorteil allgemeiner Definitionen ist es, daß es mit diesen möglich ist, Fragen zur Möglichkeit und Unmöglichkeit bestimmter Konstruktionen unter verschiedenen Komplexitätsannahmen in allgemeinster Weise zu erforschen (z. B. [GMW87, CCD88, BOGW88]).

Nun können zwar viele kryptographische Aufgabenstellungen als sichere Funktionsauswertungen aufgefaßt werden (so ist z. B. ein sicherer Münzwurf (*coin toss*) eine probabilistische Funktion ohne Eingaben, und auch eine elektronische Wahl kann als eine Funktionsauswertung gesehen werden), doch gibt es auch Anwendungen, die *reaktiv* sind. Darunter verstehen wir Anwendungen, bei denen nicht nur zu Beginn und Ende des Protokolls Ein- bzw. Ausgaben möglich sind, sondern zu beliebigen Zeitpunkten während des Protokollaufs. Dabei können sogar einige Eingaben von vorausgegangenen Ausgaben abhängen. Eines der einfachsten Beispiele für solch ein reaktives Protokoll ist das *Commitment*. Bei einem Commitment unterscheiden wir zwei Phasen. In der ersten Phase (der *Commit-Phase*) kann eine Partei (der Sender) einen Wert m festlegen (erste Eingabe), voraufhin die zweite Partei (der Empfänger) erfährt, daß der Wert festliegt, ohne den Wert selbst zu erfahren (erste Ausgabe). Zu einem späteren Zeitpunkt kann der Sender das Aufdecken (*unveil*) des Wertes m veranlassen (zweite Eingabe), woraufhin der Empfänger m erfährt (zweite Ausgabe). Ein solches Verfahren ermöglicht es also, daß der Sender sich auf einen bestimmten Wert festlegt, so daß der Empfänger sicher sein kann, daß der Sender diesen Wert nicht mehr ändern kann. Dabei wird aber garantiert, daß der Wert geheim bleibt, solange der Sender dies möchte. Würden wir nun versuchen, ein solches Commitment-Verfahren als sichere Funktionsauswertung zu sehen, würden der erste und zweite Input zeitlich nicht mehr getrennt sein, und auch die erste und zweite Ausgabe würde der Empfänger erst zu Protokollende erhalten. Das Commitment wäre zu einer einfachen Nachrichtenübermittlung entartet.

Um also solche reaktiven Protokollaufgaben in uniformer Weise zu spezifizieren, können wir keine Funktionen verwenden, sondern müssen uns allgemeine-

rer Konstruktionen bedienen, sog. *reaktiver Funktionalitäten* (im folgenden kurz *Funktionalität*). Diese können wir uns einfach als interaktive Maschinen vorstellen, die jederzeit Eingaben von Parteien empfangen und Ausgaben verschicken können. Dabei nehmen wir an, ganz im Sinne einer idealen Modellierung, daß die Kommunikation zwischen Parteien und der Funktionalität nicht abhör- oder modifizierbar ist. Eine Funktion ist dann ein Spezialfall einer Funktionalität, die erst dann Ausgaben gibt, wenn sie alle Eingaben erhalten hat. Auch das Commitment läßt sich nun leicht modellieren, es handelt sich um eine Maschine, die einen vom Sender eingegebenen Wert speichert und dem Empfänger mitteilt, daß der Wert gespeichert wurde. Schickt der Sender die Aufforderung zum Aufdecken an die Funktionalität, so übermittelt diese den zuvor gespeicherten Wert an den Empfänger. (Beispiele für die Definition verschiedener Funktionalitäten, darunter Commitment, finden sich in [Can01]).

Es liegt nun also nahe, zur allgemeinen Definition sicherer Protokolle den Simulationsansatz zu verwenden, indem man in den im vorangegangenen Absatz beschriebenen Konstruktionen einfach im idealen Modell eine ideale Funktionalität statt einer Funktion einsetzt. Allerdings führt dies noch nicht zu einem zufriedenstellenden Sicherheitsmodell, da in obigen Sicherheitsmodellen alle Eingaben von vornerein feststehen und nicht von den Ausgaben der Funktionalität abhängen. So könnte ein Protokoll bzgl. einer solchen Definition als sicher gelten, obwohl es möglich ist, das Protokoll zu brechen, wenn man manche Eingaben in geschickter Weise von vorangegangenen Ausgaben abhängen läßt. Um dies abzufangen, wurde in [PW01, Can01] eine weitere Entität eingeführt, die sogenannte *Umgebung*. Diese interagiert mit dem Protokoll im realen Modell sowie mit der Funktionalität im idealen. Dabei lernt sie die Ausgaben des Protokolls/der Funktionalität und kann die Eingaben in Abhängigkeit davon wählen. Ist es der Umgebung nun unmöglich zu unterscheiden, ob das reale oder das ideale Modell vorliegt (selbst bei Interaktion mit dem Angreifer bzw. dem Simulator), so betrachten wir ideales und reales Modell als ununterscheidbar und nennen das Protokoll sicher (dies wird im Verlauf dieses Kapitels noch wesentlich detailierter beschrieben). Gegenüber den Sicherheitsdefinitionen für sichere Funktionsauswertungen hat sich nun geändert, daß wir die Wahl der Eingaben und die Entscheidung, ob die Ausgaben von Protokoll und Angreifer/Simulator im realen wie idealen ununterscheidbar sind, nun einer einzigen Entität, der Umgebung überlassen.

Es stellte sich heraus, daß diese Sicherheitsmodelle mit Umgebung nicht nur in der Lage sind, die Sicherheit von reaktiven Protokollen zu fassen, sondern daß darüber hinaus sehr mächtige Kompositionstheoreme gelten. Man betrachte hierzu die folgende Situation: Ein Protokoll $\pi^{\mathcal{F}}$ implementiert sicher eine Funktionalität $\mathcal{G}$. Dabei benutzt $\pi^{\mathcal{F}}$ die Funktionalität $\mathcal{F}$, die wir als (im realen Modell) ideal gegeben annehmen. So könnte z. B. $\pi^{\mathcal{F}}$ ein Protokoll für elektronische Wahlen sein, das bei Benutzung von idealisierten Commitments als sicher bewie-

sen wurde. Dann ist $\mathcal{F}$ die oben bereits erwähnte Commitment-Funktionalität, und $\mathcal{G}$ stellt eine geeignet definierte Funktionalität für elektronische Wahlen dar. Weiterhin nehmen wir an, daß ein Protokoll ρ existiert, welches $\mathcal{F}$ sicher implementiert (in unserem Beispiel also ein Commitment-Protokoll). Dann würde man intuitiv erwarten, daß das Protokoll π^ρ ebenfalls $\mathcal{G}$ implementiert, wenn wir π^ρ aus $\pi^\mathcal{F}$ konstruieren, indem wir Aufrufe der Funktionalität $\mathcal{F}$ durch Aufrufe des Subprotokolls ρ ersetzen. Diese Art der Komponierbarkeit wird von den simulationsbasierten Sicherheitsdefinitionen mit Umgebung in der Tat garantiert (und, wie weiter unten erläutert, nur von diesen).

Aber genauso wie es bei den in den vorangegangenen Abschnitten genannten Typen der Komposition (bei denen ein Protokoll nur mit sich selbst und nicht in beliebigen Zusammenhängen komponiert wurde) mehrere Varianten gab (sequentiell, parallel und nebenläufig), so unterscheiden wir auch hier zwei Arten der Komposition: zum einen können wir verlangen, daß das Protokoll $\pi^\mathcal{F}$ nur eine Instanz von $\mathcal{F}$ aufruft (und somit π^ρ nur eine Instanz des Subprotokolls ρ laufen läßt), oder wir lassen zu, daß $\pi^\mathcal{F}$ polynomiell viele Instanzen von $\mathcal{F}$ aufruft (d. h. die Anzahl der Instanzen ist polynomiell im Sicherheitsparameter). Letzteres ist – falls $\pi^\mathcal{F}$ polynomiell-beschränkt ist – gleichbedeutend damit, keine Beschränkung bezüglich der Anzahl der Instanzen zu fordern, da ein polynomiell-beschränktes Protokoll nur polynomiell viele Subprotokolle aufrufen kann. Die erste Variante nennen wir im folgenden *einfache Komposition*, die zweite *(polynomiell-beschränkte) allgemeine Komposition*[1].

Da viele Protokolle die Eigenschaft haben, viele Instanzen eines Subprotokolls aufzurufen, ist allgemeine Komponierbarkeit sicherlich die wünschenswerte Eigenschaft. Doch welche Variante wird nun von den Sicherheitsmodellen mit Umgebung garantiert? Es stellt sich heraus, daß es hierbei auf ein unscheinbares Detail in der Definition ankommt, nämlich die genaue Reihenfolge der Quantoren. Erlaubt man nämlich, daß der Simulator in Abhängigkeit von der Umgebung gewählt wird (was wir im folgenden *spezielle Sicherheit* nennen werden), so wurde in [PW01] gezeigt, daß einfache Komponierbarkeit folgt. Verlangt man jedoch, daß der Simulator unabhängig von der Umgebung gewählt wird (im folgenden *allgemeine Sicherheit*), so erhalten wir allgemeine Komponierbarkeit [Can01]. Diese Arbeiten konnten allerdings nicht klären, inwiefern die angegebenen Sicherheitsbegriffe tatsächlich notwendige, und nicht nur hinreichende

[1] Diese Art von Komponierbarkeit wird oft auch *universal composability* genannt. Leider hat sich dieser Begriff gleichzeitig als Bezeichnung für Protokolle, die bezüglich des Modells von [Can01] sicher sind, eingebürgert. Da aber diese Sicherheit nur hinreichend, aber nicht notwendig für allgemeine Komponierbarkeit ist (wie wir in Kapitel 6 zeigen), führt diese sprachliche Gleichsetzung zu viel Verwirrung. Da wir in der vorliegenden Arbeit gerade auch den Unterschied zwischen simulationsbasierter Sicherheit mit Umgebung und der allgemeinen Komponierbarkeit untersuchen, ist eine Benutzung dieses mehrdeutigen Begriffs in unserem Kontext ausgeschlossen, und wir verwenden den weniger gebräuchlichen, aber klareren von [Lin03] eingeführten Begriff der *allgemeinen Komponierbarkeit* (*general composability*).

Bedingungen für die jeweiligen Typen der Komposition sind.

Dieses Problem wurde von [Lin03] aufgegriffen. Dort wurde gezeigt, daß spezielle Sicherheit tatsächlich eine notwendige Bedingung für die einfache Komponierbarkeit ist, daß diese beiden Begriffe also äquivalent sind. Doch konnte [Lin03] die Frage nicht beantworten, ob ein ähnliches Resultat auch für allgemeine Sicherheit und allgemeine Komponierbarkeit gilt. Es war sogar unklar, ob spezielle und allgemeine Sicherheit nicht zusammenfallen. Wie wir in der vorliegenden Arbeit zeigen, hängt die Antwort auf diese Frage von der (komplexitätstheoretischen) Mächtigkeit des Angreifers ab, je nach Klasse von Angreifern fallen nämlich die beiden Sicherheitsbegriffe und die allgemeine Komponierbarkeit zusammen oder sind verschieden.

Abschließend sollte noch angemerkt werden, daß simulationsbasierte Sicherheitsmodelle mit Umgebung leider nicht alle definitorischen Probleme lösen. Zum einen zeigten [CF01, CKL03], daß viele natürliche, und bis dato als realisierbar geltende Aufgabenstellungen ohne zusätzliche Annahmen über die den Protokollen zur Verfügung stehende Infrastruktur in den simulationsbasierten Sicherheitsmodellen mit Umgebung *nicht realisierbar* sind. Da keine solchen Unmöglichkeitsresultate für die Modelle [Can95, Can00] für sichere Funktionsauswertungen existieren (vielmehr gibt es allgemeine *Möglichkeits*aussagen [Gol04]), wird dies oftmals gegen die Sicherheitsmodelle mit Umgebung vorgebracht. Doch das Ergebnis von [Lin03] zeigt uns, daß diese Unmöglichkeitsresultate nicht ein Artefakt dieser speziellen Definitionen sind, sondern vielmehr ein notwendiges Übel, wenn wir allgemeine Komponierbarkeit (oder auch nur spezielle) haben wollen. Außerdem existieren Protokolle, die schon unter relativ geringen Annahmen an die zur Verfügung stehende Infrastruktur[2] oben genannte (und beliebige andere) Funktionalitäten realisieren [CF01, CLOS02]. Allerdings muß man in Kauf nehmen, daß die resultierenden Protokolle wesentlich komplexer sind als es für die alten Sicherheitsbegriffe nötig gewesen wäre.

Das zweite Problem ist, daß einige Aufgabenstellungen nur schwer über eine intuitive und übersichtliche Funktionalität spezifiziert werden können (vgl. z. B. [BH04]). Damit geht zu einem gewissen Maße der Anspruch verloren, man könne einen Sicherheitsbegriff für eine spezielle Anwendung *einfach* aus der allgemeinen Sicherheitsdefinition herleiten, indem man eine Funktionalität entwirft, die in offensichtlicher Weise die Anwendung wiedergibt.

Doch in den Fällen, in denen das Design einer passenden Funktionalität möglich ist, und in denen die obigen Unmöglichkeitsresultate nicht zuschlagen (weil z. B. ein *common reference string* oder eine Public-Key-Infrastruktur zur Verfügung steht), stellen die simulationsbasierten Sicherheitsmodelle mit Umgebung ein mächtiges Werkzeug für den modularen Protokollentwurf und Sicherheitsbe-

[2] Es ist ein sogenannter *common reference string* nötig, dies ist ein zufälliger String, der von einer vertrauenswürdigen Instanz gewählt wird und öffentlich bekannt ist.

weis dar, das aus der modernen Kryptographie nicht mehr wegzudenken ist.

2.2. Das Sicherheitsmodell

Es gibt zwei populäre Definitionen von simulationsbasierten Sicherheitsmodellen mit Umgebung ([PW01, Bac02b, BPW04b] und [Can01, Can05]).

Im folgenden stellen wir ein an [BPW04b] angelehntes Modell vor, auf dessen Basis wir unsere Ergebnisse formulieren werden. Es sei aber angemerkt, daß sich die Ergebnisse ebenso einfach im Modell von [Can01] zeigen lassen.

Bei unserer Darstellung nehmen wir einige Anpassungen gegenüber dem Originalmodell von [BPW04b] vor. Die meisten Änderungen sind rein kosmetischer Natur oder verallgemeinern die Definitionen, so daß auch die Sicherheitsvarianten aus dem Modell von [Can01] repräsentiert werden können, z. B. durch das Einführen eines optionalen Auxiliary input (siehe Abschnitt 2.2.4.1), oder durch die Möglichkeit, daß die Umgebung eine Ausgabe hat. Drei Änderungen sollten jedoch hervorgehoben werden:

- Das Konzept der sogenannten *Buffer* wurde entfernt und durch sofortige Nachrichtenauslieferung ersetzt. Wir glauben, daß asynchrone geheime Nachrichtenauslieferung besser durch entsprechende ideale Funktionalitäten modelliert werden sollte als durch einen direkt ins Netzwerkmodell integrierten Mechanismus.[3]

- Das Konzept der *Längenfunktionen* wurde variiert. Diese dienen dazu, Leitungen zu anderen Maschinen zu trennen. Aus technischen Gründen ist eine solche Trennung notwendig für die Modellierung des Polynomialzeitbegriffs (siehe Abschnitt 2.2.3). Die Längenfunktionen aus [BPW04b] jedoch erlauben es zusätzlich, Leitungen zu beliebiger Zeit wieder zu reaktivieren. Dies kann zu seltsamen Effekten führen und ist für die ursprüngliche Zweckbestimmung der Längenfunktionen nicht notwendig. Daher erlegen wir die Beschränkung auf, daß eine Leitung, die einmal getrennt wurde, nicht wieder geöffnet werden kann.

- Die Definition der statistischen Sicherheit wurde verändert. In [HU05b] wurde gezeigt, daß die ursprüngliche Definition nicht einmal einfache Komponierbarkeit ermöglicht. Wir verwenden deshalb die in [HU05b] vorgeschlagene Definition der statistischen Sicherheit.

[3] Darüber hinaus ist die Modellierung der Buffer recht willkürlich. Die ausliefernde Maschine (meist der Angreifer) darf auswählen, welche Nachricht aus einer Warteschlange ausgeliefert werden soll. Da der Angreifer dabei keinen Einblick in die Warteschlange hat (die Nachrichtenauslieferung ist geheim), macht es deshalb einen Unterschied, auf welche Art die Nachricht in der Schlange addressiert wird (Position von vorne, von hinten, absolute Nummer der Nachricht, etc.) Eine Entscheidung für eine der Adressierungsarten ist deshalb notwendig eine willkürliche Festlegung.

2.2.1. Überblick und Motivation

In diesem Abschnitt geben wir einen Überblick über das Modell. Die genauen Definitionen finden sich dann in den nachfolgenden Abschnitten.

Um die Grundidee des in den folgenden Abschnitten vorgestellten Sicherheitsmodells zu verstehen, stellen wir uns zunächst auf den folgenden Standpunkt und versuchen daraus, einen Begriff der Sicherheit herzuleiten: „Ein Angriff ist nur dann schädlich, wenn seine Auswirkungen in irgendeiner Weise beobachtbar sind. Ein Protokoll ist sicher, wenn kein schädlicher Angriff möglich ist.“ Dabei verstehen wir unter einem beobachtbaren Angriff natürlich nicht, daß der Angriff unmittelbar sichtbare Folgen haben muß, wir nennen es auch beobachtbar, wenn z. B. eine betrügende Partei etwas lernt, was sie nicht lernen sollte. Die vorliegende Formulierung unseres Standpunktes ist aber noch zu stark. Wir haben nicht festgelegt, was ein Angriff ist, außer der Tatsache, daß er beobachtbar sein muß. So wäre z. B. der Erhalt eines Blumenstraußes nach unserer Definition auch potentiell ein Angriff, da beobachtbar. Wir müssen daher irgendwie angeben, welche „Angriffe“ wir zulassen wollen. Anstatt zu versuchen, eine Klassifikation in Gut und Böse zu erstellen, beschreiben wir eine ideale Welt, in der per Definition keine schädlichen Angriffe stattfinden dürfen. Diese ideale Welt beschreibt somit gleichzeitig, was unser Protokoll überhaupt tun soll, als auch welche „Angriffe“ erlaubt sind. Unsere Leitregel ist nun also die folgende: „Ein Angriff ist nur dann schädlich, wenn seine Auswirkungen in irgendeiner Weise beobachtbar sind, *und diese beobachteten Auswirkungen in einer idealen (also per Definition sicheren) Welt nicht vorkommen können.* Ein Protokoll ist sicher, wenn kein schädlicher Angriff möglich ist.“

Welche Entitäten kommen in dieser umgangssprachlichen Formulierung vor? Zunächst einmal natürlich das *reale Protokoll* π, dessen Sicherheit wir zeigen wollen. Darüber hinaus haben wir die ideale Welt, über die wir unsere Anforderungen an das Protokoll π spezifizieren. Diese ideale Welt modellieren wir, indem wir ein weiteres *ideales* Protokoll ρ einführen, welches wir als sicher definieren. (Dieses ideale Protokoll wird üblicherweise so einfach sein, daß seine Sicherheit offensichtlich ist, etwa unter Benutzung von nach Voraussetzung unkorrumpierbaren Hilfsparteien und sicheren Kanälen. Oft besteht das ideale Protokoll auch nur aus einer Maschine, die wir die *ideale Funktionalität* nennen.)

Zusätzlich haben wir einen *Angreifer* $\mathcal{A}$, welcher das reale Protokoll π angreift. Doch wir müssen auch Angriffe in der idealen Welt betrachten. Diese sind zwar per Definition nicht schädlich, aber um vergleichen zu können, ob reale und ideale Angriffe die gleichen Auswirkungen haben können, müssen wir auch im idealen Modell Angriffe zulassen. Man mag nun versucht sein, den Angreifer $\mathcal{A}$ auch im idealen Modell anzusetzen, da wir ja dessen Angriffe vergleichen wollen. Doch das wäre die Umsetzung der Formulierung „. . . und diese beobachteten Auswirkungen in einer idealen Welt *bei dem gleichen Angriff* nicht vorkommen

können...". Dies aber wäre eine zu strenge Forderung, denn da Angriffe im idealen Modell nie schädlich sein können, wäre schon die Existenz eines *anderen* Angriffs mit der gleichen Auswirkung Beweis genug, daß diese Auswirkungen, und damit auch ein realer Angriff mit den gleichen Auswirkungen, nicht schädlich sind. Wir müssen also sagen: „Ein Angriff ist nur dann schädlich, wenn seine Auswirkungen in irgendeiner Weise beobachtbar sind, und diese beobachteten Auswirkungen in einer idealen Welt *bei keinem Angriff* vorkommen können." Somit kommt eine weitere Entität ins Spiel, die den Angriff im idealen Modell durchführt. Diese nennen wir den *Simulator* $\mathcal{S}$, da er (zumindest wenn wir die Sicherheit erfolgreich beweisen) jeden Angriff des realen Angreifers $\mathcal{A}$ so nachmachen muß, daß die Auswirkungen die gleichen sind.

Zu guter Letzt versteckt sich in unserer Leitregel noch eine weitere Entität: Wir haben schädliche Angriffe darüber definiert, daß ihre Auswirkungen beobachtbar sind. Daher muß unser Modell noch einen Beobachter enthalten, für den die Auswirkungen im realen und im idealen Modell ununterscheidbar sind. Da wir einen möglichst weiten Begriff einer beobachtbaren Auswirkung haben wollen (da wir sonst möglicherweise ein Protokoll zu unrecht als sicher bezeichnen), lassen wir als Beobachter alles zu, was mit dem Protokoll interagieren kann, wir führen die *Umgebung* $\mathcal{Z}$ ein. Man kann sich diese Umgebung als die Gesamtheit aller Protokollbenutzer vorstellen, zusammen vielleicht mit weiteren Protokollen, die diese ausführen,[4] sowie weiteren Personen, die möglicherweise mit dem Angreifer interagieren. Die Interaktion mit dem Angreifer ist besonders wichtig, da dadurch die Tatsache, daß der Angreifer etwas lernt, in den Bereich des Beobachtbaren fällt: er kann etwas darüber mitteilen. Man ist vielleicht versucht anzumerken, daß wir die Angriffe vernachlässigen, bei denen der Angreifer geheime Informationen abgreift, aber für sich behält. Dem kann man mit zweierlei Argumenten begegnen: Das formale Argument ist, daß wenn ein Angreifer Geheimnisse erfahren kann, so existiert ein anderer Angreifer, der diese auch verrät. Damit ist ein Protokoll, in dem ein Angreifer private Informationen für sich erhalten kann, auch nicht sicher. Die positivistische Antwort wäre, daß wenn ein Angreifer einen erfolgreichen Angriff durchführt, dies aber *keine* Auswirkungen auf die Welt des Beobachters hat, dieser Angriff gar nicht relevant ist. Es bleibt dem Leser überlassen, auf welche der Interpretationen er seine Intuition stützen möchte.

Unsere neue Entität, die Umgebung $\mathcal{Z}$ kann also die Ein- und Ausgaben aller Protokollteilnehmer (Parteien) wählen bzw. lesen, kann mit dem Angreifer (oder Simulator) kommunizieren, und es soll für die Umgebung dabei ununterscheidbar sein, ob sie mit dem realen Protokoll mit Angreifer oder mit dem idealen Protokoll mit Simulator spricht.

[4] Aus der Möglichkeit, sich andere Protokolle als Teil der Umgebung vorzustellen, resultiert auch die weiter unten besprochene Eigenschaft der Komponierbarkeit.

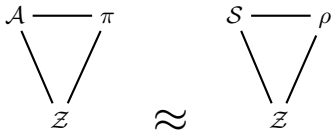


Abbildung 2.1.: Der Sicherheitsbegriff. Das reale Protokoll π ist so sicher wie das ideale Protokoll ρ, wenn für jeden Angreifer $\mathcal{A}$ ein Simulator $\mathcal{S}$ existiert, so daß keine Umgebung $\mathcal{Z}$ die beiden dargestellten Situationen unterscheiden kann.

Zusammenfassend stellen wir uns Sicherheit nun also wie folgt vor: Ein reales Protokoll π ist so sicher wie ein ideales Protokoll ρ, wenn für jeden realen Angreifer $\mathcal{A}$ ein Simulator $\mathcal{S}$ existiert, so daß keine Umgebung $\mathcal{Z}$ zwischen einem Protokollauf von $\mathcal{Z}, \mathcal{A}, \pi$ und einem von $\mathcal{Z}, \mathcal{S}, \rho$ unterscheiden kann (vgl. Abbildung 2.1).

Um diese Grundidee zu einem formalen Sicherheitsmodell werden zu lassen, gilt es einige Entwurfsentscheidungen zu treffen:

- Die wahrscheinlich wichtigste Entscheidung ist die Komplexität von Umgebung, Angreifer und Simulator, sowie die Wahl des Begriffs der Ununterscheidbarkeit. Wir betrachten drei Varianten: Zunächst haben wir die *perfekte Sicherheit*, bei denen Umgebung, Angreifer und Simulator unbeschränkte Maschinen sind, und bei der wir verlangen, daß die Beobachtungen der Umgebung perfekt ununterscheidbar sind, d. h. die Beobachtungen real wie ideal die gleiche Verteilung haben.

  Bei der *statistischen Sicherheit* lassen wir ebenfalls unbeschränkte Maschinen zu, aber wir verlangen nur die statistische Ununterscheidbarkeit der Beobachtungen der Umgebung. Das heißt die Verteilungen der Beobachtungen im realen und im idealen Modell dürfen sich nur um einen vernachlässigbaren Betrag unterscheiden (im Sinne von Definition 1.2).

  Bei der *komplexitätstheoretischen Sicherheit* hingegen berücksichtigen wir die Tatsache, daß Berechnungen in der Realität gewisse Schranken auferlegt sind. Deshalb betrachten wir dort nur polynomiell-beschränkte Umgebungen, Angreifer und Simulatoren. Weiterhin verlangen wir auch nur, daß die Beobachtungen der Umgebung komplexitätstheoretisch ununterscheidbar sind. Wir erlauben also, daß die Auswirkungen eines Angriffs im Realen völlig andere sind als im Idealen, vorausgesetzt, es ist nicht möglich diese Auswirkungen in polynomieller Zeit zu unterscheiden. (Hierhinter kann man die Auffassung sehen, daß nicht nur das, was nicht beobachtbar ist, irrelevant ist, sondern auch das, was zwar beobachtbar, aber nicht erkennbar ist.)

- Eng verwandt mit der Komplexität der Maschinen ist die Frage, ob wir einen sog. *Auxiliary input* einführen. Dieser Auxiliary input ist eine Zusatzeingabe für die Umgebung, die nichtuniform nach Angreifer, Umgebung und Simulator gewählt wird. Historisch ist der Auxiliary input aus der Interpretation erwachsen, daß er Informationen darstellt, die aus früheren Protokolläufen gewonnen wurden. Der Auxiliary input hat sich auch in vielen Fällen als sehr wichtig erwiesen, um Kompositionsergebnisse zu erhalten.

  In unserem Falle sind diese Argumente jedoch nicht mehr so zwingend: Zum einen haben wir die Umgebung eingeführt, als Teil derer wir neben dem unter Betrachtung stehenden Protokoll stattfindende Protokollausführungen auffassen können. Damit aber sind auch vor dem eigentlichen Protokollauf stattfindende Protokolle abgedeckt, und daraus gewonnene Informationen der Umgebung bekannt. Zum anderen benötigen wir den Auxiliary input auch nicht, um Kompositionstheoreme zu erhalten, da diese auch ohne Vorhandensein des Auxiliary input gelten. (In allen in dieser Arbeit betrachteten Fällen gilt das Kompositionstheorem mit Auxiliary input genau dann, wenn es auch ohne Auxiliary input gilt.)

  Der Auxiliary input ist jedoch ein in der Kryptologie vielgebrauchtes Konzept, und daher scheint es sinnvoll, sowohl Sicherheit *mit* als auch *ohne Auxiliary input* zu betrachten.

- Wir müssen weiterhin entscheiden, was wir unter den „Beobachtungen der Umgebung“ überhaupt verstehen. Wir haben hier zwei Möglichkeiten. Zum einen können wir als die Beobachtung die gesamte Kommunikation sowie alle internen Berechnungen der Umgebung betrachten. Dies nennen wir die *Sicht* der Umgebung und sprechen von Sicherheit *bzgl. der Sicht der Umgebung*. Andererseits hat es vor allem praktische beweistechnische Vorteile, die Umgebung zu einem von ihr wählbaren Zeitpunkt eine von ihr wählbare Ausgabe generieren lassen, und diese Ausgabe als ihre Beobachtung zu betrachten. Wir nennen dies Sicherheit *bzgl. der Ausgabe der Umgebung*. Da die Umgebung einfach alle ihre Beobachtungen ausgeben kann, ist der Unterschied zwischen diesen Formalisierungen nicht groß, doch macht es in gewissen Fällen einen Unterschied, wenn die Protokollausführung nicht terminiert.

- Eine unscheinbare, aber wichtige Entscheidung ist die Reihenfolge der Quantoren. Oben haben wir π als so sicher wir ρ bezeichnet, wenn für jeden Angreifer ein Simulator existiert, so daß für jede Umgebung $\mathcal{Z}$ gilt: $\mathcal{Z}$ kann nicht unterscheiden. Doch ist auch, mit ebenso guter Berechtigung, eine andere Formalisierung denkbar: Für jeden Angreifer und jede Umgebung $\mathcal{Z}$ gibt es einen Simulator, so daß $\mathcal{Z}$ nicht unterscheiden kann.

> Wir werden den ersten Begriff (bei dem die Umgebung vom Simulator abhängen darf) als *allgemeine Sicherheit*, und den zweiten (bei dem der Simulator von der Umgebung abhängt) als *spezielle Sicherheit* bezeichnen. Obgleich kein intuitiver Unterschied zwischen diesen beiden Begriffen erkennbar ist, unterscheiden sie sich doch in ihren Eigenschaften, und die Analyse dieses Unterschieds wird in dieser Arbeit eine große Rolle spielen.

In den soeben betrachteten Punkten haben wir noch eine sehr wichtige Frage außer acht gelassen: Was verstehen wir überhaupt unter einem Protokoll, und wie modellieren wir die Kommunikation, ohne die die Ausführung eines Protokolls keinen Sinn macht?

Wir betrachten zuerst das Kommunikationsmodell: Wir stellen uns ein *Netzwerk* als eine Menge von Maschinen vor. Jede Maschine hat eine Menge von ein- und ausgehenden *Ports*, und jeder eingehende Port ist mit einem ausgehenden Port einer anderen Maschine verbunden. In unserem Modell ist immer nur eine Maschine auf einmal aktiv, und diese Maschine kann nach ihrer Aktivierung eine beliebige Nachricht m über einen ihrer ausgehenden Ports senden. Daraufhin wird die Maschine aktiviert, die den zugehörigen eingehenden Port hat und erhält die Nachricht m. Sendet eine Maschine keine Nachricht, so folgt aus den angegebenen Regeln nicht, wer als nächster aktiviert werden soll (wir sagen manchmal *das Token gehe verloren*). Deshalb gibt es eine ausgezeichnete Maschine, den *Scheduler*, welche in diesem Fall aktiviert wird (bei uns wird diese Rolle der Angreifer übernehmen). Die Verbindungen zwischen den Ports stellen also die Verbindungen zwischen den Maschinen dar, und diese Verbindungen sind per Definition völlig sicher und haben sofortige Auslieferung. Solch ideale Kanäle sind vielleicht etwas unrealistisch, doch läßt uns dieses Modell die Möglichkeit, weniger sichere Kanäle durch Maschinen zu modellieren, die die entsprechenden Imperfektionen explizit modellieren.

Mit diesem Netzwerk- und Kommunikationsmodell ist es nun einfach zu definieren, was ein Protokoll ist: Intuitiv ist ein Protokoll eine Kommunikationsvorschrift, welche in Abhängigkeit von Eingaben gewisse Nachrichten zwischen gewissen Parteien umherschickt und bestimmte Ausgaben liefert. Dies modelliert man einfach dadurch, daß man die verschiedenen Parteien des Protokolls, sowie die Funktionalitäten, die z. B. evtl. vorhandene physikalische Annahmen wie bestimmte Kanäle modellieren, durch Maschinen realisiert, die ihre Ein- und Ausgaben über Ports von der Umgebung erhalten (bzw. an diese senden), sowie über andere Ports untereinander kommunizieren. Ein Protokoll ist formal also nichts weiter als eine Menge von Maschinen. Die mit der Umgebung verbundenen Ports nennen wir *Protokollein- und -ausgabeports*, die für die Kommunikation innerhalb des Protokolls *interne Ports*. Weiterhin gibt es noch Verbindungen zwischen Protokoll und Angreifer bzw. Simulator, diese gehen über die *Angreiferports*. Diese sind notwendig, falls gewisse Imperfektionen in unseren Funktio-

nalitäten es notwendig machen, diese durch einen Informationsfluß vom und zum Angreifer zu modellieren. Ein unsicherer Kanal z. B. schickt alle Nachrichten an den Angreifer, und kann von diesem neue (gefälschte) Nachrichten erhalten.

Wir können nun zusammenfassen: Es seien zwei Protokolle (also Mengen von Maschinen) π und ρ gegeben. Dann ist π so sicher wie ρ bezüglich allgemeiner/spezieller perfekter/statistischer/komplexitätstheoretischer Sicherheit mit/ohne Auxiliary input bzgl. der Ausgabe/Sicht der Umgebung, wenn folgendes gilt:

- Im Falle allgemeiner Sicherheit: Zu jedem Angreifer $\mathcal{A}$ existiert ein Simulator $\mathcal{S}$, so daß für jede Umgebung $\mathcal{Z}$ diese nicht zwischen π und ρ unterscheiden kann (s. u.).
- Im Falle spezieller Sicherheit: Zu jedem Angreifer $\mathcal{A}$ und jeder Umgebung $\mathcal{Z}$ existiert ein Simulator $\mathcal{S}$, so daß die Umgebung nicht zwischen π und ρ unterscheiden kann.
- Dabei sind Angreifer, Umgebung und Simulator im Falle der perfekten und der statistischen Sicherheit beliebige, im Falle der komplexitätstheoretischen Sicherheit polynomiell-beschränkte Maschinen.
- Die Umgebung darf nur mit den Protokollein- und -ausgabeports des Protokolls verbunden werden, Angreifer und Simulator nur mit den Angreiferports.
- „Die Umgebung kann nicht zwischen π und ρ unterscheiden“ bedeutet, daß die die Ausgabe/die Sicht der Umgebung in einer Ausführung des Netzwerkes bestehend aus $\mathcal{Z}$, $\mathcal{A}$ und π und in einer Ausführung des Netzwerkes bestehend aus $\mathcal{Z}$, $\mathcal{S}$ und ρ (i) im Falle perfekter Sicherheit die gleiche, (ii) im Falle statistischer Sicherheit statistisch ununterscheidbare, und (iii) im Falle komplexitätstheoretischer Sicherheit komplexitätstheoretisch ununterscheidbare Verteilungen hat.

Diese Definitionsskizze läßt natürlich noch viele Details offen, die in den nachfolgenden Abschnitten geklärt werden sollen.

Die hier beschriebene Sicherheitsdefinition hat einen sehr großen Vorteil: Die Möglichkeit der Komposition. Man stelle sich vor, wir wollten ein Protokoll für eine gewisse kryptographische Aufgabenstellung entwerfen, sagen wir eine eCommerce-Anwendung. Gegeben ist also das ideale Protokoll ρ_{eC} und wir suchen ein Protokoll π_{eC}, das so sicher wie ρ_{eC} ist. Um den Entwicklungs- und Beweisaufwand aber in Grenzen zu halten, nehmen wir zunächst an, wir hätten ein ideales Commitment gegeben, d. h. wir entwickeln ein Protokoll $\pi_{\mathrm{eC}}^{\mathcal{F}_{\mathrm{COM}}}$,

welches eine ideale Commitment-Funktionalität $\mathcal{F}_{\mathrm{COM}}$ verwendet, d. h. eine vertrauenswürdige Maschine, welche von einer Partei P_1 ein Bit entgegennimmt, und dieses erst dann an eine andere Partei P_2 weiterleitet, wenn P_1 dies gestattet. Wir nehmen weiter an, wir hätten ein sicheres Commitment-Protokoll, d. h. ein Protokoll π_{COM}, welches so sicher wie $\mathcal{F}_{COM}$ ist. Nun scheint es natürlich, in $\pi_{\mathrm{eC}}^{\mathcal{F}_{\mathrm{COM}}}$ Aufrufe der idealen Funktionalität $\mathcal{F}_{\mathrm{COM}}$ einfach durch Aufrufe des Commitment-Protokolls π_{COM} zu ersetzen, und zu hoffen, daß das resultierende Protokoll $\pi_{\mathrm{eC}} := \pi_{\mathrm{eC}}^{\pi_{\mathrm{COM}}}$ so sicher wie ρ_{eC} ist. In der Tat aber gibt es viele Sicherheitsbegriffe, die eine solche *Komposition* i. a. nicht erlauben, d. h. bei denen die Sicherheit bei einer solchen Operation verloren gehen kann. Der Vorteil der Sicherheitsdefinitionen mit Umgebung, erkauft durch einen sehr strengen Sicherheitsbegriff, ist gerade, daß sie sichere Komposition garantieren. Auf diese Art ist es möglich, größere Protokolle wie π_{eC} modular aufzubauen und die Sicherheit des zusammengesetzten Protokolls mittels des Kompositionstheorems zu beweisen.

Nun gilt es aber noch eine Feinheit zu unterscheiden. Benutzt $\pi_{eC}^{\mathcal{F}_{\mathrm{COM}}}$ *eine* oder *mehrere* Instanzen der idealen Funktionalität $\mathcal{F}_{\mathrm{COM}}$? Wenn wir nur *eine* Instanz benutzen, sprechen wir von *einfacher Komposition* (Abschnitt 2.3.1). Wenn das Rahmenprotokoll jedoch nicht nur eine, sondern bis zu f Instanzen von $\mathcal{F}_{\mathrm{COM}}$ verwendet, so reden wir von *(f-beschränkter) allgemeiner Komposition* (Abschnitt 2.3.4). Um diesen Unterschied klarer hervortreten zu lassen, schreiben wir $\pi_{eC}^{f\cdot\mathcal{F}_{\mathrm{COM}}}$ in dem Fall, daß π_{eC} f Instanzen von $\mathcal{F}_{\mathrm{COM}}$ aufruft. Es stellt sich nun die Frage: Welche Variante der Komposition erlauben unsere Sicherheitsbegriffe? Hier ist der Unterschied zwischen spezieller und allgemeiner Sicherheit relevant: Spezielle Sicherheit garantiert einfache Komposition, und allgemeine Sicherheit garantiert sogar (polynomiell-beschränkte) allgemeine Komposition. Ob und in welchen Fällen aber spezielle Sicherheit allgemeine Komposition garantiert, hängt von der betrachteten Variante der speziellen Sicherheit ab, und ist eine der zentralen Fragen dieser Arbeit.

2.2.2. Maschinen, Netzwerke und Protokolle

Wir möchten an dieser Stelle den Leser vor diesem Abschnitt und Abschnitt 2.2.3 warnen: Beide enthalten viele Details der Definitionen unseres Maschinenmodells und der Komplexität von Maschinen, die zwar für die formalen Details der späteren Kapitel notwendig, aber für ein intuitives Verständnis der Aussagen *und Beweise* in dieser Arbeit nicht unabdingbar sind. Wir empfehlen dem Leser daher, beim ersten Lesen diese zwei Abschnitte zu überspringen oder zu überfliegen, und stattdessen zunächst mit den intuitiven Vorstellungen aus Abschnitt 2.2.1 zu arbeiten.

Da es unser Ziel ist, ein reales Protokoll mit einer idealen Funktionalität zu vergleichen, brauchen wir zunächst einmal Definitionen für die elementaren Objekte wie Maschinen, Netzwerke und Protokolle (wir benötigen keine spezielle Definition für ideale Funktionalitäten, da diese einfach als Maschinen aufgefaßt werden).

Zunächst betrachten wir die Definition einer *Maschine*. Wir verstehen unter einer Maschine eine Entität, die durch eine von außen kommende Nachricht aktiviert werden kann, und dann mittels eines näher zu spezifizierenden probabilistischen Prozesses ihren internen Zustand verändert und potentiell eine neue ausgehende Nachricht erzeugt. Wir gehen davon aus, daß eine Maschine bei jeder Aktivierung auf die folgenden Daten zurückgreifen kann (vgl. auch Abbildung 2.2).

- Die eingehende Nachricht m
- Ihren internen Zustand s
- Den Sicherheitsparameter k

Die eingehende Nachricht und den internen Zustand haben wir bereits erwähnt. Der Sicherheitsparameter wird gebraucht, um eine asymptotische Formulierung der Sicherheit zuzulassen. Da sich das Protokoll üblicherweise für jeden Sicherheitsparameter leicht anders verhält, müssen die Maschinen diesen kennen.

Damit eine Maschine mit anderen kommunizieren kann, brauchen wir außerdem das Konzept eines *Ports*. Es gibt zwei Typen von Ports, eingehende und ausgehende, und jeder Port hat einen eindeutigen Namen (genaugenommen kann eine Maschine zwei Ports gleichen Namens haben, einen ein- und einen ausgehenden). Eine eingehende Nachricht ist immer einem eingehenden Port zugeordnet (diese Zugehörigkeit gibt an, woher diese Nachricht stammt), und eine ausgehende Nachricht wird immer über einen ausgehenden Port geschickt (hiermit wird der Empfänger bestimmt). Formal identifizieren wir einen Port mit seinem Namen.

Um den probabilistischen Prozess, der das Verhalten der Maschine definiert, in allgemeinster Weise zu beschreiben, verzichten wir darauf, bei der abstrakten Definition der Maschine irgendein konkretes Maschinenmodell wie Turing- oder RAM-Maschinen zugrundezulegen. Vielmehr beschränken wir uns auf eine probabilistische Beschreibung des Verhaltens und verwenden konkrete Maschinenmodelle nur zur Definition von bestimmten Klassen von Maschinen, z. B. den polynomiell-beschränkten Maschinen. Deshalb ordnen wir einfach jeder Kombination der Eingaben eine Wahrscheinlichkeitsverteilung auf den Ausgaben zu.[5]

[5] Da nur abzählbar viele Eingaben möglich sind, ist diese Vorgehensweise gerechtfertigt. Gäbe es überabzählbar viele verschiedene Eingaben, so müßte man stattdessen einen Markov-Kern verwenden. (Vgl. z. B. [Bau02, § 36])

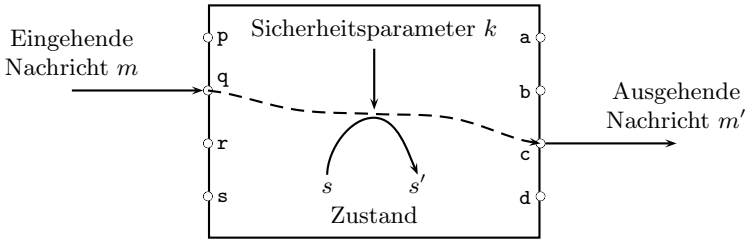


Abbildung 2.2.: Schema einer Aktivierung einer Maschine. Beispielhaft sind die eingehenden Ports $\mathtt{p}, \mathtt{q}, \mathtt{r}, \mathtt{s}$ und die ausgehenden Ports $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}$ eingezeichnet. Die Maschine wird durch eine Nachricht auf Port $\mathtt{q}$ aktiviert und antwortet mit einer Nachricht auf Port $\mathtt{c}$. Dabei verändert sich der interne Zustand von s zu s'.

Definition 2.1 (Maschinen)
Eine *Maschine* M besteht aus einem Namen $name_M \in \Sigma^+$, einer endlichen Menge $in(M) \subset \Sigma^+$ (den *eingehenden Ports*), einer endlichen Menge $out(M) \subset \Sigma^+$ (den *ausgehenden Ports*) und einer Funktion M von $\mathbb{N} \times \Sigma^* \times in(M) \times \Sigma^*$ auf die Menge der Wahrscheinlichkeitsmaße über $\Sigma^* \times \{out(M) \cup \lambda\} \times \Sigma^*$ (der *Zustandsübergangsfunktion*).

Dabei sind die Argumente der Funktion wie folgt zu interpretieren: $M(k, s, p, m)$ beschreibt das Verhalten der Maschine bei Sicherheitsparameter k, Zustand s und eingehender Nachricht m auf Port p. Dabei ist die Wahrscheinlichkeit, daß M nach der Aktivierung in Zustand s' ist und die Nachricht m' auf Port p' ausgibt, gerade die Wahrscheinlichkeit von (s', p', m') unter der Verteilung $M(k, s, p, m)$. Soll keine Nachricht gesandt werden, besteht die Möglichkeit, $p' := \lambda$ zu setzen.

Wenn wir konkrete Maschinen angeben, werden wir sie nicht als probabilistische Funktionen notieren, sondern durch ihr Verhalten textuell beschreiben. Dies folgt nicht nur den auf diesem Gebiet üblichen Konventionen, sondern erhöht die Lesbarkeit wesentlich. Die obige Definition soll es dem Leser aber ermöglichen, im Zweifelsfalle nachzuvollziehen, was formal vor sich geht.

Versehen mit dieser Maschinendefinition können wir nun einfach den Begriff eines *Netzwerks* fassen. Ein Netzwerk ist nämlich nichts anderes als eine Ansammlung von Maschinen. Die Verbindungen zwischen den Maschinen sind dabei automatisch durch die Portnamen gegeben: wir nehmen an, daß eine Nachricht, die über einen ausgehenden Port gesandt wurde, bei dem eingehenden Port *des gleichen Namens* ankommt. In einem Netzwerk kann zusätzlich eine Maschine

speziell ausgezeichnet werden, der sogenannte *Scheduler*. Dies ist eine Maschine, die immer dann aktiviert wird, wenn sonst keine aktiviert würde (z. B. wenn die zuletzt aktivierte keine Nachricht gesandt hat). Wir kennzeichnen diesen Scheduler dadurch, daß er einen speziellen eingehenden Port mit dem Namen `clk` hat.

Weiterhin reservieren wir noch die speziellen Portnamen `input` und `output` für Zwecke, die weiter unten beschrieben werden.

Definition 2.2 (Netzwerk)
Ein *Netzwerk* N ist eine Menge von Maschinen, die die folgenden Bedingungen erfüllt:

- Es gibt keine zwei Maschinen gleichen Namens, d. h. für alle $M_1 \neq M_2 \in N$ ist $name_{M_1} \neq name_{M_2}$
- Es gibt keine zwei eingehenden Ports mit gleichem Namen mit Ausnahme des speziellen Namens `input`, d. h. für alle $M_1 \neq M_2 \in N$ ist $in(M_1) \cap in(M_2) \subseteq \{\texttt{input}\}$.
- Es gibt keine zwei ausgehenden Ports mit gleichem Namen mit Ausnahme des speziellen Namens `output`, d. h. für alle $M_1 \neq M_2 \in N$ ist $out(M_1) \cap out(M_2) \subseteq \{\texttt{output}\}$.
- Es gibt keine Maschine, die einen ausgehenden Port mit dem Namen `clk` oder `input` hat, d. h. für alle $M \in N$ ist $\texttt{clk} \notin out(M)$ und $\texttt{input} \notin out(M)$.
- Es gibt keine Maschine, die einen eingehenden Port mit dem Namen `output` hat, d. h. für alle $M \in N$ ist $\texttt{output} \notin in(M)$.

Der strukturelle Begriff des Netzwerks allein ermöglicht es uns noch nicht, Aussagen über Ereignisse im Netzwerk zu treffen. Hierzu müssen wir zunächst definieren, was wir unter einer Ausführung eines Netzwerks verstehen. Wir stellen uns die Ausführung des Netzwerkes in etwa wie folgt vor (vgl. auch Abbildung 2.3):

- Zu Beginn der Ausführung wird der Scheduler aktiviert.

- Wann immer eine Maschine M aktiviert wurde, und eine Nachricht m über einen ausgehenden Port p geschickt hat, wird danach die Maschine M' aktiviert, welche den zugehörigen eingehenden Port p besitzt. Diese Maschine M' erhält die Nachricht m über den Port p.

- Wenn eine Maschine nach ihrer Aktivierung keine Nachricht sendet, oder eine Nachricht m über einen ausgehenden Port schickt, zu dem kein Gegenstück existiert, so wird wieder der Scheduler aktiviert (dieser erhält aber *nicht* die Nachricht m).

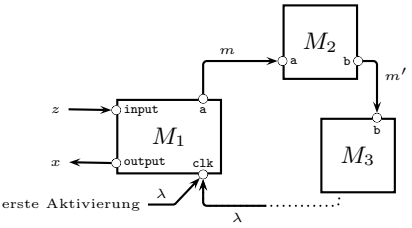


Abbildung 2.3.: Beispielhafter Überblick der Ausführung eines Netzwerks. Dargestellt ist ein Netzwerk $N = \{M_1, M_2, M_3\}$. Bei einer Ausführung mit z als Eingabe für M_1 wird zunächst M_1 mit z auf dem eingehenden Port `input` aktiviert. Danach wird – in der ersten richtigen Aktivierung – M_1 als Scheduler mit einer leeren Nachricht auf dem Port `clk` aktiviert. M_1 sendet daraufhin über den ausgehenden Port `a` eine Nachricht m, woraufhin M_2 mit m auf dem eingehenden Port `a` aktiviert wird. Dann sendet M_2 eine Nachricht m' auf dem ausgehenden Port `b`, worauf M_3 mit der Nachricht m' auf dem eingehenden Port `b` aktiviert wird. M_3 sendet keine Nachricht, was dazu führt, daß M_1 wieder mit leerer Nachricht über `clk` aktiviert wird. Zum Schluß gibt M_1 den Wert x aus, d. h. M_1 sendet x über den ausgehenden Port `output`.

- Die Zufallsvariable $run_{N,k}$ beschreibt den Protokollauf, d. h. es handelt sich hierbei um eine Folge, die alle gesandten Nachrichten und alle Zustandsübergänge der Maschinen enthält

Mit dieser Modellierung des Protokollaufs können wir bereits die Sicherheitsbegriffe formulieren, wie sie im Modell von [BPW04b] vorgestellt wurden. Einige Varianten (so z. B. die Sicherheitsbegriffe mit Auxiliary input, s. u., oder die Definition der einfachen und allgemeinen Komponierbarkeit, s. Def. 2.22 und 2.28) jedoch verlangen noch die Möglichkeit, über die Eingabe und die Ausgabe von einzelnen Maschinen zu sprechen. Die Ausgabe ist einfach zu modellieren: um eine Ausgabe zu machen, sendet eine Maschine eine Nachricht über den speziellen Port `output`, diese kann dann aus dem Protokollauf extrahiert werden.

Um die Idee einer Eingabe einzufangen, führen wir vor der ersten Aktivierung des Schedulers noch eine Phase in die Protokollausführung ein, in der die Eingaben verteilt werden. Dabei wird jede Maschine, die eine Eingabe x erhalten soll, mit dieser Eingabe auf dem eingehenden Port `input` aktiviert. Da diese Aktivierung nur zum Übergeben der Eingabe dient, darf die Maschine in dieser Aktivierung keine Nachricht senden.

Es ergibt sich die folgende Definition:

Definition 2.3 (Ausführung eines Netzwerks)

(Fortsetzung nächste Seite)

(Fortsetzung)

Sei N ein Netzwerk, $k \in \mathbb{N}$, $id_1, \ldots, id_n \in \Sigma^+$ eine Liste von paarweise verschiedenen Maschinennamen, und $x_1, \ldots, x_n \in \Sigma^*$ eine Liste von zugehörigen Eingaben. Weiter enthalte N eine Maschine mit eingehendem Port `clk`.

Dann ist die Zufallsvariable $run_{N,k}(id_1 \leftarrow x_1, \ldots, id_n \leftarrow x_n)$ als die (unendliche) Ausgabe des folgenden probabilistischen Algorithmus definiert:

0. *Initialisierung.* Zu Beginn seien die Variablen $s_{id} \in \Sigma^*$ (Zustand der Maschine id) mit λ (dem leeren Wort) initialisiert, $port := \texttt{clk}$ (d. h. der Scheduler, also die Maschine mit Port `clk`, ist – nach dem Verteilen der Eingaben – als erste zu aktivieren) und $msg := \lambda$ (bei der ersten Aktivierung ist die eingehende Nachricht leer).
1. *Verteilung der Eingaben.* Für $i = 1$ bis $i = n$ aktiviere die Maschine id_i mit Nachricht x_i auf Port `input`.
   Das heißt, wenn M die Maschine mit Namen $name_M = id_i$ ist und $\texttt{input} \in in(M)$, dann bezeichne $P := M(k, s_M, \texttt{input}, x_i)$ die Wahrscheinlichkeitsverteilung, die das Verhalten von M bei Sicherheitsparameter k, Zustand s_M und eingehender Nachricht über den Port `input` beschreibt. Wähle ein Tripel (s', p', m') gemäß der Verteilung P. Setze $s_M := s'$ (der Zustand von M wird aktualisiert). (Eine evtl. gesandte Nachricht m' wird ignoriert.)
   Existiert für ein i keine Maschine M mit Namen $name_M = id_i$ und $\texttt{input} \in in(M)$, so hat das Argument $id_i \leftarrow x_i$ keine Wirkung.
2. *Aktivierung einer Maschine.* Es sei M die Maschine, die den eingehenden Port $port$ besitzt, d. h. $port \in in(M)$. Es sei $P := M(k, s_M, port, msg)$ die Wahrscheinlichkeitsverteilung, die das Verhalten von M bei Sicherheitsparameter k, Zustand s_M und eingehender Nachricht msg über den Port $port$ beschreibt. Wähle dann (s', p', m') gemäß P und setze $s_M := s'$ (der Zustand von M wird aktualisiert).
3. *Ausgeben des Ereignisses.* Der Algorithmus gibt das Tupel $(name_M, s_M, port, msg, s', p', m')$ aus.
4. *Zuordnen der Nachricht.* Existiert eine Maschine M mit eingehendem Port p', d. h. $p' \in in(M)$ für ein $M \in N$, so setze $port := p'$ und $msg := m'$ (die Nachricht m' wird an den Port p' geschickt).
   Existiert keine solche Maschine M oder ist $p' = \lambda$, so setze $port := \texttt{clk}$ und $msg := \lambda$ (so daß der Scheduler als nächstes aktiviert wird).
5. Gehe zu Schritt 2.

Die Zufallsvariable $run_{N,k}(id_1 \leftarrow x_1, \ldots, id_n \leftarrow x_n)$ ist also die unendliche

(Fortsetzung nächste Seite)

(Fortsetzung)

Folge der in Schritt 3 ausgegebenen Tupel.

Die bislang vorgestellten Definitionen erlauben es uns nun auch zu spezifizieren, was wir unter einem Protokoll verstehen. Formal ist ein Protokoll nichts weiter als ein Netzwerk (bestehend aus Maschinen, die die Protokollparteien und die idealen Funktionalitäten repräsentieren), das aber noch nicht mit einer Umgebung oder einem Angreifer versehen ist, und auch keinen Scheduler enthält.

Gegenüber rohen Netzwerken haben Protokolle jedoch noch zusätzliche Struktur. So gibt es verschiedene Typen von Verbindungen. Zum einen gibt es interne Verbindungen, die zwei Maschinen im Protokoll verbinden (welche Parteien oder Funktionalitäten repräsentieren können). Darüber hinaus hat das Protokoll eine Schnittstelle, über die es seine Eingaben erhalten und seine Ausgaben liefern wird. Mit dieser Schnittstelle wird später die Umgebung verbunden. Und schließlich gibt es noch Verbindungen zum Angreifer. Diese können z. B. Verbindungen zwischen Funktionalität und Angreifer sein, die dazu dienen, akzeptable Imperfektionen des Protokolls zu modellieren, oder aber einfach unsichere Leitungen im Protokoll. Zusätzlich gibt es noch Leitungen zwischen Umgebung und Angreifer, diese sind insofern wichtig für die Definition von Protokollen, daß sie in Protokollen *nicht* vorkommen dürfen.

Um komfortabel zwischen den verschiedenen Typen unterscheiden zu können, führen wir folgende Konvention für die Portnamen ein: Ein Port heißt *intern*, wenn sein Name mit `int_` beginnt. Eine Verbindung zum Angreifer wird durch *Angreiferports* charakterisiert, deren Namen mit `adv_` beginnen. Die Kommunikation zwischen Umgebung und Angreifer geht über *Umgebungsports*, die mit `env_` beginnen. Die Ports schließlich, die die Schnittstelle zwischen Protokoll und Umgebung bilden, nennen wir *Protokollports*. Dabei unterscheiden wir *Protokolleingabeports* und *Protokollausgabeports*, die für zum Protokoll gehendende bzw. vom Protokoll kommende Verbindungen gedacht sind. Diese beginnen mit `in_` und `out_`.

Zusätzlich brachen wir noch die Begriffe der freien und der gebundenen Ports. Ein Port p ist *gebunden* (in einem konkreten Netzwerk), wenn es sowohl einen ein- als auch einen ausgehenden Port p gibt (ein gebundener Port kennzeichnet also eine Verbindung innerhalb des Netzwerks). Im Gegensatz dazu heißt ein Port p *frei*, wenn es nur einen ausgehenden oder einen eingehenden Port p gibt, nicht aber beides (ein freier Port stellt also eine potentielle Verbindung nach außen dar).

Definition 2.4 (Porttypen)
Ein Port $p \in \Sigma^+$ heißt

- *interner Port*, wenn p mit `int_` beginnt,
- *Angreiferport*, wenn p mit `adv_` beginnt,
- *Umgebungsport*, wenn p mit `env_` beginnt,
- *Protokolleingabeport*, wenn p mit `in_` beginnt,
- *Protokollausgabeport*, wenn p mit `out_` beginnt, und
- *Spezialport*, wenn $p \in \{\texttt{input}, \texttt{output}, \texttt{clk}\}$.

Es sei N ein Netzwerk.

Wir nennen $p \in \Sigma^+$ einen *gebundenen Port von* N, wenn p ein ausgehender Port von einer Maschine in N ist, und gleichzeitig ein eingehender Port von einer Maschine in N.

Wir nennen $p \in \Sigma^+$ einen *freien Port von* N, wenn p ein Port von N ist, aber weder ein Spezialport noch ein gebundener Port von N ist.

Damit können wir ein Protokoll einfach als Netzwerk ohne Angreifer oder Umgebung kennzeichnen, welches nur interne Ports, ausgehende Protokollausgabeports, eingehende Protokolleingabeports und freie Angreiferports enthält.

Definition 2.5 (Protokolle)
Ein Netzwerk π heißt *Protokoll*, wenn folgende Bedingungen erfüllt sind:

- π ist endlich.
- π enthält keine Maschine mit Namen `env` oder `adv`.
- π enthält nur interne, Angreifer-, Protokolleingabe- und Protokollausgabeports.
- Jeder Angreiferport von π ist ein freier Port.
- Die Menge $out(\pi)$ der ausgehenden Ports von π enthält keine Protokolleingabeports.
- Die Menge $in(\pi)$ der eingehenden Ports von π enthält keine Protokollausgabeports.

Dem Leser mag an dieser Stelle auffallen, daß wir keine Möglichkeit haben, Protokollteilnehmer als korrumpierbar oder unkorrumpierbar zu deklarieren. Tatsächlich sind alle Maschinen in einem Protokoll im Sinne der Definition 2.5 unkorrumpierbar. Um Korruption zu modellieren, gibt es zwei Möglichkeiten: In einem Korruptionsmodell, bei dem zu Protokollbeginn feststeht, welche Parteien korrumpiert sind (*statische Korruption*), betrachtet man einfach für jede Korruptionssituation ein eigenes Protokoll, bei dem die korrumpierten Parteien durch Verbindungen zum Angreifer ersetzt wurden und zeigt für jedes dieser

Protokolle Sicherheit (vgl. [BPW04b]). Wenn wir zulassen wollen, daß Parteien zur Laufzeit korrumpiert werden (*adaptive Korruption*), so modellieren wir einfach explizit die Möglichkeit, korrumpiert zu werden, in die Maschinen ein, es gibt also spezielle Nachrichten, die zur Korruption führen (vgl. [Can01]). Da wir für unsere Ergebnisse keine korrumpierten Parteien betrachten müssen, spezifizieren wir die möglichen Mechanismen der Korruption nicht weiter.

Definition 2.3 gibt uns eine Zufallsvariable $run_{N,k}(\dots)$, welche alle in der Ausführung des Netzwerks (oder Protokolls) auftretenden Ereignisse angibt. Zumeist sind wir jedoch nicht an der Gesamtheit dieser Ereignisse interessiert, sondern in einem Ausschnitt derselben. Besonders wichtige Zufallsvariablen sind zum einen die Ausgabe einer Maschine (insbesondere der Umgebung), zum anderen die Sicht der Umgebung. Um einer Maschine die Möglichkeit einer Ausgabe zu geben, haben wir weiter oben den speziellen Portnamen `output` eingeführt. Die Ausgabe einer Maschine ist dann die *erste* Nachricht, die diese Maschine über diesen speziellen Port sendet. Diese Ausgabe schreiben wir $\text{OUTPUT}_{N,k}(id_1 \leftarrow x_1, \dots, id_n \leftarrow x_n : id'_1, \dots, id'_m)$, womit wir das Tupel der Ausgaben der Maschinen $id'_1, \dots, id'_2$ meinen, wenn das Netzwerk N mit Sicherheitsparameter k läuft und die Maschinen id_i die Eingaben x_i erhalten. Da die Ausgabe der Umgebung in einem Protokoll ein wichtiger Spezialfall ist, definieren wir noch abkürzend $\text{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ als die Ausgabe der Umgebung, wenn sie mit dem Protokoll π und dem Angreifer $\mathcal{A}$ läuft, wobei der Sicherheitsparameter k und die Eingabe der Umgebung z sei.

Unter der Sicht einer Maschine M verstehen wir die Folge aller Ereignisse, die diese Maschine „zu Gesicht bekommt“, genauer alle ein- und ausgehenden Nachrichten und alle Zustände dieser Maschine. Da die Zufallsvariable $run_{N,k}(\dots)$ all diese Ereignisse auflistet, und jeweils mit dem Namen der betroffenen Maschine annotiert, ist die Sicht einer speziellen Maschine leicht daraus zu extrahieren. Wir schreiben dann $\text{VIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ für die Sicht der Umgebung in einer Ausführung des Protokolls π mit Angreifer $\mathcal{A}$ und Umgebung $\mathcal{Z}$, Sicherheitsparameter k und Auxiliary input z.

Die folgende Definition liefert die Details der soeben beschriebenen Konstruktionen:

Definition 2.6 (Sicht und Ausgabe)
Es sei $r := (name_i, s_i, p_i, m_i, s'_i, p'_i, m'_i)$ eine Folge von Tupeln (einen Protokollauf darstellend). Es sei $(name, s, p, m, s', p', m')$ das erste Tupel, das $name = id$ und $p' = \texttt{output}$ erfüllt. Dann ist $output_{id}(r) := m'$. Existiert

(Fortsetzung nächste Seite)

(Fortsetzung)

kein solches Tupel, so sei $output_{id}(r) := \perp$ (wobei $\perp$ ein nicht in Σ^* liegendes Symbol ist).

Es seien N, k, id_i, x_i und $run := run_{N,k}(id_1 \leftarrow x_1, \ldots, id_n \leftarrow x_n)$ (der Protokollauf von N) wie in Definition 2.3, sowie $id'_i \in \Sigma^+$. Dann ist $\text{OUTPUT}_{N,k}(id_1 \leftarrow x_1, \ldots, id_n \leftarrow x_n : id'_1, \ldots, id'_m) := \big(output_{id'_1}(run), \ldots, output_{id'_m}(run)\big)$.

Des weiteren schreiben wir abkürzend $\text{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) := \text{OUTPUT}_{\pi\cup\{\mathcal{A},\mathcal{Z}\},k}(\texttt{env} \leftarrow z : \texttt{env})$ (die Ausgabe der Maschine mit dem Namen `env` (die Umgebung), wenn diese initial die Eingabe z erhält).

Es sei wieder $r := (name_i, s_i, p_i, m_i, s'_i, p'_i, m'_i)$ eine Folge von Tupeln (einen Protokollauf darstellend). Dann sei r' die Teilfolge der Tupel mit $name_i =$ `env` (die zur Umgebung gehörigen).

Dann definieren wir $\text{VIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) := run'$, wobei $run := run_{\pi\cup\{\mathcal{A},\mathcal{Z}\},k}(\texttt{env} \leftarrow z)$ (die Sicht der Umgebung bei Eingabe z).

2.2.3. Komplexität von Maschinen

Im vorangegangenen Abschnitt haben wir definiert, was wir unter Maschinen und Protokollen verstehen. Diese Definitionen sind ausreichend für eine Definition der perfekten und der statistischen Sicherheit (siehe Abschnitt 2.2.4), doch um komplexitätstheoretische Sicherheit zu definieren, müssen wir noch angeben, was es bedeutet, daß eine Maschine in ihrer Rechenleistung beschränkt ist. Dem werden wir in diesem Abschnitt nachkommen. Zusätzlich definieren wir noch einige andere komplexitätsbezogene Begriffe für Maschinen, die später gebraucht werden.

Es gibt in der Kryptographie zwei Grundansätze, wie man definiert, daß ein Algorithmus oder eine Maschine polynomiell-beschränkt ist. Der erste Ansatz ist, eine Maschine polynomiell-beschränkt zu nennen, wenn ihre Laufzeit polynomiell ist in der Länge ihrer Eingaben. Der zweite Ansatz besteht darin, einen Sicherheitsparameter k anzunehmen und zu verlangen, daß die Laufzeit der Maschine in k polynomiell ist, unabhängig von der Länge der Eingabe. Es stellt sich heraus, daß der erste Ansatz im Zusammenhang mit interaktiven Maschinen Schwierigkeiten aufwirft. So ergibt sich zunächst die Frage, was man als Eingabe versteht. Ist damit die initiale Eingabe gemeint, also das, was eine Maschine zu Beginn eines Protokollaufs erhält? In diesem Fall treten Probleme auf, wenn die Rechenzeit einer Maschine von der Eingabelänge einer anderen abhängen soll (so ist z. B. schon bei einer einfachen Nachrichtenübermittlung der

Aufwand des Empfängers von der Länge der Nachricht, also von der Eingabe des *Senders* abhängig). Legt man andererseits fest, daß die Rechenzeit einer Maschine polynomiell beschränkt sein soll in der Gesamtlänge aller empfangenen Nachrichten, so entsteht ein Begriff, der nicht mehr der intuitiven Forderung an polynomiell-beschränkte Rechenzeit genügt: So können z. B. zwei Maschinen einander abwechselnd Nachrichten schicken, wobei jede doppelt so lang ist wie die vorhergehende. Da hier die Antwort auf eine Nachricht immer nur doppelt so groß ist wie die eingehende Nachricht, ist jede dieser Maschinen polynomiell in der eingehenden Nachrichtenlänge. Doch offensichtlich ist das gesamte Nachrichtenvolumen (und die gesamte Rechenzeit) exponentiell in der Anzahl der versandten Nachrichten. Diese Probleme sind zwar nicht unüberwindbar, so schlagen [HMQU05a, Can05, Küs06, HMQU06b] verschiedene Modellierungen des Polynomialzeitbegriffs vor, die auf obigem Ansatz basieren und dennoch den ebengenannten Effekt vermeiden. Doch führen diese Ansätze zu einer wesentlich komplexeren Modellierung, was der Idee zuwiderläuft, durch den Simulationsansatz eine möglichst einfache aber universelle Definition der Sicherheit zu erhalten.

Der Ansatz, die Laufzeit der Maschinen polynomiell im Sicherheitsparameter zu beschränken, wurde in [BPW04b, Can01] verfolgt und stellte sich als wesentlich einfacher heraus. Doch auch hier gibt es einige Fallen, die oft übersehen werden. Zum einen werden Protokolle und ideale Funktionalitäten meist in einer Form angegeben, die nicht polynomiell-beschränkt ist. So nimmt beispielsweise die ideale Funktionalität $\mathcal{F}_{\text{SMT}}$ für sichere Nachrichtenübertragung [Can01] eine beliebige Nachricht an und leitet sie an den Empfänger. Da keine Beschränkung der Länge vorliegt, ist die Laufzeit dieser Funktionalität nicht mehr durch eine Funktion im Sicherheitsparameter beschränkt. Ähnliches gilt auch für andere Funktionalitäten aus [Can01] wie $\mathcal{F}_{\text{RSIG}}$, $\mathcal{F}_{\text{SIG}}$ (digitale Signaturen), $\mathcal{F}_{\text{PKE}}$ (Public-Key-Verschlüsselung) und $\mathcal{F}_{\text{M-SMT}}$ (sichere Nachrichtenübertragung für mehrere Nachrichten). Obwohl diese Funktionalitäten (und die dazugehörigen Protokolle) formal gesehen nicht polynomiell-beschränkt sind, wird dieses Problem oft nicht als bedrohlich angesehen, da die Funktionalitäten in einem intuitiven Sinne eine „vernüftige“ Laufzeit haben. Doch für eine exakte Analyse von Sicherheit und Protokollen ist es unabdingbar, daß die betrachteten Objekte nicht nur ungefähr, sondern genau den Definitionen entsprechen. Daher muß man, will man Polynomialität in Abhängigkeit vom Sicherheitsparameter k definieren, ein willkürliches Polynom p wählen und verlangen, daß – z. B. im Falle der sicheren Nachrichtenübertragung – die Funktionalität höchstens $p(k)$ Nachrichten mit maximaler Länge $p(k)$ überträgt. Die gleiche Modifikation ist bei Protokollen durchzuführen. Dies ist insofern unglücklich, daß das Polynom p tatsächlich völlig willkürlich ist; üblicherweise ist jedes Polynom zulässig und die Wahl unabhängig von den Eigenschaften des Protokolls und der Funktionalität.

Das zweite Problem, das mit dieser Modellierung der Polynomialzeit auftritt,

ist noch subtiler. Betrachten wir z. B. die Funktionalität $\mathcal{F}_{\text{PKE}}$ für Public-Key-Verschlüsselung aus [Can01]: Hier steht (neben anderen Operationen) jeder Partei die Möglichkeit zur Verfügung, eine an sie gerichtete Nachricht zu entschlüsseln. Da die Funktionalität polynomiell-beschränkte Laufzeit haben muß, gibt es (für festen Sicherheitsparameter k) eine Anzahl n von Aufrufen der Entschlüsselungsoperation durch eine Partei A, nach der die Funktionalität spätestens terminiert.[6] Ruft nun A die Funktionalität n-mal auf, so wird eine andere Partei B feststellen, daß die Funktionalität nicht mehr reagiert. Deshalb muß nun auch ein diese Funktionalität sicher realisierendes Protokoll diese Eigenschaft haben. Es tritt also auch im realen Protokoll eine Art Informationsübertragung von A zu B auf. Dies impliziert, daß selbst eine Entschlüsselung Kommunikation zwischen den Parteien voraussetzt, was der Idee eines Verschlüsselungsverfahrens zuwiderläuft. Abhilfe für dieses Problem kann nicht geschaffen werden, indem man die Definition der Funktionalität ändert, man muß vielmehr bei der Definition polynomiell-beschränkter Laufzeit Rücksicht auf diesen Effekt nehmen. Eine Lösung (erstmalig in [Bac02b] vorgeschlagen) ist, einer Maschine zu erlauben, die Kommunikationsverbindung zu *einer* anderen Maschine zu trennen, ohne dabei die Verbindung zu anderen Maschinen zu unterbrechen. Dann könnte die Funktionalität $\mathcal{F}_{\text{PKE}}$ nach p Aufrufen der Entschlüsselungsoperation durch eine Partei A die Verbindung zu dieser Partei A trennen, aber mit den anderen Parteien weiter kommunizieren. Die gesamte Anzahl von Aufrufen der Entschlüsselungsoperation ist dann durch mp beschränkt, wenn m die Anzahl der Parteien ist. Also können wir die Funktionalität als polynomiell-beschränkt auffassen, und der oben genannte Effekt tritt dennoch nicht mehr auf, man kann also die Funktionalität durch ein Protokoll ohne Kommunikation realisieren [Can01].

Diese Betrachtungen zeigen, daß auch die Definition polynomiell-beschränkter Laufzeit relativ zum Sicherheitsparameter nicht unproblematisch ist, doch derzeit handelt es sich dabei um das bestverstandene Modell, welches wir im folgenden verwenden werden. Langfristig ist es natürlich wünschenswert, einen Begriff zu haben, der eine Abhängigkeit der Laufzeit von der Eingabelänge zuläßt und dennoch leicht durchschaubar ist.

Weiterhin ist zu klären, welches Maschinenmodell dem Laufzeitbegriff zugrundeliegen soll. So ist die Anzahl der Schritte, die ein Algorithmus braucht, sehr unterschiedlich, je nachdem ob man z. B. Turingmaschinen oder RAM-Maschinen zugrundelegt. Wenn wir aber nur an einer Definition von polynomiell-beschränkter Zeit und nicht an feineren Laufzeitklassen (z. B. linear-beschränkter Zeit) interessiert sind, so scheint es nach der erweiterten Church-Turing-These

[6] Hierbei ist es nicht wichtig, ob die Funktionalität auf diese Aufrufe antwortet, denn allein die Entscheidung, ob ein Aufruf ignoriert werden soll, braucht mindestens einen Rechenschritt.

unwichtig zu sein, welches konkrete Modell gewählt wird.[7] Es gibt allerdings Indizien, warum die Wahl des Maschinenmodells zumindest theoretisch einen Unterschied machen kann. Gegeben sei folgende Aufgabe: Eine Maschine bekomme eine Nachricht ungerader Länge l und soll das Symbol in der Mitte der Nachricht bestimmen. Offensichtlich braucht eine Turingmaschine $\Omega(l)$ Schritte für diese Aufgabe, während eine RAM-Maschine, die wahlfreien Zugriff auf die Nachricht hat, dies in $O(\log n)$ Schritten schaffen kann, indem sie zunächst per Binärsuche die Länge der Nachricht bestimmt und dann das mittlere Symbol direkt ausliest. Somit kann eine polynomielle RAM-Maschine diese Aufgabe z. B. für Nachrichtengrößen bis 2^k lösen (wenn k der Sicherheitsparameter ist), also insbesondere asymptotisch für jede polynomiell-beschränkte Länge, während eine Turingmaschine (selbst mit mehreren Bändern) dies nur für ein vorher festgelegtes Polynom kann (für jede Turingmaschine gibt es ein Polynom p, so daß die Turingmaschine das Problem für Nachrichten der Länge $p(k)$ nicht mehr löst). Wir folgen der Konvention und verwenden als Maschinenmodell Mehrband-Turingmaschinen (was äquivalent dazu ist, ein anderes vernüftiges Maschinenmodell im Sinne der erweiterten Church-Turing-These zu verwenden und dann Zugriff auf die Nachrichten nur sequentiell zuzulassen).

Wir geben nun eine Definition für die Laufzeitklasse einer Maschine an. Hierbei formulieren wir unsere Definition gleich etwas allgemeiner, um auch andere Klassen wie die der exponentiell-beschränkten Maschinen mit abzudecken. Zunächst müssen wir aber die folgende Definition vorausschicken:

Definition 2.7 (Implementierung einer Maschine)
Es sei eine Maschine M gegeben. Wir nennen ein Paar (T, L) bestehend aus einer probabilistischen Mehrband-Turingmaschine T und einer deterministischen Mehrband-Turingmaschine L mit Ein-Bit-Ausgabe eine *Implementierung* von M, wenn für jeden Sicherheitsparameter $k \in \mathbb{N}$, beliebige Zustände $s, s' \in \Sigma^*$, beliebige Nachrichten $m, m' \in \Sigma^*$ und Portnamen $p, q \in in(M)$, $p' \in out(M) \cup \{\perp\}$ gilt:

- Die Verteilungen $T(k, s, p, m)$ und $M(k, s, p, m)$ sind gleich (d. h. M führt das Programm T aus).

(Fortsetzung nächste Seite)

[7] Es sei denn natürlich, wir wollen auch quantenkryptographische Effekte betrachten. Denn zum einen scheinen Quantencomputer gewisse Probleme deutlich (d. h. mehr als polynomiell) schneller lösen zu können [Sho94], und zum anderen sind, wenn wir nicht nur das Rechnen, sondern auch Kommunikation mit Quantenzuständen zulassen, ganz neue Protokolle möglich, die Aufgaben lösen, die ohne Quantenkommunikation als unmöglich gezeigt werden können (vgl. z. B. [BB84, Yao95, MQ02, MQS03]). Um diese zu untersuchen sind natürlich spezielle Maschinen- und Kommunikationsmodelle notwendig. Eine Untersuchung solcher Sicherheitsmodelle findet sich in [Unr04c] und [BOM04].

(Fortsetzung)

- Ist $L(k,s,p) = 0$, so ordnet $M(k,s,p,m)$ dem Tupel (s,λ,λ) die Wahrscheinlichkeit 1 zu (d. h. gibt L an, daß der eingehende Port p geschlossen ist, so wird auf eine Nachricht über p nicht reagiert).
- Ist $L(k,s,q) = 0$ und $L(k,s',q) = 1$, so hat das Tupel (s',p',m') unter der Verteilung $M(k,s,p,m)$ die Wahrscheinlichkeit 0. (D. h. der Zustand s' kann von s aus nicht erreicht werden, wenn der Port q in s geschlossen und in s' geöffnet ist. In anderen Worten kann eine einmal getrennte Verbindung nicht wieder hergestellt werden.)

Hierbei nehmen wir an, daß eine Mehrband-Turingmaschine T Eingabe-, Ausgabe- und Arbeitsbänder hat, und daß $T(a,b,c,\dots)$ den Inhalt der Ausgabebänder bezeichnet, wenn T mit $a,b,c,\dots$ auf den Eingabebändern und leeren Ausgabe- und Arbeitsbändern aufgerufen ist. $T(a,b,c,\dots)$ ist also im Falle einer deterministischen Turingmaschine ein Tupel und im Falle einer probabilistischen eine Wahrscheinlichkeitsverteilung über Tupeln.

Wir sprechen von einer *nichtuniformen Implementierung* von M, wenn die Mehrband-Turingmaschinen T und L nichtuniforme Mehrband-Turingmaschinen sind. Dabei verstehen wir unter einer nichtuniformen Mehrband-Turingmaschine eine, die auf einem zusätzlichen Band eine nichtuniforme Eingabe erhält, die nur vom ersten Argument (hier dem Sicherheitsparameter k) abhängt.

Wir nennen eine Maschine *Turing-unbeschränkt*, wenn sie eine Implementierung hat, und *nichtuniform Turing-unbeschränkt*, wenn sie eine nichtuniforme Implementierung hat.

Eine Maschine läuft dann in Zeit f, wenn sie höchstens $f(k)$ Schritte läuft, wobei wir die Aktivierungen, die durch getrennte Verbindungen auftreten, nicht mitzählen. Außerdem soll die Überprüfung, ob eine Verbindung getrennt ist, auch in $f(k)$ Schritten möglich sein.

Definition 2.8 (Laufzeit einer Maschine)
Es sei f eine Funktion. Eine Maschine M *hat Laufzeit* f, wenn sie eine Implementierung (T,L) hat, so daß folgendes gilt:

(Fortsetzung nächste Seite)

(Fortsetzung)

- Für beliebige $k \in \mathbb{N}$, $s, p \in \Sigma^*$ läuft $L(k, s, p)$ höchstens $f(k)$ Schritte.
- Für $k \in \mathbb{N}$ und beliebige Folgen $m_i, p_i \in \Sigma^*$ führen wir $(s_i, p_i', m_i') := T(k, s_{i-1}, p_i, m_i)$ für $i = 1, \ldots$ aus mit $s_0 := \lambda$. Es bezeichne t die Summe der Schritte, die die Turingmaschine T in jenen Aktivierungen gelaufen ist, in denen $L(k, s_{i-1}, p_i) = 1$ war. Dann ist $t \leq p(k)$ mit Wahrscheinlichkeit 1.

Ist F eine Menge von Funktionen, so sagen wir *M habe Laufzeit in F*, wenn M Laufzeit f für ein $f \in F$ hat.

Eine Maschine ist *polynomiell-beschränkt*, wenn sie Laufzeit in $POLY$ hat, und *exponentiell-beschränkt*, wenn sie Laufzeit in EXP hat. Weiter nennen wir eine Maschine *beschränkt*, wenn sie Laufzeit f für irgendeine Funktion f hat.

Analog sprechen wir von *nichtuniformer Laufzeit* einer Maschine, wenn obiges mit einer nichtuniformen Implementierung gilt. Entsprechend definieren wir *nichtuniforme polynomiell-beschränkte nichtuniforme exponentiell-beschränkte* und *nichtuniforme beschränkte* Maschinen.

Im weiteren werden wir neben der Rechenzeit, die eine Maschine verbraucht, auch noch eine weitere Kenngröße in Betracht ziehen müssen, die Kommunikationskomplexität. Wir sagen, eine Maschine habe Kommunikationskomplexität f, wenn sie höchstens f Nachrichten pro Port schickt und liest, jede gesandte Nachricht höchstens Länge f hat, und von jeder Nachricht höchstens ein Präfix der Länge f gelesen wird.[8] Diese Eigenschaft ist erfreulicherweise etwas einfacher zu beschreiben als die Laufzeit, da wir keine Rücksicht auf die internen Vorgänge in der Maschine nehmen müssen (d. h. wir müssen nicht den Umweg über die Implementierung nehmen).

[8] Eine Maschine mit Kommunikationskomplexität f kann also bis zu $f^2 n$ Symbole senden, wenn n die Anzahl der Ports ist. Obwohl es natürlicher wäre, bei Kommunikationskomplexität f insgesamt nur f Symbole senden zu dürfen, verwenden wir die vorliegende Definition der größeren Einfachheit halber. Für Begriffe wie polynomiell-beschränkte Kommunikationskomplexität macht dies keinen Unterschied, und eine feinere Untergliederung der Komplexitätshierarchie werden wir nicht betrachten.

Definition 2.9 (Kommunikationskomplexität einer Maschine)
Es sei M eine Maschine und f eine Funktion. Wir sagen, *M habe Kommunikationskomplexität f*, wenn für beliebige Folgen von eingehenden Nachrichten und Ports die folgenden Punkte mit Wahrscheinlichkeit 1 erfüllt sind:

(i) Für jeden ausgehenden Port p gilt: Die Maschine M schickt höchstens $f(k)$ Nachrichten über p.

(ii) Für jeden eingehenden Port p gilt: Wenn die Maschine M zum i-ten Mal mit $i > f(k)$ (k ist der Sicherheitsparameter) über Port p aktiviert wird, dann wird die Nachricht ignoriert, d. h. der Zustand der Maschine nach Aktivierung ist der gleiche wie vor der Aktivierung, und es wird keine Nachricht gesandt.

(iii) Jede von M gesandte Nachricht hat eine Länge von höchstens $f(k)$.

(iv) In jedem Zustand von M führen zwei eingehende Nachrichten m_1, m_2, die auf den ersten $f(k)$ Symbolen gleich sind, zu der gleichen Verteilung von ausgehender Nachricht, Port und Zustand nach der Aktivierung. D. h. für alle k, s, p, m_1, m_2, wobei m_1 und m_2 das gleiche Präfix der Länge $f(k)$ haben, ist $M(k, s, p, m_1) = M(k, s, p, m_2)$.

Sind die Bedingungen (i) und (iii) erfüllt, so sprechen wir von *ausgehender Kommunikationskomplexität*.

Eine Maschine *M hat Kommunikationskomplexität in F*, wenn sie für ein $f \in F$ Kommunikationskomplexität f hat. Eine Maschine *hat beschränkte Kommunikationskomplexität*, wenn sie Kommunikationskomplexität f für irgendeine Funktion f hat. Eine Maschine *hat polynomiell-beschränkte Kommunikationskomplexität*, wenn sie Kommunikationskomplexität in $POLY$ hat.

Gewappnet mit diesen Definitionen können wir nun auch ein Netzwerk oder Protokoll mit beschränkter Laufzeit oder Kommunikationskomplexität definieren:

Definition 2.10 (Komplexität von Netzwerken und Protokollen)
Ein Netzwerk oder Protokoll *N hat Laufzeit/Kommunikationskomplexität f/in F*, wenn jede Maschine $M \in N$ Laufzeit/Kommunikationskomplexität f/in F hat.

Analog definieren wir *(nichtuniform) beschränkte/polynomiell-beschränk-*

(Fortsetzung nächste Seite)

(Fortsetzung)

te/exponentiell-beschränkte/Turing-unbeschränkte Netzwerke und Protokolle, sowie Netzwerke und Protokolle mit *beschränkter/polynomiell-beschränkter Kommunikationskomplexität.*

Diese Definition hat den Nachteil, daß ein Netzwerk N mit Laufzeit f *pro Maschine* diese Laufzeit zur Verfügung hat, die summierte Laufzeit also bis zu $\#N \cdot f \in O(f)$ sein kann. Da wir aber keine so feinen Unterscheidungen der Laufzeit treffen werden, ist dies für unsere Zwecke akzeptabel und hat den Vorteil einer einfacheren Handhabbarkeit.

Im Zusammenhang mit beschränkten Maschinen ist es noch notwendig, den Begriff der Sicht einer Maschine aus Definition 2.6 leicht abzuändern. Dies hat zwei Gründe: Zum einen wollen wir in der Lage sein, von komplexitätstheoretisch ununterscheidbaren Sichten zu sprechen, und dies ist nur möglich, wenn die Sicht als Wort kodiert ist. Zum anderen haben wir bei der Definition der Implementierung einer Maschine zugelassen, daß eine Maschine Verbindungen „trennt“ und somit Nachrichten auf diesen Verbindungen ignoriert. Diese ignorierten Nachrichten treten aber in der Sicht nach Definition 2.6 auf. Wir brauchen deshalb den Begriff der komplexitätstheoretischen Sicht, in der diese Einträge nicht vorkommen.

Definition 2.11 (Komplexitätstheoretische Sicht)
Es sei $f : \Sigma \cup \{stop\} \to \Sigma^*$ eine injektive Abbildung, so daß $f(x)$ nur dann Präfix von $f(y)$ ist, wenn $x = y$. Weiter sei $f : \Sigma^* \to \Sigma^*$ definiert durch $f(x_1 \dots x_n) := f(x_1) \dots f(x_n) f(stop)$. (Es bezeichne hierbei ab die Konkatenation von a und b).

Für $x_i \in \Sigma^*$ sei dann $f(x_1, \dots, x_n) := f(x_1) \dots f(x_n)$. Für eine Folge v sei v' die (möglicherweise endliche) Teilfolge von v, die durch Streichen der Glieder v_i mit $v_i = v_{i-1}$ entsteht. Für eine Folge v von Tupeln sei $f(v) := f(v'_1) f(v'_2) \dots$.[9]

Dann ist die *komplexitätstheoretische Sicht* $\mathrm{CVIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ als $f(\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k, z))$ definiert.

[9] Natürlich ist die Wahl der Funktion f sehr willkürlich. Wir geben aber diese konkrete Funktion an, damit ihre Eigenschaften (wie z. B. effiziente Invertierbarkeit) klar sind.

2.2.4. Die Sicherheitsdefinition

In diesem Abschnitt stellen wir die Definition eines sicheren Protokolls vor. Dazu geben wir an, was es bedeutet, daß ein Protokoll π so sicher ist wie ein anderes Protokoll ρ. Hierbei wird es sich herausstellen, daß viele verschiedene Varianten der Definition existieren, selbst wenn man sich bereits auf ein Maschinen- und Netzwerkmodell festgelegt hat.

2.2.4.1. Allgemeine Sicherheit

Wie bereits in Abschnitt 2.2.1 erwähnt, ist die Grundidee, π als so sicher wie ρ, also als Implementation von ρ zu bezeichnen, wenn keine Umgebung $\mathcal{Z}$ zwischen einem Angriff eines Angreifers $\mathcal{A}$ auf das Protokoll π und einem Angriff des zugehörigen Simulators $\mathcal{S}$ auf das ideale Protokoll ρ unterscheiden kann. Hierzu ist es zunächst nötig zu klären, welche Umgebungen, Angreifer und Simulatoren strukturell überhaupt zulässig sind. So darf z. B. der Angreifer nur auf die Angreiferports des Protokolls und die Umgebung nur auf die Protokollports zugreifen. Diese Definition kann beim ersten Lesen getrost übersprungen werden.

Definition 2.12 (Zulässige Angreifer, Umgebungen und Simulatoren)
Es seien π und ρ Protokolle.

Wir nennen eine Maschine $\mathcal{A}$ einen *zulässigen Angreifer* (für π), wenn gilt:

- $name_{\mathcal{A}} = \texttt{adv}$ (der Angreifer hat den korrekten Maschinennamen),
- $\mathcal{A}$ hat einen eingehenden Port `clk` (der Angreifer ist der Scheduler).
- $\mathcal{A}$ hat nur spezielle, Angreifer- und Umgebungsports.
- $\pi \cup \{\mathcal{A}\}$ ist ein Netzwerk (es treten keine Namenskonflikte auf).

Wir nennen einen Simulator $\mathcal{S}$ *zulässig* (für ρ und $\mathcal{A}$), wenn gilt:

- $\mathcal{S}$ ist ein zulässiger Angreifer für ρ.
- $\mathcal{S}$ hat die gleichen eingehenden Umgebungsports wie $\mathcal{A}$ und die gleichen ausgehenden Umgebungsports wie $\mathcal{A}$.

Eine Umgebung $\mathcal{Z}$ heißt *zulässig* (für π und einen Angreifer $\mathcal{A}$) , wenn gilt:

- $name_{\mathcal{Z}} = \texttt{env}$ (die Umgebung hat den korrekten Maschinennamen),
- $\pi \cup \{\mathcal{A}, \mathcal{Z}\}$ ist ein Netzwerk (es treten keine Namenskonflikte auf).
- $\mathcal{Z}$ hat nur spezielle, Umgebungs-, ausgehende Protokolleingabe- und eingehende Protokollausgabeports.

Da wir nur zulässige Angreifer, Umgebungen und Simulatoren betrachten, lassen wir die Qualifikation „zulässig“ oft weg, wenn sich diese aus dem Zusammenhang

versteht.

Man beachte, wenn $\mathcal{A}$ ein zulässiger Angreifer für π, $\mathcal{S}$ ein zulässiger Simulator für ρ und $\mathcal{A}$, und $\mathcal{Z}$ eine zulässige Umgebung für π und $\mathcal{A}$ ist, dann ist $\mathcal{Z}$ auch zulässig für ρ und $\mathcal{S}$.

Nun können wir die erste Sicherheitsdefinition formulieren:

Definition 2.13 (Perfekte allgemeine Sicherheit)
Es seien π und ρ Protokolle. Dann ist π so sicher wie ρ bezüglich *perfekter allgemeiner Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung*, wenn für jeden zulässigen Angreifer $\mathcal{A}$ ein zulässiger Simulator $\mathcal{S}$ existiert, so daß für jede zulässige Umgebung $\mathcal{Z}$, jedes $k \in \mathbb{N}$ (dem Sicherheitsparameter) und jedes $z \in \Sigma^*$ (dem Auxiliary input) gilt:

$$\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) = \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z),$$

also daß diese beiden Zufallsvariablen die gleiche Verteilung haben.

Dabei ist $\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ die Ausgabe von $\mathcal{Z}$ bei einer Ausführung von π zusammen mit $\mathcal{A}$ als Angreifer und $\mathcal{Z}$ als Umgebung bei Sicherheitsparameter k, wenn $\mathcal{Z}$ den Auxiliary input z bekommt (nach Definition 2.6), und analog $\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)$ die Ausgabe von $\mathcal{Z}$ bei einer Ausführung von ρ mit dem Simulator $\mathcal{S}$ als Angreifer und $\mathcal{Z}$ als Umgebung.

Wir sagen, π implementiere ρ bezüglich perfekter allgemeiner Sicherheit *ohne Auxiliary input*, wenn in obiger Definition über $z \in \{\lambda\}$ statt $z \in \Sigma^*$ quantifiziert wird.

Wir sagen, π implementiere ρ bezüglich perfekter allgemeiner Sicherheit mit/ohne Auxiliary input *bzgl. der Sicht der Umgebung*, wenn in obiger Definition die Zufallsvariablen $\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ und $\mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)$ verglichen werden (statt $\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ und $\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)$).

Hierbei bezeichnet $\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ die gesamte Sicht von $\mathcal{Z}$ in einem Protokollauf mit π und $\mathcal{A}$ (nach Definition 2.6), und analog $\mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)$.

Fügen wir dem Sicherheitsbegriff die zusätzliche Angabe *mit Angreifern in C* hinzu, so quantifizieren wir über alle zulässigen Angreifer $\mathcal{A} \in C$ und nicht über alle zulässigen Angreifer $\mathcal{A}$. Analoges gilt für die Angaben *mit Simulatoren in C* und *mit Umgebungen in C*.

Zu dieser Definition sind einige Bemerkungen angebracht:

- Zunächst einmal fällt auf, daß wir explizit die Mengen angeben können, denen Angreifer, Simulator und Umgebung entstammen. Diese zusätzliche Allgemeinheit werden wir selten nutzen, doch brauchen wir sie, um die Ergebnisse in Kapitel 7 formulieren zu können. In den meisten Fällen jedoch können wir uns mit dem Standardfall begnügen, der diesen Entitäten keine Einschränkungen auferlegt. (Wir werden die Kompositionstheoreme auch nur für diesen Fall angeben. In anderen Fällen ist jeweils zu prüfen, ob die Klassen die in den Beweisen implizit benutzten Abgeschlossenheitseigenschaften besitzen.)

- Weiter bemerken wir, daß zwei Varianten der Sicherheit angegeben wurden, eine mit und eine ohne Auxiliary input. Dies liegt daran, daß wir unsere Ergebnisse so präsentieren wollen, daß sie sowohl im Modell von Backes, Pfitzmann und Waidner [BPW04b] als auch im Modell von Canetti [Can01] gelten. Da ersteres eine Definition ohne Auxiliary input vorschlägt, während das zweite mit formuliert ist, haben wir deshalb beide Fälle in unserer Definition abgedeckt.

  Die Motivation hinter dem Auxiliary input ist vor allem durch historische Erfahrung bedingt. Wie in Abschnitt 2.1.1 angedeutet, hat sich herausgestellt, daß Sicherheitsmodelle mit Auxiliary input meist bessere Kompositionseigenschaften haben (so erlaubte Zero-Knowledge ohne Auxiliary input nicht einmal sequentielle Komposition [GK96b]). Aus dieser Erfahrung heraus macht es Sinn, Protokolle als selbst unter Annahme eines Auxiliary inputs sicher zu zeigen. Doch kann dies bei manchen Protokollkonstruktionen eine echte Einschränkung sein, denn manche Komplexitätsannahmen sind prinzipiell unmöglich in Präsenz eines Auxiliary inputs.[10] Da aber alle in Abschnitt Abschnitt 2.3 vorgestellten Kompositionstheoreme unabhängig davon gelten, ob ein Auxiliary input vorhanden ist oder nicht, ist der Sicherheitsbegriff ohne Auxiliary input sicher ebenfalls keine schlechte Wahl, und ein Protokoll, das zwar nur ohne Auxiliary input sicher ist, aber ansonsten viele Vorteile hat, ist einem mit Auxiliary input sicheren Protokoll oft vorzuziehen.

- Den obigen Sicherheitsbegriff gibt es in zwei Varianten, einmal bzgl. der Ausgabe der Umgebung und einmal bzgl. der Sicht der Umgebung (welche

[10] Ein Beispiel hierfür ist die Kollisionsresistenz eine Hashfunktion f. Diese besagt, daß von einer polynomiell-beschränkten Maschine keine zwei verschiedenen Urbilder x_1, x_2 gefunden werden können, die eine Kollision bilden, d. h. für die $f(x_1) = f(x_2)$ gilt. (Genaugenommen hängt dabei f auch noch vom Sicherheitsparameter k ab.) Da der Auxiliary input eine solche Kollision enthalten darf, ist die Annahme der Kollisionsresistenz einer einzelnen Hashfunktion in Präsenz eines Auxiliary inputs unsinnig. (Nicht aber die Annahme einer kollisionsresistenten *Familie* von Hashfunktionen im Sinne von Definition 1.8.)

die Ausgabe enthält). Es wird meist angenommen, daß diese Begriffe zusammenfallen, da die Umgebung o. B. d. A. ihre Sicht ausgeben kann und deshalb über die Ausgabe der Umgebung genauso gut unterschieden werden kann wie über die Sicht. Tatsächlich ist die Situation jedoch nicht so einfach. So zeigen wir zwar, daß die beiden Begriffe im Falle der perfekten Sicherheit zusammenfallen (Kapitel 3), und für die meisten Protokolle auch im Falle der statistischen und komplexitätstheoretischen Sicherheit (Lemma A.10). Doch in Abschnitt 3.2.1 geben wir ein Beispiel an, das zumindest im statistischen Fall die beiden Begriffe trennt.

- Die obige Variante der simulierbaren Sicherheit wurde als *allgemeine* Sicherheit bezeichnet. Dies geschah zur Abgrenzung vom Begriff der *speziellen* Sicherheit, der weiter unten vorgestellt wird. Die beiden Varianten unterscheiden sich nur in der Reihenfolge der Quantoren; bei der speziellen Sicherheit darf der Simulator von der Umgebung abhängen.

- Schließlich fällt auf, daß wir verlangt haben, daß die Zufallsvariablen, die die Ausführung des realen bzw. des idealen Protokolls beschreiben, exakt die gleiche Verteilung haben sollen (daher der Name „*perfekte* Sicherheit"). Dies ist eine sehr strenge Forderung und nur in den seltensten Fällen erfüllbar. Aus diesem Grund geben wir nun eine weitere Variante der Sicherheit an, bei der wir erlauben, daß die Zufallsvariablen leicht unterschiedlich sind.

Definition 2.14 (Statistische allgemeine Sicherheit)
Es seien π und ρ Protokolle. Dann ist π so sicher wie ρ bezüglich *statistischer allgemeiner Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung*, wenn für jeden zulässigen Angreifer $\mathcal{A}$ ein zulässiger Simulator $\mathcal{S}$ existiert, so daß für jede zulässige Umgebung $\mathcal{Z}$ die Familien

$$\left\{\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\right\}_{k,z} \quad \text{und} \quad \left\{\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)\right\}_{k,z}$$

statistisch ununterscheidbar sind (mit $k \in \mathbb{N}, z \in \Sigma^*$).

Statistische allgemeine Sicherheit *ohne Auxiliary input* ist analog zur perfekten allgemeinen Sicherheit ohne Auxiliary input (Definition 2.13) definiert, d. h. die Familien von Zufallsvariablen werden über $z \in \{\lambda\}$ statt $z \in \Sigma^*$ indiziert.

Analog wie bei der perfekten allgemeinen Sicherheit definieren wir auch statistische allgemeinen Sicherheit *bzgl. der Sicht der Umgebung* und *mit Angreifern, Simulatoren* oder *Umgebungen in C*.

Aber auch diese Definition ist noch sehr streng, denn Protokolle, die auf Komplexitätsannahmen basieren, können sie nicht erfüllen. Um solche Protokolle zuzulassen, müssen der Angreifer und die Umgebung in ihrer Rechenkapazität beschränkt werden. Beschränkt man den Angreifer und die Umgebung, so muß auch der Simulator beschränkt werden (dies ist nötig, um die Komponierbarkeit nicht zu verlieren, denn die Beweise von Lemma 2.20 und Theorem 2.26 benutzen diese Eigenschaft). Allerdings ist auch dies noch nicht genug. Zusätzlich brauchen wir noch, daß die Ausgabe der Umgebung im realen und idealen Modell komplexitätstheoretisch ununterscheidbar ist (statt der strengeren statistischen Ununterscheidbarkeit). Dies wird in der folgenden Definition realisiert:

Definition 2.15 (Komplexitätstheoretische allgemeine Sicherheit) Es seien π und ρ Protokolle. Dann ist π so sicher wie ρ bezüglich *komplexitätstheoretischer allgemeiner Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung*, wenn für jeden zulässigen *polynomiell-beschränkten* Angreifer $\mathcal{A}$ ein zulässiger *polynomiell-beschränkter* Simulator $\mathcal{S}$ existiert, so daß für jede zulässige *polynomiell-beschränkte* Umgebung $\mathcal{Z}$ die Familien

$$\left\{\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\right\}_{k,z} \quad \text{und} \quad \left\{\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)\right\}_{k,z}$$

komplexitätstheoretisch ununterscheidbar sind (mit $k \in \mathbb{N}, z \in \Sigma^*$).

Komplexitätstheoretische allgemeine Sicherheit *ohne Auxiliary input* ist analog wie bei der perfekten und statistischen allgemeinen Sicherheit definiert (vgl. Definition 2.13).

Komplexitätstheoretische allgemeine Sicherheit *bzgl. der Sicht der Umgebung* ist analog wie bei der perfekten und statistischen allgemeinen Sicherheit definiert, nur daß die komplexitätstheoretische Sicht CVIEW aus Definition 2.11 statt der Sicht VIEW verwendet wird.

Bei komplexitätstheoretischer allgemeiner Sicherheit *mit Angreifern in C* quantifizieren wir über alle zulässigen Angreifer in C (und nicht nur über alle zulässigen polynomiell-beschränkten Angreifer in C). Analoges gilt für Simulatoren und Umgebungen.

Die angegebenen Definitionen enthalten eine gewisse Redundanz. So kann man z. B. die Varianten mit Auxiliary input auch charakterisieren, indem man in der Variante ohne Auxiliary input nichtuniforme Umgebungen (also Umgebungen, die ihren eigenen Auxiliary input haben) zuläßt. Da im Falle der perfekten und der statistischen Sicherheit die Umgebungen sowieso unbeschränkt sind, fallen hier also Sicherheit mit und ohne Auxiliary input zusammen (solange keine zusätzliche Einschränkung der Umgebungen vorliegt), siehe Lemma A.2.

Weiterhin kann man einsehen, daß es nicht nötig ist, Umgebungen mit mehr als einem Bit Ausgabe zuzulassen, da man den Unterscheider aus Definition 1.4 bzw. die unterscheidende Menge aus Definition 1.2 in die Umgebung integrieren kann (Lemma A.1). Tatsächlich geht die Definition von [Can01] auch nur von solchen Umgebungen aus. Da statistische und komplexitätstheoretische Ununterscheidbarkeit auf endlichen Mengen zusammenfallen (Lemma 1.5 (iii)), kann man dann auch die komplexitätstheoretische allgemeine Sicherheit als statistische allgemeine Sicherheit (Definition 2.14) mit polynomiell-beschränkten Angreifern, Simulatoren und Umgebungen mit Ein-Bit-Ausgabe charakterisieren (Lemma A.3).

Aus folgenden Gründen haben wir darauf verzichtet, weniger redundante Definitionen zu präsentieren:

- Bei den Varianten der speziellen Sicherheit (s. u.) fallen Sicherheit mit Auxiliary input und Sicherheit bezüglich nichtuniformen Umgebungen nicht notwendig zusammen. Analoges gilt für Umgebungen mit Ein-Bit-Ausgabe (vgl. aber auch Korollar 7.20).

  Um die Definitionen einheitlich zu halten, haben wir deshalb auch im Falle der allgemeinen Sicherheit die Varianten mit und ohne Auxiliary input, sowie Umgebungen mit beliebiger Ausgabe zugelassen.

- Da im Falle der speziellen Sicherheit die Umgebung i. a. nicht auf Ein-Bit-Ausgabe beschränkt werden darf, können wir die komplexitätstheoretische Sicherheit in diesem Fall nicht als Spezialfall der statistischen betrachten. Da wir also in diesem Fall sowieso eine eigene Definition für die komplexitätstheoretische Sicherheit benötigen, haben wir der Einheitlichkeit halber auch bei der allgemeinen Sicherheit eine angegeben.

- Statistische allgemeine Sicherheit mit und ohne Auxiliary input fallen zwar zusammen, doch gilt dies i. a. nicht, wenn wir statt unbeschränkten Maschinen nur Turing-unbeschränkte Maschinen zulassen (also Maschinen, die durch eine Turingmaschine ohne Laufzeitbeschränkung realisiert werden können). Da auch diese Sichtweise verbreitet ist, haben wir die Definitionen so gewählt, daß auch dieser Fall betrachtet werden kann.

2.2.4.2. Spezielle Sicherheit

Wie bereits im vorangegangenen Abschnitt erwähnt, ist eine Eigenart der bislang angegebenen Sicherheitsdefinitionen, daß der Simulator nicht von der Umgebung abhängen darf. Für diese Definitionen haben wir den Namen der *allgemeinen* Sicherheit gewählt (ein genauerer aber unhandlicherer Name wäre beispielsweise *Sicherheit mit universellem Simulator*). Variiert man die Definitionen insofern,

daß der Simulator nun von der Umgebung abhängen darf (also eine bloße Vertauschung zweier Quantoren), erhält man weitere Definitionen, die wir unter dem Namen der *speziellen* Sicherheit zusammenfassen (genauer wäre *Sicherheit mit speziellem Simulator*):

Definition 2.16 (Spezielle Sicherheit)
Es seien π und ρ Protokolle.

Dann ist π so sicher wie ρ bezüglich *perfekter spezieller Sicherheit mit Auxiliary input*, wenn für jeden zulässigen Angreifer $\mathcal{A}$ und jede zulässige Umgebung $\mathcal{Z}$ ein zulässiger Simulator $\mathcal{S}$ existiert, so daß für jedes $k \in \mathbb{N}$ (dem Sicherheitsparameter) und jedes $z \in \Sigma^*$ (dem Auxiliary input) gilt:

$$\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) = \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z),$$

also daß diese beiden Zufallsvariablen die gleiche Verteilung haben.

Das Protokoll π ist so sicher wie ρ bezüglich *statistischer spezieller Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung*, wenn für jeden zulässigen Angreifer $\mathcal{A}$ und jede zulässige Umgebung $\mathcal{Z}$ ein zulässiger Simulator $\mathcal{S}$ existiert, so daß die Familien

$$\Big\{\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\Big\}_{k,z} \quad \text{und} \quad \Big\{\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)\Big\}_{k,z}$$

statistisch ununterscheidbar sind (mit $k \in \mathbb{N}, z \in \Sigma^*$).

Das Protokoll π ist so sicher wie ρ bezüglich *komplexitätstheoretischer spezieller Sicherheit mit Auxiliary input*, wenn für jeden zulässigen *polynomiell-beschränkten* Angreifer $\mathcal{A}$ und jede zulässige *polynomiell-beschränkte* Umgebung $\mathcal{Z}$ ein zulässiger *polynomiell-beschränkter* Simulator $\mathcal{S}$ existiert, so daß die Familien

$$\Big\{\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\Big\}_{k,z} \quad \text{und} \quad \Big\{\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)\Big\}_{k,z}$$

komplexitätstheoretisch ununterscheidbar sind (mit $k \in \mathbb{N}, z \in \Sigma^*$).

Die Varianten der speziellen Sicherheit *ohne Auxiliary input*, *bzgl. der Sicht der Umgebung* und *mit Angreifern, Simulatoren* oder *Umgebungen in C* sind analog zu den entsprechenden Varianten der allgemeinen Sicherheit definiert (vgl. Definitionen 2.13, 2.14 und 2.15).

Man beachte, daß die Argumente, die zu den nach Definition 2.15 genannten Äquivalenzen verschiedener Varianten der allgemeinen Sicherheit geführt haben, im Falle der speziellen Sicherheit nicht mehr gültig sind. Einige (aber bei weitem nicht alle) der Äquivalenzen jedoch lassen sich jedoch auch im Falle der speziellen Sicherheit mit anderen Methoden zeigen, vgl. z. B. Satz 3.8 und Korollar 7.20.

2.3. Komposition

Im vorliegenden Abschnitt stellen wir verschiedene Varianten der Komposition und Komponierbarkeit vor. Komposition meint hierbei eine Operation, die aus mehreren Protokollen (oder mehreren Instanzen eines Protokolls) ein neues Protokoll zusammensetzt. Komponierbarkeit bezeichnet die Eigenschaft eines Protokolls, unter der Kompositionsoperation seine Sicherheit nicht zu verlieren. Manchmal wird auch eine Sicherheitsdefinition selbst als komponierbar bezeichnet, damit ist üblicherweise gemeint, daß alle Protokolle, die dieser Definition genügen, komponierbar sind.

Wir stellen im folgenden drei Typen der Komposition vor: die einfache Komposition, die nebenläufige Komposition und die allgemeine Komposition. Bei der einfachen Komposition wird ein Protokoll in ein größeres eingebettet. Dabei darf das umgebende Protokoll nur eine Instanz des eingebetteten aufrufen. Bei der nebenläufigen Komposition werden mehrere Instanzen des Protokolls gleichzeitig ausgeführt. Bei der allgemeinen Komposition schließlich wird ähnlich wie bei der einfachen ein Protokoll in ein anderes eingebettet. Jedoch darf das umgebende Protokoll mehrere Instanzen des eingebetteten gleichzeitig ausführen. Sowohl die nebenläufige als auch die allgemeine Komposition können darüber parametrisiert werden, wie viele Instanzen maximal ausgeführt werden dürfen.

Man mag an dieser Stelle die Frage stellen, warum drei Typen der Komposition betrachtet werden müssen. Ist nicht die allgemeine Komposition hinlänglich, evtl. beschränkt auf eine Instanz des eingebetteten Protokolls. In der Tat wird z. B. in [Lin03] nur die allgemeine Komponierbarkeit untersucht, mit der Unterscheidung in $O(1)$-beschränkte und polynomiell-beschränkte allgemeine Komponierbarkeit (d. h. eine konstant- bzw. polynomiell-beschränkte Anzahl von Instanzen wird zugelassen). In der Tat kann man auch die Ergebnisse unserer Arbeit allein mit diesem Begriff formulieren. Da aber die allgemeine Komposition in natürlicher Weise dargestellt werden kann als die Hintereinanderausführung der einfachen und der nebenläufigen Komposition, können wir die einfache und die nebenläufige Komponierbarkeit getrennt betrachten und erhalten dadurch genauere Einsichten darüber, in welcher Weise die allgemeine Komponierbarkeit scheitert. Darüber hinaus scheint es auch definitorisch einfacher, zunächst die Definitionen der einfachen und der nebenläufigen Komposition anzugeben, um dann die allgemeine als Kombination der beiden anderen zu definieren.

2.3.1. Einfache Komposition

Bei der einfachen Komposition betrachten wir ein umgebendes Protokoll σ und ein eingebettetes Protokoll π. Ziel der Kompositionsoperation ist es nun, ein komponiertes Protokoll σ^π zu konstruieren, bei dem die Maschinen in σ die entsprechenden Maschinen aus π als Subprotokoll aufrufen. Da das umgebende Protokoll σ nicht mit den Protokollinterna von π konfrontiert werden darf, soll es σ nur möglich sein, Verbindungen zu den Protokollports von π aufzubauen. Weiterhin soll das komponierte Protokoll nach außen hin (d. h. zur Umgebung) weiterhin so aussehen wie σ. Also darf das komponierte Protokoll nur die Protokollports von σ haben. Wir realisieren dies, indem wir zunächst alle Protokollports von π in interne Ports umwandeln (aufgrund unserer Namenskonvention handelt es sich hierbei um eine einfache Umgebennung von Ports). Dann besteht das komponierte Protokoll σ^π aus den Maschinen aus σ und π, und die Maschinen aus σ rufen das Protokoll π auf, indem sie über die internen Ports kommunizieren, die ursprünglich Protokollports von π waren. Außerdem lassen wir nur Protokolle σ und π zu, die außer diesen Verbindungen keine weiteren untereinander aufbauen.

Die Situation ist nun also die folgende (vgl. Abbildung 2.4): Die Umgebung hat Verbindungen zu den Maschinen von σ über die Protokollports von σ. Das Protokoll σ hat über interne Ports Verbindungen zwischen seinen Maschinen und zu den ehemaligen Protokollports von π. Das Protokoll π hat Verbindungen zu σ nur über seine ehemaligen Protokollports (die zu internen geworden sind) und Verbindungen zwischen seinen Maschinen über seine (ursprünglichen) internen Ports. Außerdem können alle Maschinen in σ und π noch Verbindungen zum Angreifer oder Simulator über die Angreiferports aufbauen.

Um die einfache Komposition formal zu formulieren, müssen wir zunächst festlegen, was genau wir unter einer Portumbenennung verstehen. Intuitiv ist eine Portumbenennung nichts anderes, als der Name sagt, jedem Port wird ein neuer Name zugewiesen. Es handelt sich also um eine Abbildung von der Menge der Portnamen (Σ^+) auf die Menge der Portnamen. Was es formal in unserem Maschinenmodell heißt, Ports umzubenennen, klärt die folgende Definition:

Definition 2.17 (Portumbenennung)
Eine Portumbenennung P ist eine Abbildung von Σ^+ nach Σ^+.

Ist P eine Portumbenennung und M eine Maschine, so ist umbenannte Maschine M_P wie folgt definiert:

- $name_{M_P} = name_M$ (M_P und M haben den gleichen Namen).
- $out(M_P) = P(out(M))$ und $in(M_P) = P(in(M))$ (die Portnamen

(Fortsetzung nächste Seite)

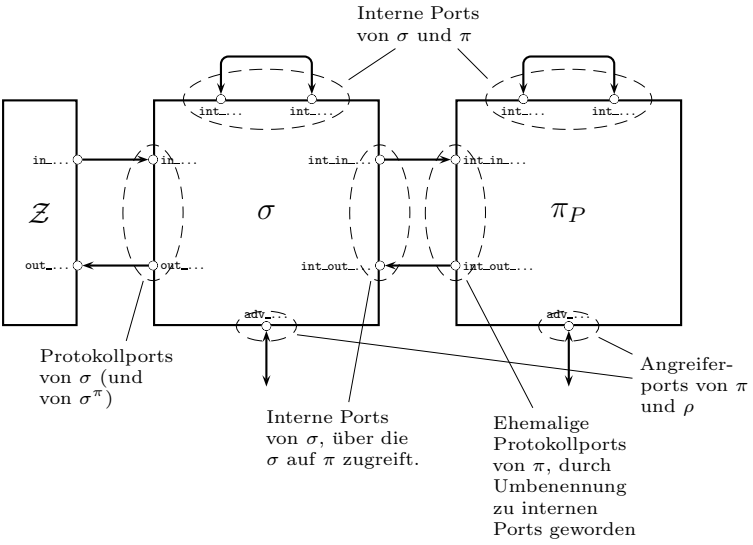


Abbildung 2.4.: Die einfache Komposition σ^π zweier Protokolle σ und π. Dargestellt sind das ursprüngliche Protokoll σ, das Protokoll π_P, welches aus π durch Umbenennung von Ports entsteht und die Umgebung, welche auf σ zugreift. Die Protokolle σ und π_P zusammen bilden das komponierte Protokoll σ^π. Die Pfeile in der Darstellung repräsentieren die Verbindungen zwischen den Ports, die durch die Umbenennungen in π_P erhalten werden.

(Fortsetzung)

von M_P sind die Bilder von der Portnamen von M).

- Die Zustandsübergangsfunktion von M_P resultiert, indem man das dritte Argument p (der eingehende Port) der Zustandsübergangsfunktion von M durch $P^{-1}(p)$ ersetzt, und die dritte Komponente p' (ausgehender Port) der Ausgabe der Zustandsübergangsfunktion von M durch $P(p')$ ersetzt (bzw. unverändert läßt, wenn $p' = \lambda$, also keine Nachricht gesandt wird). Formal:

$$\tilde{P}(s', \lambda, m') := (s', \lambda, m'),$$
$$\tilde{P}(s', p', m') := (s', P(p'), m') \text{ für } p' \in \Sigma^+,$$
$$M_P(k, s, p, m) := \tilde{P}(M(k, s, P^{-1}(p), m')).$$

(Für ein Wahrscheinlichkeitsmaß X ist das Bildmaß $\tilde{P}(X)$ definiert durch $\tilde{P}(X)(A) := X(\tilde{P}^{-1}(A))$ für meßbare Mengen A.)

Mit dieser Definition ist es nun nicht weiter schwierig, die einfache Komposition zu definieren (vgl. auch Abbildung 2.4):

Definition 2.18 (Einfache Komposition)
Es seien σ und π Protokolle. Weiter sei P die Portumbenennung, die jeden Protokollport p auf `int_`p abbildet (also aus Protokollports interne Ports macht), und auf allen anderen Ports die Identität ist.

Wir sagen, π sei *einbettbar* in σ, wenn folgendes gilt:

- $\sigma \cup \pi_P$ ist ein Netzwerk (d. h. es gibt keine Namenskonflikte).
- Die Portnamen von σ und π sind disjunkt (es werden keine Verbindungen außer über die ehemaligen Protokollports von π aufgebaut).
- Ist p ein Protokollport von π, so ist $P(p)$ kein Port von π (sonst würde π_P durch die Umbenennung neue interne Verbindungen erhalten).

Ist π in σ einbettbar, so ist die *einfache Komposition* σ^π als $\sigma \cup \pi_P$ definiert.

Gewappnet mit der einfachen Kompositionsoperation können wir nun untersuchen, inwiefern die Sicherheit eines Protokolls unter einfacher Komposition erhalten bleibt. Es gilt das folgende bemerkenswerte Kompositionstheorem:

Theorem 2.19 (Einfaches Kompositionstheorem)
Es bedeute $\pi \geq \rho$, daß π so sicher wie ρ ist bezüglich perfekter/statistischer/komplexitätstheoretischer allgemeiner/spezieller Sicherheit mit/ohne Auxiliary input bzgl. der Ausgabe/der Sicht der Umgebung.

Seien π, ρ und σ Protokolle, so daß π und ρ in σ einbettbar sind, und so daß $\pi \geq \rho$ gilt. Im Falle der komplexitätstheoretischen Sicherheit seien π, ρ und σ außerdem polynomiell-beschränkt.

Dann ist $\sigma^\pi \geq \sigma^\rho$.

Da die einfache Komposition in dieser Arbeit nicht von zentraler Bedeutung ist (sie wurde in [Lin03] bereits gut untersucht, unsere Analysen hingegen konzentrieren sich auf die nebenläufige und damit indirekt auf die allgemeine Komposition), verzichten wir an dieser Stelle darauf, einen genauen Beweis des einfachen Kompositionstheorem zu geben. Der geneigte Leser sei auf die Beweise in [Bac02b] und [Can01] verwiesen, die sich leicht auf unsere Modellierung übertragen lassen. Wir präsentieren jedoch eine Beweisskizze, die die wesentlichen Punkte des Beweises illustriert.

Beweisskizze: Wir zeigen zunächst den Fall der komplexitätstheoretischen speziellen Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung. Danach geben wir an, in welcher Weise der Beweis geändert werden muß, um die anderen Fälle zu zeigen.

Es seien also σ, π und ρ polynomiell-beschränkte Protokolle und wir nehmen an, π sei so sicher wie ρ bzgl. des obengenannten Sicherheitsbegriffs.

Um nun zu zeigen, daß σ^π auch so sicher wie σ^ρ ist, nehmen wir also einen zulässigen polynomiell-beschränkten Angreifer $\mathcal{A}$ und eine zulässige polynomiell-beschränkte Umgebung $\mathcal{Z}$ an, und müssen einen zulässigen polynomiell-beschränkten Simulator $\mathcal{S}$ konstruieren, so daß $\mathcal{Z}$ nicht zwischen σ^π mit $\mathcal{A}$ und σ^ρ mit $\mathcal{S}$ unterscheiden kann, d. h., daß

$$\mathrm{OUTPUT}_{\sigma^\pi,\mathcal{A},\mathcal{Z}} \qquad \text{und} \qquad \mathrm{OUTPUT}_{\sigma^\rho,\mathcal{S},\mathcal{Z}}$$

komplexitätstheoretisch ununterscheidbar sind (wir lassen dabei die Argumente k und z von OUTPUT der Kürze halber weg).

Wir betrachten nun das Netzwerk $\sigma^\pi \cup \{\mathcal{A}, \mathcal{Z}\}$ (siehe Abbildung 2.5a). Es fällt auf, daß, zumindest aus Sicht von π, es so aussieht, als wäre σ Teil der Umgebung (denn so war die Komposition σ^π gerade konstruiert). Wir können also statt des Netzwerks $\sigma^\pi \cup \{\mathcal{A}, \mathcal{Z}\}$ auch das modifizierte Netzwerk $\pi \cup \{\mathcal{A}, \mathcal{Z}_\sigma\}$ betrachten (siehe Abbildung 2.5b), bei dem $\mathcal{Z}_\sigma$ eine Umgebung ist, die sowohl $\mathcal{Z}$ als auch das Protokoll σ simuliert.[11] Wir haben dann

$$\mathrm{OUTPUT}_{\sigma^\pi,\mathcal{A},\mathcal{Z}} \;=\; \mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}_\sigma}. \tag{2.1}$$

[11] Genaugenommen müssen zusätzlich einige Ports von π, ρ und $\mathcal{Z}_\sigma$ umbenannt werden, da

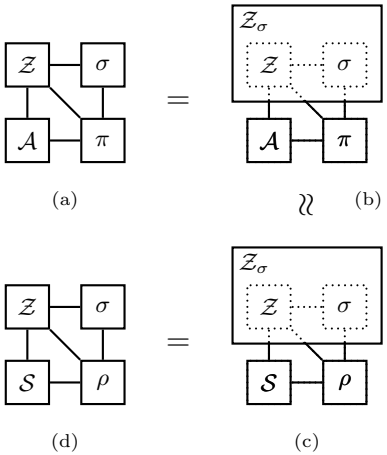


Abbildung 2.5.: Die Beweisschritte im einfachen Kompositionstheorem

Da π so sicher wie ρ ist, und da mit $\mathcal{Z}$ und σ auch $\mathcal{Z}_\sigma$ polynomiell-beschränkt ist, gibt es nun einen Simulator $\mathcal{S}$, so daß

$$\text{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}_\sigma} \qquad \text{und} \qquad \text{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}_\sigma} \tag{2.2}$$

komplexitätstheoretisch ununterscheidbar sind.

Das Netz $\rho, \mathcal{S}, \mathcal{Z}_\sigma$ (Abbildung 2.5c) wiederum können wir ersetzen durch $\sigma^\rho, \mathcal{S}, \mathcal{Z}$ (Abbildung 2.5d), indem wir $\mathcal{Z}_\sigma$ wieder „auspacken“. Wir haben dann also

$$\text{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}_\sigma} \;=\; \text{OUTPUT}_{\sigma^\rho,\mathcal{S},\mathcal{Z}}. \tag{2.3}$$

Kombinieren wir (2.1)–(2.3), so erhalten wir die Ununterscheidbarkeit der Zufallsvariablen $\text{OUTPUT}_{\sigma^\pi,\mathcal{A},\mathcal{Z}}$ und $\text{OUTPUT}_{\sigma^\rho,\mathcal{S},\mathcal{Z}}$. Also ist σ^π so sicher wie σ^ρ bzgl. der komplexitätstheoretischen speziellen Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung.

Die anderen Fälle (perfekte/statistische Sicherheit, Sicherheit ohne Auxiliary input sowie Sicherheit bzgl. der Sicht der Umgebung) werden analog bewiesen. Im Falle der Sicht der Umgebung nutzt man zusätzlich, daß die Sicht von $\mathcal{Z}$ aus der Sicht von $\mathcal{Z}_\sigma$ (effizient) berechnet werden kann. □

sonst $\mathcal{Z}_\sigma$ keine zulässige Umgebung ist (denn die Kommunikation zwischen der Umgebung und π darf nur über Umgebungsports laufen). Der Übersicht halber ignorieren wir diese und andere Portumbennenungen.

Das Kompositionstheorem allein genügt jedoch in den meisten Fällen nicht, wie das folgende Beispiel zeigt: Wir nehmen an, wir hätten ein Protokoll, welches unter Benutzung eines sicheren Münzwurfes eine bestimmte erstrebenswerte Funktionalität $\mathcal{F}_{\text{good}}$ implementiert. In anderen Worten, wenn wir die ideale Funktionalität des sicheren Münzwurfes als $\mathcal{F}_{\text{CT}}$ bezeichnen, so kennen wir ein Protokoll σ, so daß $\sigma^{\mathcal{F}_{\text{CT}}}$ so sicher wie die Funktionalität $\mathcal{F}_{\text{good}}$ ist. Weiterhin möge ein Münzwurfprotokoll π gegeben sein, d. h. π ist so sicher wie $\mathcal{F}_{\text{CT}}$. Dann sagt uns das Kompositionstheorem, daß σ^{π} so sicher wie $\sigma^{\mathcal{F}_{\text{CT}}}$ ist, welches wiederum so sicher wie $\mathcal{F}_{\text{good}}$ ist. Doch uns interessiert, ob σ^{π} (das endgültige Protokoll) $\mathcal{F}_{\text{good}}$ (die angestrebte Spezifikation) implementiert. Darüber macht das Kompositionstheorem allein keine Aussage, wir benötigen zusätzlich, daß die durch den Sicherheitsbegriff gegebene Relation „so sicher wie“ transitiv ist. Dies garantiert das folgende Lemma:

Lemma 2.20 (Transitivität und Reflexivität der Sicherheit)
Es seien $C_{\mathcal{Z}}$, $C_{\mathcal{A}}$ und $C_{\mathcal{S}}$ Mengen von Maschinen.

Es bedeute dann $\pi \geq \rho$, daß das Protokoll π so sicher wie das Protokoll ρ ist (bzgl. allgemeiner/spezieller komplexitätstheoretischer/statistischer/perfekter Sicherheit mit/ohne Auxiliary input bzgl. der Ausgabe/der Sicht der Umgebung und mit Umgebungen, Angreifern und Simulatoren in $C_{\mathcal{Z}}$, $C_{\mathcal{A}}$ bzw. $C_{\mathcal{S}}$).

Ist $C_{\mathcal{A}} \subseteq C_{\mathcal{S}}$, so ist $\geq$ reflexiv.

Ist $C_{\mathcal{A}} \supseteq C_{\mathcal{S}}$, so ist $\geq$ transitiv.

Beweisskizze: Wenn $\pi = \rho$ und $\mathcal{S} = \mathcal{A}$ gilt, sind trivialerweise die Sicht/Ausgabe von $\mathcal{Z}$ in einem Protokollauf von $\pi, \mathcal{A}, \mathcal{Z}$ und $\rho, \mathcal{S}, \mathcal{Z}$ gleich. Somit erhalten wir die Reflexivität, indem wir $\mathcal{S} := \mathcal{A}$ setzen, was wegen $C_{\mathcal{A}} \subseteq C_{\mathcal{S}}$ immer erlaubt ist.

Um die Transitivität zu zeigen, nehmen wir an, π sei so sicher wie ρ und ρ so sicher wie σ. Dann existiert zu jedem Angreifer $\mathcal{A} \in C_{\mathcal{A}}$ ein Simulator $\mathcal{S} \in C_{\mathcal{S}} \subseteq C_{\mathcal{A}}$, so daß $\text{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}$ und $\text{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}$ ununterscheidbar sind. Weiter existiert für jeden Angreifer $\mathcal{A}' \in C_{\mathcal{A}}$ ein Simulator $\mathcal{S}' \in C_{\mathcal{S}}$, so daß $\text{OUTPUT}_{\rho,\mathcal{A}',\mathcal{Z}}$ und $\text{OUTPUT}_{\sigma,\mathcal{S}',\mathcal{Z}}$ ununterscheidbar sind. Setzen wir $\mathcal{A}' := \mathcal{S}$, so erhalten wir, daß $\text{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}$ und $\text{OUTPUT}_{\sigma,\mathcal{S}',\mathcal{Z}}$ ununterscheidbar sind. Damit ist π so sicher wie σ. Völlig identisch sieht der Beweis bzgl. der Sicht der Umgebung aus, mit VIEW statt OUTPUT. □

Man beachte, daß in den meisten Fällen $C_{\mathcal{A}} = C_{\mathcal{S}}$ ist, so daß Reflexivität und Transitivität gegeben sind. Allerdings ist auch der Fall interessant, in dem $C_{\mathcal{S}}$ deutlich größer ist als $C_{\mathcal{A}}$, da dann die in [CF01, CKL03] gezeigten Unmöglichkeitsresultate nicht mehr zutreffen [BS05]. In diesem Fall gilt die Transitivität

nicht mehr uneingeschränkt, und bei der Komposition muß man entsprechend Vorsicht walten lassen.

Anders als die Transitivität wird die Reflexivität so gut wie nie in Beweisen verwendet. Doch scheint ein Sicherheitsbegriff, der nicht reflexiv ist, nicht sehr sinnvoll, so daß es dennoch wichtig ist, einen Sicherheitsbegriff auf Reflexivität zu überprüfen.

2.3.2. Notwendigkeit der speziellen Sicherheit

Wir wissen nun, daß spezielle und allgemeine Sicherheit einfache Komponierbarkeit garantieren. Jedoch wurde in [CF01, CKL03] gezeigt, daß diese Begriffe sehr streng sind und viele kryptographische Aufgabenstellungen bezüglich dieser einfach unlösbar sind. Es stellt sich also die Frage, ob nicht auch weniger strenge Begriffe denkbar sind, die ebenfalls einfache Komposition erlauben. Unglücklicherweise ist dem nicht so, wie [Lin03] zeigte. Um das Resultat von [Lin03] zu verstehen, müssen wir zunächst klären, was wir unter einem einfach komponierbaren Protokoll verstehen. Man wird im ersten Moment geneigt sein, ein Protokoll (genauer ein Paar von Protokollen) einfach komponierbar zu nennen, wenn es das Kompositionstheorem 2.19 erfüllt. Doch was bedeutet dies? Verlangen wir, daß ein Protokoll π, in ein umgebendes Protokoll σ eingesetzt, zu einem sicheren Protokoll π^σ im Sinne unserer strikten Sicherheitsdefinition wird? In diesem Fall ist es nicht überraschend, daß schon das eingebettete Protokoll unsere strikte Sicherheitsdefinition erfüllen muß. Eine andere Möglichkeit wäre, nach der schwächsten Sicherheitsdefinition zu fragen, die das Kompositionstheorem erfüllt. Doch auch dies führt nicht zu Erfolg, denn der entartete Sicherheitsbegriff, der jedes Protokoll als so sicher wie jedes andere bezeichnet, erfüllt das Kompositionstheorem (sowohl die Prämisse als auch die Folgerung des Theorems sind immer trivial erfüllt).

Man muß also zunächst einen Sicherheitsbegriff $\mathrm{Sec}_{\min}$ fixieren, der die Mindestanforderungen angibt, die wir neben der Komponierbarkeit noch von einem Protokoll fordern. Dann sagen wir, ein Protokoll π sei so sicher wie ρ bezüglich einfacher Komponierbarkeit, wenn für jedes Protokoll σ gilt, daß σ^π so sicher wie σ^ρ bezüglich $\mathrm{Sec}_{\min}$ ist. Dies ist der von [Lin03] verfolgte Ansatz. Dort wurde für den minimalen Sicherheitsbegriff $\mathrm{Sec}_{\min}$ eine Variante der simulierbaren Sicherheit *ohne Umgebung* gewählt (vgl. Abschnitt 2.1.2). Dann wurde gezeigt, daß die einfache Komponierbarkeit bereits die spezielle Sicherheit impliziert. Die untersuchte Variante der simulierbaren Sicherheit ohne Umgebung war die komplexitätstheoretische Sicherheit mit Auxiliary input (den dann natürlich der Angreifer erhält, da die Umgebung nicht existiert). Allerdings kann man leicht einsehen, daß auch für perfekte oder statistische, sowie für Sicherheit ohne Auxiliary input Vergleichbares gilt. Wir präsentieren deshalb das Resultat von [Lin03] in verallgemeinerter Form, um auch diese Varianten abzudecken.

(Insbesondere der Fall ohne Auxiliary input scheint sehr wichtig, denn gäbe es einen schwächeren Sicherheitsbegriff, der nach Komposition nur Sicherheit *ohne* Auxiliary input garantiert, so spräche wohl nicht viel dagegen, diesen zu verwenden.)

Um die einfache Komponierbarkeit zu definieren, müssen wir zunächst den „minimalen Sicherheitsbegriff“ festlegen. Es ist hierbei wichtig, daß dieser Sicherheitsbegriff im gleichen Netzwerk- und Maschinenmodell formuliert wird wie die Sicherheitsbegriffe aus Abschnitt 2.2.4. Andernfalls ist es nicht möglich, von dem gleichen Protokoll im einen und im anderen Modell zu sprechen. Wie in Abschnitt 2.1.2 skizziert, unterscheidet sich eine simulierbare Sicherheitsdefinition ohne Umgebung von denen aus Abschnitt 2.2.4 im wesentlichen dadurch, daß statt einer Umgebung, die mit dem Protokoll interagiert, feste (vor Protokollbeginn festgelegte) Eingaben für die Parteien vorliegen. Damit ein Protokoll π als so sicher wie ein anderes Protokoll ρ gilt, wird dann nicht verlangt, daß die Ausgabe der Umgebung ununterscheidbar ist, sondern daß die Ausgabe des realen Protokolls π und des Angreifers ununterscheidbar von der Ausgabe des idealen Protokolls und des Simulators ist.

Die formale Definition lautet also wie folgt:

Definition 2.21 (Sicherheit ohne Umgebung)
Für ein Protokoll π sei $parties(\pi)$ die (lexikographisch sortierte) Liste der Identitäten aller Maschinen in π, welche einen ausgehenden Port `output` haben (d. h., die Maschinen, die Ausgabe generieren können).

Es seien π und ρ Protokolle mit $parties(\pi) = parties(\rho) = (id_1, \ldots, id_n)$.

Dann heißt π so sicher wie ρ bezüglich *perfekter/statistischer/komplexitätstheoretischer Sicherheit ohne Umgebung mit Auxiliary input*, wenn für jeden zulässigen (und – im Falle komplexitätstheoretischer Sicherheit – polynomiell-beschränkten) Angreifer $\mathcal{A}$ ein zulässiger (und – im Falle komplexitätstheoretischer Sicherheit – polynomiell-beschränkter) Simulator $\mathcal{S}$ existiert, so daß die folgenden zwei Familien von Zufallsvariablen im Falle perfekter Sicherheit gleich, im Falle statistischer Sicherheit statistisch ununterscheidbar und im Falle komplexitätstheoretischer Sicherheit komplexitätstheoretisch ununterscheidbar sind:

$$\left\{\mathrm{OUTPUT}_{\pi\cup\{\mathcal{A}\},k}(X)\right\}_{k\in\mathbb{N},(x_0,\ldots,x_n)\in(\Sigma^*)^{n+1}}$$

und

$$\left\{\mathrm{OUTPUT}_{\rho\cup\{\mathcal{S}\},k}(X)\right\}_{k\in\mathbb{N},(x_0,\ldots,x_n)\in(\Sigma^*)^{n+1}},$$

(Fortsetzung nächste Seite)

(Fortsetzung)

mit

$$X := \texttt{adv} \leftarrow x_0, id_1 \leftarrow x_1, \ldots, id_n \leftarrow x_n : \texttt{adv}, id_1, \ldots, id_n.$$

(Also die Ausgabe des Angreifers/Simulators und aller Parteien, wenn der Angreifer/Simulator die Eingabe x_0 und die Parteien die Eingaben $x_1, \ldots, x_n$ erhalten.)

Sicherheit ohne Umgebung ohne Auxiliary input definieren wir genauso, nur daß $(x_0, \ldots, x_n)$ aus $\{\lambda\}^{n+1}$ statt aus $(\Sigma^*)^{n+1}$ gewählt wird.

Wir können nun π als so sicher wie ρ bezüglich einfacher Komponierbarkeit definieren, wenn für jedes Protokoll σ (in das sowohl π als auch ρ einbettbar ist) das komponierte Protokoll σ^π so sicher wie σ^ρ bezüglich Sicherheit ohne Umgebung ist. Dabei ist es wichtig, daß wir im Falle der komplexitätstheoretischen Sicherheit nur über polynomiell-beschränkte umgebende Protokolle σ quantifizieren.

Definition 2.22 (Einfache Komponierbarkeit)
Es seien π und ρ Protokolle mit $\mathit{parties}(\pi) = \mathit{parties}(\rho) = \varnothing$, d. h. keine Maschine in π oder ρ hat direkt Ausgabe (die Maschinen dürfen aber über Protokollports Nachrichten an die Umgebung schicken).[12] Darüber hinaus habe weder π noch ρ einen internen Port, der mit `int_in_` oder `int_out_` beginnt.[13] (Wir nennen Protokolle mit diesen Eigenschaften im folgenden *prinzipiell einbettbar*.)

Dann heißt π so sicher wie ρ bezüglich perfekter/statistischer/komplexitätstheoretischer einfacher Komponierbarkeit mit/ohne Auxiliary input, wenn für jedes (im komplexitätstheoretischen Falle polynomiell-beschränkte) Protokoll σ gilt, daß σ^π so sicher wie σ^ρ ist bezüglich perfekter/statistischer/komplexitätstheoretischer Sicherheit ohne Umgebung mit/ohne Auxiliary input.

Mit dieser Definition können wir uns nun die Frage stellen, welche Protokolle einfach komponierbar sind. Verallgemeinern wir das Ergebnis von [Lin03] und übertragen es auf unsere Modellierung, so ergibt sich das Theorem:

[12] Maschinen mit direkter Ausgabe würden die Definition unnötig komplizieren und keine wesentliche zusätzliche Allgemeinheit bringen, da die Sicherheitsdefinitionen mit Umgebung die Ausgabe der Protokollmaschinen ignorieren.

[13] Wir verlangen dies, um zu garantieren, daß es immer Protokolle σ gibt, in die π und ρ einbettbar sind. Es handelt sich um keine wesentliche Einschränkung, da man solche Ports einfach umbenennen kann.

Theorem 2.23 (Äquivalenz von einfacher Komponierbarkeit und spezieller Sicherheit)
Es seien π und ρ prinzipiell einbettbare Protokolle. Im Falle komplexitätstheoretischer Sicherheit seien außerdem π und ρ polynomiell-beschränkt.

Das Protokoll π ist genau dann so sicher wie ρ bezüglich *spezieller* perfekter/statistischer/komplexitätstheoretischer Sicherheit mit/ohne Auxiliary input bzgl. der Ausgabe der Umgebung, wenn π so sicher wie ρ ist bezüglich perfekter/statistischer/komplexitätstheoretischer *einfacher* Komponierbarkeit mit/ohne Auxiliary input.

Wie beim einfachen Kompositionstheorem 2.19 skizzieren wir den Beweis nur.

Beweisskizze: Zunächst wollen wir zeigen, daß einfache Komponierbarkeit schon spezielle Sicherheit impliziert. Hierzu beschränken wir uns auf den komplexitätstheoretischen Fall mit Auxiliary input. Die anderen Fälle werden völlig analog gezeigt.

Es seien also π und ρ Protokolle wie im Theorem, und wir nehmen an, π sei so sicher wie ρ bezüglich einfacher Komponierbarkeit.

Wir wollen nun zeigen, daß π auch so sicher wie ρ bezüglich spezieller Sicherheit ist. Dazu seien ein zulässiger polynomiell-beschränkter Angreifer $\mathcal{A}$ und eine zulässige polynomiell-beschränkte Umgebung $\mathcal{Z}$ gegeben, und wir müssen nun einen zulässigen polynomiell-beschränkten Simulator $\mathcal{S}$ konstruieren, so daß die Familien

$$\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) \qquad \text{und} \qquad \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z) \tag{2.4}$$

komplexitätstheoretisch ununterscheidbar sind. (Wir lassen hier die geschweiften Klammern zur Kennzeichnung der Familien von Zufallsvariablen der Kürze halber weg.) Vgl. Abbildungen 2.6a und 2.6d.

Dazu konstruieren wir zunächst ein umgebendes Protokoll σ für π und ρ. Dieses Protokoll enthält als einzige Maschine die (ehemalige) Umgebung $\mathcal{Z}$.[14] So wie vorher die Umgebung das Protokoll π oder ρ als aufgerufen hat, so ruft nun σ das Protokoll π bzw. ρ als Unterprotokoll in der Komposition σ^π bzw. σ^ρ auf. Auch die Kommunikation zwischen Umgebung und Angreifer wurde durch Kommunikation zwischen σ und Angreifer ersetzt. Die neue Situation zeigen Abbildungen 2.6b und 2.6c.

[14] Genaugenommen muß die Umgebung umbenannt werden, da ein Protokoll keine Maschine mit dem Namen der Umgebung haben darf. Darüber hinaus darf eine Protokollmaschine nicht die gleichen Porttypen wie die Umgebung haben, so daß Umgebungsports in Angreiferports, und Protokollports in interne Ports umbenannt werden müssen. Diese Details übergehen wir im folgenden und tun so, als enthielte das Protokoll σ die unveränderte Maschine $\mathcal{Z}$.

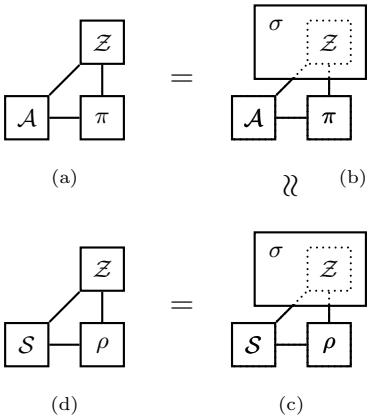


Abbildung 2.6.: Die Beweisschritte für die Folgerung der speziellen Sicherheit aus der einfachen Komponierbarkeit

Da das Netzwerk $\sigma^\pi \cup \mathcal{A}$ aus Abbildung 2.6b lediglich eine Regruppierung des Netzwerks $\pi \cup \{\mathcal{A}, \mathcal{Z}\}$ aus Abbildung 2.6a ist, ergibt sich direkt, daß die Ausgabe der Umgebung im letzteren die gleiche Verteilung wie die Ausgabe der „Umgebung“ in σ hat, also

$$\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) = \mathrm{OUTPUT}_{\sigma^\pi \cup \{\mathcal{A}\},k}(\mathtt{env} \leftarrow z : \mathtt{env}). \tag{2.5}$$

Ganz analog erhalten wir für jeden beliebigen Simulator $\mathcal{S}$ (vgl. Abbildungen 2.6c und 2.6d)

$$\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z) = \mathrm{OUTPUT}_{\sigma^\rho \cup \{\mathcal{S}\},k}(\mathtt{env} \leftarrow z : \mathtt{env}). \tag{2.6}$$

Da σ ein polynomiell-beschränktes Protokoll ist, garantiert nun die einfache Komponierbarkeit von π und ρ, daß σ^π so sicher wie σ^ρ ist bezüglich komplexitätstheoretischer Sicherheit ohne Umgebung mit Auxiliary input. Das heißt nach Definition, daß ein Simulator $\mathcal{S}$ existiert, so daß die Ausgabe von $\mathcal{A}$ und aller Parteien in σ^π (d. h. die Ausgabe von $\mathcal{Z}$) ununterscheidbar von der Ausgabe von $\mathcal{S}$ und aller Parteien in σ^ρ (ebenfalls nur $\mathcal{Z}$) ist, d. h. die folgenden Zufallsvariablen sind komplexitätstheoretisch ununterscheidbar:

$$\mathrm{OUTPUT}_{\sigma^\pi \cup \{\mathcal{A}\},k}(\mathtt{adv} \leftarrow x_0, \mathtt{env} \leftarrow x_1 : \mathtt{adv}, \mathtt{env}) \quad \text{und}$$
$$\mathrm{OUTPUT}_{\sigma^\rho \cup \{\mathcal{S}\},k}(\mathtt{adv} \leftarrow x_0, \mathtt{env} \leftarrow x_1 : \mathtt{adv}, \mathtt{env}).$$

Wenn wir die Eingabe des Angreifers auf $x_0 := \lambda$ festlegen, und nur die Ausgabe der Umgebung (statt der Ausgabe der Umgebung und des Angreifers)

berücksichtigen, bleibt die Ununterscheidbarkeit erhalten, es sind also komplexitätstheoretisch die folgenden Familien von Zufallsvariablen ununterscheidbar:

$$\mathrm{OUTPUT}_{\sigma^\pi \cup \{\mathcal{A}\},k}(\mathtt{env} \leftarrow z : \mathtt{env}) \quad \text{und}$$
$$\mathrm{OUTPUT}_{\sigma^\rho \cup \{\mathcal{S}\},k}(\mathtt{env} \leftarrow z : \mathtt{env}).$$

Mit Gleichungen (2.5,2.6) ergibt sich daraus (2.4), also ist π so sicher wie ρ bezüglich spezieller Sicherheit.

Man beachte, daß man diese Argumentation nicht auf den Fall der allgemeinen Sicherheit übertragen kann, da dort die Umgebung vom Simulator abhängen darf, und deshalb bei obiger Konstruktion das Protokoll σ vom Simulator abhängen müßte.

Es bleibt einzusehen, daß spezielle Sicherheit auch einfache Komponierbarkeit impliziert. Das Kompositionstheorem 2.19 sagt uns, daß – ist π so sicher wie ρ bezüglich spezieller Sicherheit – auch σ^π so sicher wie σ^ρ bezüglich spezieller Sicherheit ist (wobei σ ein beliebiges polynomiell-beschränktes Protokoll ist). Weiterhin kann man einsehen, daß die spezielle Sicherheit mindestens so streng wie die Sicherheit ohne Umgebung ist, denn die Umgebung kann die Eingaben der Parteien und des Angreifers aus ihrem Auxiliary input generieren (und im Falle ohne Auxiliary input gibt es keine Eingabe für Parteien und Angreifer), und die Ausgabe aller Parteien und des Angreifers kann wiederum von der Umgebung ausgegeben werden.[15] Somit ist σ^π so sicher wie σ^ρ bezüglich Sicherheit ohne Umgebung, und damit, da σ beliebig war, auch π so sicher wie ρ bezüglich einfacher Komponierbarkeit. □

2.3.3. Nebenläufige Komposition

Auf den ersten Blick mag es scheinen, als decke die einfache Komponierbarkeit bereits alle denkbaren Kompositionsszenarien ab. Selbst wenn wir mehrere idealisierte Protokollbausteine $\mathcal{F}_1, \ldots, \mathcal{F}_n$ in einem umgebenden Protokoll σ durch ihre sicheren Implementierungen $\pi_1, \ldots, \pi_n$ ersetzen wollen, so erhalten wir durch sukzessive Anwendung des einfachen Kompositionstheorems 2.19 und der Transitivität (Lemma 2.20)

$$\sigma^{\pi_1,\ldots,\pi_n} \geq \sigma^{\pi_1,\ldots,\pi_{n-1},\mathcal{F}_n} \geq \cdots \geq \sigma^{\pi_1,\mathcal{F}_2,\ldots,\mathcal{F}_n} \geq \sigma^{\mathcal{F}_1,\ldots,\mathcal{F}_n} \tag{2.7}$$

Hier meint $\alpha \geq \beta$, daß α so sicher wie β ist, und $\sigma^{\pi_1,\ldots,\pi_n}$ bedeutet, daß in σ sukzessive die Protokolle π_1 bis π_n eingebettet wurden.

[15] Im Detail muß man hier vorsichtig sein, sollte das Protokoll nicht terminieren. Falls dann einige Parteien nie Ausgabe liefern, würde die Umgebung ewig auf diese Ausgabe warten, und damit selbst die Ausgaben der terminierenden Parteien nicht ausgeben. Dies läßt sich aber lösen, indem die Umgebung zufällig wählt, von welchen Parteien sie die Ausgabe abwartet.

Es scheint also, als wäre die sichere Komposition beliebig vieler Protokolle durch das einfache Kompositionstheorem garantiert. Man betrachte aber das folgende (durchaus realistische) Beispiel: Wir wollen eine Funktionalität $\mathcal{G}$ zur sicheren Nachrichtenübertragung implementieren. Wir nehmen an, diese Funktionalität lasse bei Sicherheitsparameter k die Übertragung von k Nachrichten der Länge k zu. Dazu verwenden wir ein Protokoll σ, welches für jede zu übertragende Nachricht einen Schlüsselaustausch durchführt. Es bezeichne $\mathcal{F}$ die Funktionalität für einen einzelnen Schlüsselaustausch. Dann ist die Situation also die folgende: $\sigma^{\mathcal{F}_1,\ldots,\mathcal{F}_k}$ ist so sicher wie $\mathcal{G}$, wobei $\sigma^{\mathcal{F}_1,\ldots,\mathcal{F}_k}$ das Protokoll σ bezeichnet, welches k Instanzen der Schlüsselaustauschfunktionalität $\mathcal{F}_i := \mathcal{F}$ aufruft. Man beachte, daß die Anzahl der Funktionalitäten mit dem Sicherheitsparameter k wächst.

Man ist nun versucht, analog zu (2.7) folgendes zu schreiben:

$$\sigma^{\pi_1,\ldots,\pi_k} \geq \sigma^{\pi_1,\ldots,\pi_{k-1},\mathcal{F}_k} \geq \cdots \geq \sigma^{\pi_1,\mathcal{F}_2,\ldots,\mathcal{F}_k} \geq \sigma^{\mathcal{F}_1,\ldots,\mathcal{F}_k} \tag{2.8}$$

Doch hier täuscht das Auslassungszeichen $(\cdots)$ über ein subtiles Problem hinweg. In (2.7) ist es prinzipiell möglich, für festes n die Ungleichung ohne Auslassungszeichen auszuschreiben, und die Notation mit Auslassungszeichen bedeutet, daß für jedes n die für dieses n ausgeschriebene Formel wahr ist. Dies wird dann per Induktion über n gezeigt.

In (2.8) hingegen ist es nicht möglich, die Formel ohne Auslassungszeichen zu schreiben, da wir nicht über einen *festen* Sicherheitsparameter k sprechen können (denn das Symbol $\geq$ macht implizit eine Aussage über durch k parametrisierte Familien, und Sicherheit für einen *einzelnen* Sicherheitsparameter ist nicht definiert). Deshalb hat (2.8) keine Bedeutung, und kann auch nicht induktiv aus dem einfachen Kompositionstheorem gefolgert werden. Tatsächlich werden wir in Kapitel 6 zeigen, daß (2.8) im allgemeinen nicht gilt.

Um also Beispiele wie das obige behandeln zu können, brauchen wir zweierlei: Zum einen ist eine Definition von $\sigma^{\mathcal{F}_1,\ldots,\mathcal{F}_k}$ nötig, die ohne Auslassungszeichen auskommt (oder die zumindest prinzipiell ohne Auslassungszeichen formulierbar ist), und zum anderen ein Theorem, welches eine allgemeine Komposition wie im obigen Beispiel zuläßt. Bevor wir diese Problemstellung in Abschnitt 2.3.4 in ihrer Allgemeinheit angehen, behandeln wir zunächst den Spezialfall der *nebenläufigen Komposition*, auf den wir dann den allgemeinen Fall zurückführen werden.

Unter nebenläufiger Komposition verstehen wir, daß eine (vom Sicherheitsparameter k abhängige) Anzahl von Instanzen eines Protokolls oder einer Funktionalität gleichzeitig ausgeführt werden. In der in diesem Abschnitt bisher verwendeten Notation würde man das in etwa als $\mathcal{F}_1, \ldots, \mathcal{F}_{f(k)}$ schreiben, wir werden im folgenden aber die genauere Notation $f \cdot \mathcal{F}$ („f-mal $\mathcal{F}$“, also $f(k)$ Instanzen von $\mathcal{F}$) verwenden. Wir wollen im folgenden einsehen, unter welchen Bedingungen $f \cdot \pi$ so sicher wie $f \cdot \mathcal{F}$ ist.

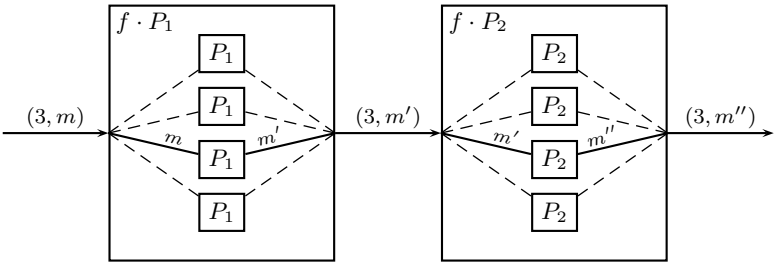


Abbildung 2.7.: Beispielhafte Darstellung der nebenläufigen Komposition $f \cdot \pi$ eines Protokolles π bestehend aus zwei Maschinen P_1, P_2. In diesem Fall ist $f(k) = 4$, so daß vier Submaschinen simuliert werden. Eine Nachricht $(3, m)$ erreicht $f \cdot P_1$ und wird an die dritte Submaschine P_1 weitergeleitet. Diese antwortet mit m', was an die entsprechende (also dritte) Submaschine von $f \cdot P_2$ weitergeleitet wird. Zwischen den Maschinen wird die Nachricht als $(3, m')$ dargestellt. Die Submaschine von $f \cdot P_2$ antwortet mit einer Nachricht m''.

Zunächst wenden wir uns dem Problem der formalen Definition von $f \cdot \pi$ zu. Eine Möglichkeit ist, analog zur einfachen Komposition in Definition 2.18, $f \cdot \pi$ als ein Netzwerk aufzufassen, das $f(k)$ Kopien von jeder Maschine in π enthält [BPW04a]. Die Schwierigkeit hierbei ist, daß die Struktur des Netzwerkes (also die Menge der enthaltenen Maschinen und die Menge der Ports) vom Sicherheitsparameter abhängt. Dies macht es zum einen nötig, den Netzwerkbegriff dahingehend zu erweitern, und darüber hinaus das Maschinenmodell für Umgebung, Angreifer und Simulator zu ändern, da diese nun auch eine vom Sicherheitsparameter abhängige Anzahl von Ports haben müssen. Um diese Schwierigkeiten zu umgehen (die die Präsentation stark verkomplizieren würden), wählen wir einen anderen Zugang. Jede Maschine $f \cdot P_i$ im Protokoll $f \cdot \pi$ simuliert $f(k)$ Instanzen der zugehörigen Maschine P_i aus π. Sendet dann die i-te simulierte Kopie von $f(k)$ eine Nachricht m über einen ausgehenden Port p, so wird von $f \cdot P_i$ die Nachricht (i, m) über p gesandt, so daß die Nachricht der entsprechenden Submaschine zugeordnet werden kann (vgl. Abbildung 2.7). Ebenso wird eine eingehende Nachricht der Form (i, m) als m an die i-te Kopie von P_i weitergeleitet. Die Nummer i der Submaschine bezeichnen wir auch als die *Session-ID*. Der Effekt dieser Konstruktion ist, daß alle Maschinen mit der gleichen Session-ID miteinander kommunizieren, und – sofern der Angreifer oder die Umgebung nicht explizit Daten von einer Protokollinstanz in die andere übertragen – von den anderen Protokollen nichts mitbekommen. Diese Konstruktion $f \cdot \pi$ verhält sich also genauso wie ein Netzwerk, das aus $f(k)$ unverbundenen Kopien des Protokolls π besteht.

Wir geben zwei Definitionen für die nebenläufige Komposition. Eine informel-

le, die die wesentlichen Details klar darstellt, und eine formale, die die genaue Konstruktion im Maschinenmodell aus Definition 2.1 angibt. Wir empfehlen dem Leser, zunächst die informelle Definition 2.24 zu verwenden und nur im Zweifelsfalle auf die genaue, aber schwer lesbare Definition zurückzugreifen.

Definition 2.24 (Nebenläufige Komposition – skizziert)
Es sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion und M eine Maschine. Dann ist die *f-fache nebenläufige Komposition* $f \cdot M$ eine Maschine mit den gleichen Ports wie M, die das folgende Verhalten hat:

- Es sei k der Sicherheitsparameter. Die Maschine $f \cdot M$ simuliert $f(k)$ Kopien von M, deren Zustände wir mit $s_1, \ldots, s_{f(k)}$ bezeichnen. Anfangs sind alle $s_i = \lambda$.
- Wenn M eine Nachricht der Form (sid, m) auf dem eingehenden Port p erhält, wobei die Session-ID sid in $\{1, \ldots, f(k)\}$ liegt, dann wird die sid-te Kopie von M mit der Nachricht m auf dem eingehenden Port p aufgerufen (was zu einer Veränderung des Zustandes s_{sid} führt).
- Nachrichten, die nicht von dieser Form sind, werden ignoriert.
- Wurde die sid-te Kopie von M aufgerufen, und hat sie dabei eine Nachricht m über den ausgehenden Port p gesandt, so schickt $f \cdot M$ die Nachricht (sid, m) über den ausgehenden Port p.

Ist π ein Protokoll bestehend aus den Maschinen $P_1, \ldots, P_n$, so ist die *f-fache nebenläufige Komposition* $f \cdot \pi$ das Protokoll bestehend aus den Maschinen $f \cdot P_1, \ldots, f \cdot P_n$.

Definition 2.25 (Nebenläufige Komposition – formal)
Es sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion und M eine Maschine. Dann ist die *f-fache nebenläufige Komposition* $f \cdot M$ die Maschine mit dem Namen $name_{f \cdot M} := name_M$, den eingehenden Ports $in(f \cdot M) := in(M)$, den ausgehenden Ports $out(f \cdot M) := out(M)$, und der Zustandsübergangsfunktion M', die im folgenden beschrieben wird.

Es sei $\iota : \mathbb{N} \to \Sigma^+$ eine feste, effizient berechenbare und effizient invertierbare Funktion, so daß für alle $i \neq j$ das Wort $\iota(i)$ kein Präfix von $\iota(j)$ ist. Es bezeichne (i, m) die Konkatenation von $\iota(i)$ und m.[16]

Weiterhin sei $tupel : (\Sigma^*)^* \to \Sigma^*$ effizient berechenbar und effizient in-

(Fortsetzung nächste Seite)

(Fortsetzung)

vertierbar.

Für $k \in \mathbb{N}$, $sid \in \{1, \ldots, f(k)\}$, $s_1, \ldots, s_{f(k)} \in \Sigma^*$, $p \in in(M)$, $m \in \Sigma^*$ sei

$$M'(k, tupel(s_1, \ldots, s_{f(k)}), p, (sid, m)) := \gamma^{sid}_{s_1,\ldots,s_{f(k)}}\big(M(k, s_{sid}, p, m)\big)$$

wobei $\gamma^{sid}_{s_1,\ldots,s_{f(k)}}$ wie folgt definiert ist:

$$\gamma^{sid}_{s_1,\ldots,s_{f(k)}}(s', p', m') := \begin{cases} \big(tupel(s'_1, \ldots, s'_{f(k)}), p', (sid, m')\big), & \text{falls } p' \neq \lambda, \\ \big(tupel(s'_1, \ldots, s'_{f(k)}), \lambda, \lambda\big), & \text{falls } p' = \lambda. \end{cases}$$

Hierbei sei $s'_{sid} := s'$ und $s'_j := s_j$ für $j \neq sid$.

Ist s nicht von der Form $tupel(s_1, \ldots, s_{f(k)})$ oder m nicht von der Form $(sid, \tilde{m})$, dann sei $M'(k, s, p, m)$ die Verteilung, die dem Tupel (s, λ, λ) die Wahrscheinlichkeit 1 zuordnet (nicht wohlgeformte Nachrichten und unerreichbare interne Zustände werden ignoriert).

Bei genauer Betrachtung dieser Definition fallen zwei Dinge auf (die beim ersten Lesen getrost übersprungen werden können):

- Die Session-IDs der Subprotokolle werden aus dem Bereich $\{1, \ldots, f(k)\}$ gewählt. Ein anderer, allgemeinerer Ansatz wäre (der in der Modellierung der allgemeinen Komposition in [Can01] verfolgt wird) beliebige Session-IDs zuzulassen, aber zu verlangen, daß nur polynomiell viele verschiedene Session-IDs verwendet werden. Dies gibt zusätzliche Möglichkeiten beim Entwurf von Protokollen, da ein umgebendes Protokoll zwei Protokollinstanzen mit zufälligen Session-IDs aufrufen könnte, so daß die beiden Protokolle nicht kollidieren, also nicht die gleiche Session-ID haben. In unserem Falle aber ist die Menge der möglichen Session-IDs nur von polynomieller Größe, eine Kollision hat also eine nicht vernachlässigbare Wahrscheinlichkeit. Deshalb muß das umgebende Protokoll auf andere Weise garantieren, daß verschiedene Session-IDs gewählt werden.

  Die allgemeinere Definition hat jedoch den großen Nachteil, daß das Resultat der nebenläufigen Komposition dann nicht polynomiell-beschränkt

[16] Es sind hier natürlich auch andere Konstruktionen des Paares (i, m) möglich. Doch nicht jede effizient berechenbare und effizient invertierbare Funktion t von $\mathbb{N} \times \Sigma^*$ nach Σ^* ist geeignet. Denn obwohl eine Turingmaschine z. B. ein Präfix der Länge k von m in polynomieller Zeit in k lesen kann, ist bei vielen effizienten Funktionen f die Laufzeit, um dieses Präfix aus $t(i, m)$ zu extrahieren, selbst für festes i nur noch in $|m|$ und nicht mehr in k polynomiell.

ist, und auch nicht klar ist, wie man die Konstruktion modifizieren kann, so daß man ein polynomiell-beschränktes Protokoll erhält. Dies möge das folgende Beispiel erläutern: Es sei ein Protokoll π gegeben, das das erste Bit, das es vom Angreifer erhält, speichert, und, wenn es von der Umgebung (oder vom umgebenden Protokoll) gefragt wird, dieses Bit zurückliefert. Die nebenläufige Komposition ohne Beschränkung der Session-IDs (selbst im Falle $f = 1$!) ist nun ein Protokoll, in dem der Angreifer eine (a priori) unbeschränkte Anzahl von Bits speichern kann (indem er dieses Bit an verschiedene Submaschinen schickt), und die Umgebung kann von diesen Bits eines auslesen. Mit einem polynomiell-beschränkten Protokoll ist dies prinzipiell nicht möglich, da das Protokoll eine a priori gegebene Schranke für die Anzahl der speicherbaren Bits haben muß. Versucht man dieses Problem zu umgehen, indem man verlangt, daß insgesamt nur $f(k)$ Instanzen des Protokolls entstehen dürfen, so erhält man eine Art „Fernwirkung“, weil Submaschinen die Arbeit verweigern müssen, weil z. B. *andere* Parteien mit *anderen* Session-IDs aufgerufen wurden. Der andere offensichtliche Ansatz, daß eine Maschine nur dann auf Nachrichten vom Angreifer reagieren darf, wenn sie bereits Nachrichten von der Umgebung erhalten hat (sozusagen „initialisiert“ wurde), ist auch nicht praktikabel, weil damit viele natürliche Protokolle ausgeschlossen würden. Bei einer sicheren Nachrichtenübertragung z. B. wird der Empfänger üblicherweise eine Nachricht an die Umgebung senden, weil er Daten über einen unsicheren Kanal erhalten hat (der unter Kontrolle des Angreifers steht) und nicht umgekehrt, so daß bereits eine natürliche Modellierung der sicheren Nachrichtenübertragung nicht mehr möglich ist.

Aus diesen Gründen verzichten wir auf diese zusätzliche Allgemeinheit und beschränken die Session-IDs auf eine feste Menge polynomieller Größe.

- Der zweite Punkt, der bei obiger Definition auffällt, ist ein eher subtiles Detail: Selbst wenn das Protokoll π polynomiell-beschränkt ist, und f ein Polynom ist, ist $f \cdot \pi$ noch nicht notwendig polynomiell-beschränkt. Dies erkennt man an folgendem Beispiel: Es sei $f = 2$ und M eine Maschine, die auf eine Nachricht auf Port `p` mit einer festen Nachricht antwortet und danach terminiert. Offensichtlich ist M polynomiell-beschränkt. Doch $f \cdot M$ zeigt folgendes Verhalten: Senden wir n Nachrichten der Form $(1, m)$ an $f \cdot M$ und dann eine Nachricht $(2, m)$, so wird $f \cdot M$ auf das erste $(1, m)$ antworten und auf $(2, m)$. Um dies zu tun, müssen alle n Nachrichten der Form $(1, m)$ zumindest daraufhin überprüft werden, ob sie nicht die Form $(2, m)$ haben. Da wir a priori keine Schranke für n kennen, hat auch die Laufzeit von $f \cdot M$ keine Schranke.

  Es ist möglich, diesem Problem wie folgt beizukommen: Es sei p ein Poly-

nom, das (für jeden eingehenden Port einzeln) die maximale Anzahl von Nachrichten angibt, die von M an diesem Port empfangen werden können, bevor der Port geschlossen wird (d. h. bevor an diesem Port keine Nachrichten mehr empfangen werden können, vgl. Abschnitt 2.2.3). Modifiziert man dann $f \cdot M$ insofern, daß $f \cdot M$ einen Port q schließt, wenn mehr als $p(k)$ Nachrichten der Form (i, m) mit gleichem i empfangen wurden, oder wenn eine ungültige Nachricht empfangen wurde, so erhält man eine polynomiell-beschränkte Maschine $\overline{f \cdot M}$ (vorausgesetzt, f ist effizient berechenbar und polynomiell-beschränkt). Man kann nun das nebenläufige Kompositiontheorem (s. u.) auch für diese veränderte Konstruktion zeigen.

Das einzige Ergebnis dieser Arbeit, welches explizit die Definition der nebenläufigen Komposition im *komplexitätstheoretischen* Fall benutzt, ist die Unsicherheit derselben bezüglich komplexitätstheoretischer spezieller Sicherheit in Kapitel 6. Es ist leicht einzusehen, daß die dort gezeigte Unsicherheit des komponierten Protokolls unabhängig davon ist, wie das Protokoll sich verhält, wenn es mehr Nachrichten erhält, als die simulierten Submaschinen „überleben würden“, da die im Beweis konkret angegebenen Angreifer und Umgebungen keine solche „überzähligen Nachrichten“ senden. Aus diesem Grund haben wir für die Präsentation unserer Ergebnisse die einfachere Definition der nebenläufigen Sicherheit gewählt. Der geneigte Leser kann die Allgemeingültigkeit der Ergebnisse an einer Definition der nebenläufigen Sicherheit seiner Wahl prüfen.

Wie auch bei der einfachen Komposition stellt sich hier die Frage, welche Sicherheitsbegriffe nebenläufige Komposition zulassen. Bereits in [Can01] wurde gezeigt, daß komplexitätstheoretische allgemeine Sicherheit mit Auxiliary input nebenläufig komponiert. Der Beweis überträgt sich direkt auf die anderen Varianten der allgemeinen Sicherheit mit Auxiliary input, und mit einer in [HMQS04] vorgestellten kleinen Anpassung auch auf die allgemeine Sicherheit ohne Auxiliary input. Zusammenfassend garantieren alle Varianten der allgemeinen Sicherheit nebenläufige Komposition. Dies faßt das folgende Theorem zusammen:

Theorem 2.26 (Nebenläufiges Kompositionstheorem)
Es bedeute $\pi \geq \rho$, daß π so sicher wie ρ ist bezüglich perfekter/statistischer/komplexitätstheoretischer *allgemeiner* Sicherheit mit/ohne Auxiliary input bzgl. der Ausgabe/der Sicht der Umgebung.

Seien π und ρ Protokolle, und f eine polynomiell-beschränkte Funktion. Im Falle der komplexitätstheoretischen Sicherheit seien außerdem π und ρ

(Fortsetzung nächste Seite)

(Fortsetzung)

polynomiell-beschränkt und f effizient berechenbar.
Dann ist $f \cdot \pi \geq f \cdot \rho$.

Beweisskizze: Wir betrachten zunächst den Fall der komplexitätstheoretischen allgemeinen Sicherheit ohne Auxiliary input bzgl. der Ausgabe der Umgebung. Die Beweise für die anderen Fälle sind sehr ähnlich und bringen keine neuen Einsichten, weshalb wir an dieser Stelle darauf verzichten, sie zu präsentieren.

Nach Lemma A.1 können wir ohne Beschränkung der Allgemeinheit davon ausgehen, daß alle betrachteten Umgebungen Ein-Bit-Ausgabe haben.

Um das Theorem zu beweisen, müssen wir für jeden polynomiell-beschränkten Angreifer $\mathcal{A}$ einen ebenfalls polynomiell-beschränkten Simulator $\mathcal{S}$ konstruieren, so daß für jede Umgebung $\mathcal{Z}$

$$\text{OUTPUT}_{f\cdot\pi,\mathcal{A},\mathcal{Z}} \qquad \text{und} \qquad \text{OUTPUT}_{f\cdot\rho,\mathcal{S},\mathcal{Z}} \tag{2.9}$$

ununterscheidbar sind (wir lassen wieder die Argumente k und z der Übersichtlichkeit halber weg).

Wir werden dies zunächst für einen ganz speziellen Angreifer zeigen, den *Dummy-Angreifer* $\tilde{\mathcal{A}}_f$, und danach einsehen, daß daraus die Sicherheit für alle anderen Angreifer folgt.

Der Dummy-Angreifer $\tilde{\mathcal{A}}_f$ ist wie folgt konstruiert: Er ist mit jedem Angreiferport des Protokolls $f \cdot \pi$ verbunden und hat außerdem zu jedem eingehenden Angreiferport einen ausgehenden Umgebungsport und zu jedem ausgehenden Angreiferport einen eingehenden Umgebungsport. Jede vom Protokoll kommende Nachricht leitet er direkt zur Umgebung weiter (über den entsprechenden Umgebungsport), und jede von der Umgebung kommende Nachricht wird über den entsprechenden Angreiferport weitergeleitet.[17]

Ganz analog definieren wir den Dummy-Angreifer $\tilde{\mathcal{A}}_\pi$ für das Protokoll π (statt $f \cdot \pi$). Da nach Annahme π so sicher wie ρ ist, gibt es einen polynomiell-beschränkten Simulator $\tilde{\mathcal{S}}_\rho$ zu $\tilde{\mathcal{A}}_\pi$, so daß für jede polynomiell-beschränkte Umgebung $\mathcal{Z}$

$$\text{OUTPUT}_{\pi,\tilde{\mathcal{A}}_\pi,\mathcal{Z}} \qquad \text{und} \qquad \text{OUTPUT}_{\rho,\tilde{\mathcal{S}}_\rho,\mathcal{Z}} \tag{2.10}$$

komplexitätstheoretisch ununterscheidbar sind.

[17] Hier ist Vorsicht geboten: $\tilde{A}_f$ ist nicht polynomiell-beschränkt, da er beliebig viele Nachrichten weiterleiten kann. In einem genauen Beweis muß man daher die Anzahl und Länge der Nachrichten, die er weiterleitet, beschränken. Dies ist aber nicht weiter problematisch, da eine polynomielle Schranke für die Kommunikation des Protokolls π bekannt ist.

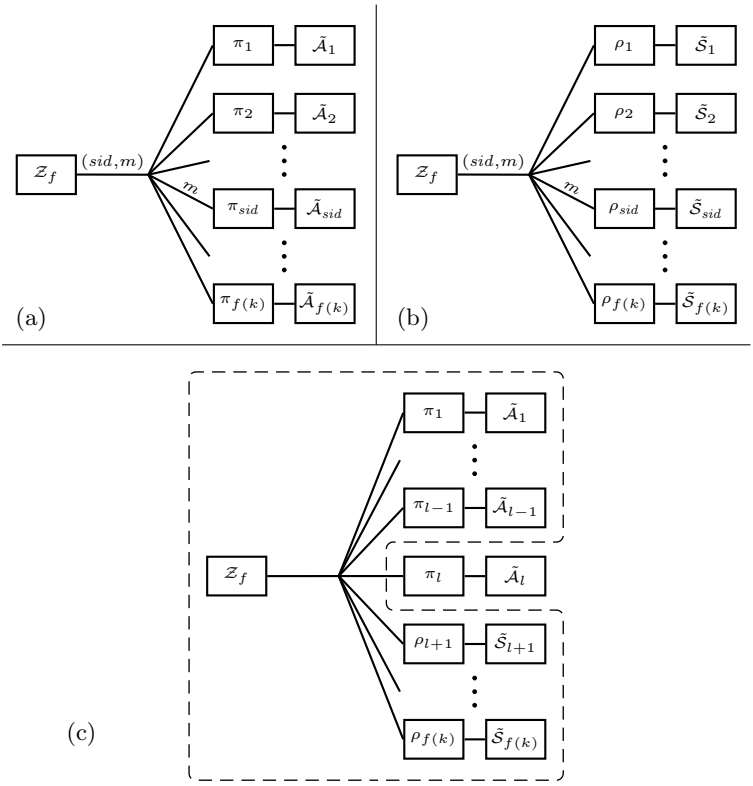


Abbildung 2.8.: (a) Konstruktion von O_f. $\mathcal{Z}_f$ läuft zusammen mit $f(k)$ Kopien des Protokolls π und des Angreifers $\tilde{\mathcal{A}}$. Eine Nachricht (sid, m) von $\mathcal{Z}_f$ wird als m an π_{sid} bzw. $\tilde{\mathcal{A}}_{sid}$ weitergeleitet, und umgekehrt wird eine Nachricht m von π oder $\tilde{\mathcal{A}}_{sid}$ als (sid, m) an $\mathcal{Z}_f$ ausgeliefert. Die Verbindungen zwischen $\mathcal{Z}_f$ und den Instanzen $\mathcal{A}_i$ des Angreifers sind der Übersichtlichkeit halber nicht eingezeichnet. Die Ausgabe von $\mathcal{Z}_f$ in diesem Netzwerk wird mit O_f bezeichnet.

(b) Konstruktion von O_0. Wie (a), nur läuft $\mathcal{Z}_f$ hier mit Kopien des Protokolls ρ und des Simulators $\tilde{\mathcal{S}}_\rho$.

(c) Konstruktion von O_l. Es handelt sich um eine Mischung von (a) und (b). Die ersten l Instanzen sind Kopien von π und $\tilde{\mathcal{A}}_\pi$ wie in (a), die restlichen $f(k)-l$ Instanzen sind Kopien von ρ und $\tilde{\mathcal{S}}_\rho$ wie in (b). Die Ausgabe von $\mathcal{Z}_f$ in diesem hybriden Konstrukt wird durch O_l bezeichnet.

Die gestrichelte Umrandung erläutert die Konstruktion der Maschine $\mathcal{Z}_l$. Diese simuliert die Maschinen innerhalb der Umrandung, nur π_l und $\tilde{\mathcal{A}}_l$ werden nicht mitsimuliert. Die Abbildung (c) repräsentiert somit nicht nur die Konstruktion von O_l, sondern auch einen Protokollauf von $\mathcal{Z}_l$ mit π_l und $\tilde{\mathcal{A}}_l$.

Es sei nun eine beliebige Umgebung $\mathcal{Z}_f$ für das Protokoll $f \cdot \pi$ gegeben. Wir betrachten die folgende Konstruktion (vgl. Abbildung 2.8a): Wir lassen die Maschine $\mathcal{Z}_f$, sowie $f(k)$ Kopien des Protokolls π (die wir mit $\pi_1, \ldots, \pi_{f(k)}$ bezeichnen) und $f(k)$ Kopien des Dummy-Angreifers $\tilde{\mathcal{A}}_\pi$ (mit $\tilde{\mathcal{A}}_1, \ldots, \tilde{\mathcal{A}}_{f(k)}$ bezeichnet) laufen. Dabei verbinden wir jeweils den Angreifer $\tilde{\mathcal{A}}_i$ mit der Protokollinstanz π_i. Schickt die Umgebung $\mathcal{Z}_f$ eine Nachricht (sid, m) an das Protokoll oder an den Angreifer, so wird m an π_{sid} bzw. $\tilde{\mathcal{A}}_{sid}$ ausgeliefert. Entsprechend werden Nachrichten m von π_{sid} oder $\tilde{\mathcal{A}}_{sid}$ zu $\mathcal{Z}$ als (sid, m) an die Umgebung $\mathcal{Z}_f$ weitergeleitet. Wir bezeichnen die Ausgabe von $\mathcal{Z}_f$ in einem Lauf dieses Netzwerks mit O_f.

Vergleicht man diese Konstruktion mit der Definition 2.24 der nebenläufigen Komposition und der Definition des Dummy-Angreifers, so erkennt man, daß das soeben beschriebene Netzwerk nichts anderes ist als die Umgebung $\mathcal{Z}_f$ zusammen mit dem Dummy-Angreifer $\tilde{\mathcal{A}}_f$ und dem komponierten Protokoll $f \cdot \pi$, denn das Protokoll $f \cdot \pi$ ist eine Simulation von $f(k)$ Instanzen von π. Also ist

$$O_f \;=\; \mathrm{OUTPUT}_{f\cdot\pi, \mathcal{Z}_f, \tilde{\mathcal{A}}_f}.$$

Eine ganz analoge Konstruktion erhalten wir, wenn wir statt Instanzen von π und $\tilde{\mathcal{A}}_\pi$ Instanzen ρ_i und $\tilde{\mathcal{S}}_i$ des idealen Protokolls ρ und des Simulators $\tilde{\mathcal{S}}_\rho$ verwenden (vgl. Abbildung 2.8b). Nennen wir die Ausgabe von $\mathcal{Z}_f$ in diesem Szenario O_0, so existiert ein polynomiell-beschränkter Simulator $\tilde{\mathcal{S}}_f$, so daß

$$O_0 \;=\; \mathrm{OUTPUT}_{f\cdot\rho, \mathcal{Z}_f, \tilde{\mathcal{S}}_f}.$$

Diesen Simulator $\tilde{\mathcal{S}}_f$ erhalten wir, indem wir die einzelnen Instanzen $\tilde{\mathcal{S}}_i$ des Simulators $\tilde{\mathcal{S}}_\rho$ zu einer Maschine kombinieren.

Um (2.9) im Falle $\mathcal{A} = \tilde{\mathcal{A}}_f$ zu zeigen, genügt es also einzusehen, daß O_f und O_0 komplexitätstheoretisch ununterscheidbar sind. Dies werden wir nun mit einem sogenannten hybriden Argument beweisen. Dazu definieren wir zunächst Zwischenstufen zwischen O_0 und O_f, in denen sowohl reale als auch ideale Protokollinstanzen vorkommen (daher der Name „hybrides Argument“). Es sei $l \in \mathbb{N}$ (möglicherweise abhängig vom Sicherheitsparameter k). Wir konstruieren dann ein Netzwerk, indem $\mathcal{Z}_f$ zusammen mit l Instanzen von π und $\tilde{\mathcal{A}}_\pi$ und mit $f - l$ Instanzen von ρ und $\tilde{\mathcal{S}}_\rho$ ausführen (vgl. Abbildung 2.8c, die gestrichelte Linie kann vorerst ignoriert werden). Die Instanzen von π und $\tilde{\mathcal{A}}_\pi$ erhalten die Indizes $1, \ldots, l$, wohingegen die Instanzen von ρ und $\tilde{\mathcal{S}}_\rho$ die Indizes $l+1, \ldots, f(k)$ erhalten. Die Kommunikation zwischen den Maschinen wird wie schon in der Konstruktion von O_0 und O_f geregelt. Insbesondere wird eine Nachricht (sid, m) von $\mathcal{Z}_f$ für $sid \leq l$ an eine Kopie von π bzw. $\tilde{\mathcal{A}}_\pi$ weitergeleitet, während sie für $sid > l$ an eine Kopie von ρ bzw. $\tilde{\mathcal{S}}_\rho$ ausgeliefert wird. Wir bezeichnen die Ausgabe von $\mathcal{Z}_f$ in diesem Netzwerk mit O_l.

Man sieht direkt, daß O_0 und O_f wie oben definiert Spezialfälle von O_l mit $l = 0$ bzw. $l = f$ sind.

Um einzusehen, daß O_0 und O_f komplexitätstheoretisch ununterscheidbar sind, führen wir zunächst noch eine weitere Konstruktion ein. Für ein $l \in \{1, \ldots, f(k)\}$ können wir wir die Maschinen $\mathcal{Z}_f, \pi_1, \ldots, \pi_{l-1}, \rho_{l+1}, \ldots, \rho_{f(k)}$ und $\tilde{\mathcal{A}}_1, \ldots, \tilde{\mathcal{A}}_{l-1}, \tilde{\mathcal{S}}_{l+1}, \ldots, \tilde{\mathcal{S}}_{f(k)}$, zu einer Maschine $\mathcal{Z}^l$ kombinieren (welche alle diese Maschinen simuliert). Diese Maschine ist in Abbildung 2.8c durch eine gestrichelte Umrandung dargestellt. Wir betrachten nun einen Protokollauf von $\mathcal{Z}^l$ zusammen mit π und $\tilde{\mathcal{A}}_\pi$. Da $\mathcal{Z}^l$ sämtliche in Abbildung 2.8c dargestellten Maschinen bis auf π_l und $\tilde{\mathcal{A}}_l$ simuliert, ist dies bis auf eine Umgruppierung der Maschinen das gleiche wie ein Protokollauf von O_l, also ist

$$\mathrm{OUTPUT}_{\pi, \tilde{\mathcal{A}}_\pi, \mathcal{Z}^l} = O_l.$$

Analog erhalten wir

$$\mathrm{OUTPUT}_{\rho, \tilde{\mathcal{S}}_\rho, \mathcal{Z}^l} = O_{l-1}.$$

Wenn wir nun eine Maschine $\mathcal{Z}^R$ konstruieren, die l gleichverteilt aus der Menge $\{1, \ldots, f(k)\}$ zieht und dann $\mathcal{Z}^l$ ausführt, so folgt für beliebiges $out \in \{0, 1, \perp\}$

$$P(\mathrm{OUTPUT}_{\pi, \tilde{\mathcal{A}}_\pi, \mathcal{Z}^R} = out) = \sum_{l=1}^{f(k)} \frac{1}{f(k)} P(O_l = out)$$

und

$$P(\mathrm{OUTPUT}_{\rho, \tilde{\mathcal{S}}_\rho, \mathcal{Z}^R} = out) = \sum_{l=1}^{f(k)} \frac{1}{f(k)} P(O_{l-1} = out).$$

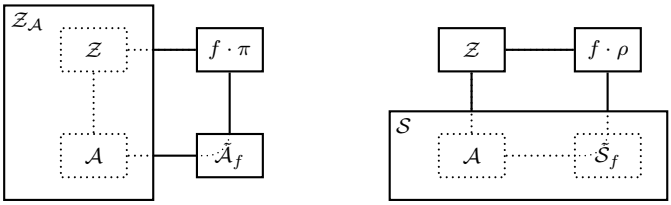
Da aber $\tilde{\mathcal{S}}_\rho$ der zu $\tilde{\mathcal{A}}_\pi$ gehörige Simulator ist (vgl. (2.10)), existiert also eine vernachlässigbare Funktion μ, so daß

$$\left| \sum_{i=1}^{f(k)} \frac{1}{f(k)} P(O_i = out) \; - \; \sum_{i=1}^{f(k)} \frac{1}{f(k)} P(O_{i-1} = out) \right| < \mu(k).$$

Damit ist

$$\begin{aligned} &\big| P(O_f = out) - P(O_0 = out) \big| \\ &\qquad = \left| \sum_{i=1}^{f(k)} P(O_i = out) \; - \; \sum_{i=1}^{f(k)} P(O_{i-1} = out) \right| < f(k)\mu(k). \end{aligned}$$

Da μ vernachlässigbar und f polynomiell-beschränkt ist, ist auch $f\mu$ vernachlässigbar. Da wir angenommen hatten, daß alle betrachteten Umgebungen Ein-Bit-Ausgabe haben, und somit O_0 und O_f nur Werte aus $\{0, 1, \perp\}$ annehmen



(a) (b)

Abbildung 2.9.: (a) Konstruktion der Umgebung $\mathcal{Z}_\mathcal{A}$. (b) Konstruktion des Simulators $\mathcal{S}$.

können, folgt damit, daß O_0 und O_f komplexitätstheoretisch ununterscheidbar sind.

Damit ist, wie bereits oben diskutiert, (2.9) im Falle $\mathcal{A} = \tilde{\mathcal{A}}_f$ gezeigt. Dies beschließt den ersten Teil des Beweises.

Es bleibt einzusehen, daß es hier tatsächlich hinreichend ist, (2.9) nur für den Dummy-Angreifer $\tilde{\mathcal{A}}_f$ zu zeigen.

Wir nehmen dazu an, es seien ein Angreifer $\mathcal{A}$ und eine Umgebung $\mathcal{Z}$ für das Protokoll $f \cdot \pi$ gegeben. Wir können nun eine Umgebung $\mathcal{Z}_\mathcal{A}$ konstruieren, die $\mathcal{Z}$ und $\mathcal{A}$ simuliert, und alle Nachrichten, die $\mathcal{A}$ an das Protokoll schicken will und von dort empfängt, durch den Dummy-Angreifer $\tilde{\mathcal{A}}_f$ leitet. Da dieser alle Nachrichten zwischen Umgebung $\mathcal{Z}_\mathcal{A}$ und Protokoll unverändert weiterleitet, ist dies äquivalent dazu, daß $\mathcal{A}$ diese Nachrichten direkt an das Protokoll schickt, vgl. Abbildung 2.9a. Es ist somit

$$\mathrm{OUTPUT}_{f\cdot\pi,\mathcal{A},\mathcal{Z}} = \mathrm{OUTPUT}_{f\cdot\pi,\tilde{\mathcal{A}}_f,\mathcal{Z}_\mathcal{A}}.$$

Da (2.9) für den Dummy-Angreifer gilt, sind

$$\mathrm{OUTPUT}_{f\cdot\pi,\tilde{\mathcal{A}}_f,\mathcal{Z}_\mathcal{A}} \quad \text{und} \quad \mathrm{OUTPUT}_{f\cdot\rho,\tilde{\mathcal{S}}_f,\mathcal{Z}_\mathcal{A}}$$

ununterscheidbar.

Nun können wir einen Simulator $\mathcal{S}$ konstruieren, der aus $\mathcal{A}$ und $\tilde{\mathcal{S}}_f$ besteht (vgl. Abbildung 2.9b). Offensichtlich ist

$$\mathrm{OUTPUT}_{f\cdot\rho,\tilde{\mathcal{S}}_f,\mathcal{Z}_\mathcal{A}} = \mathrm{OUTPUT}_{f\cdot\rho,\mathcal{S},\mathcal{Z}}.$$

Damit sind aber

$$\mathrm{OUTPUT}_{f\cdot\pi,\mathcal{A},\mathcal{Z}} \quad \text{und} \quad \mathrm{OUTPUT}_{f\cdot\rho,\mathcal{S},\mathcal{Z}}$$

ununterscheidbar, und da $\mathcal{S}$ unabhängig von $\mathcal{Z}$ gewählt wurde (er hängt nur von $\mathcal{A}$ und $\tilde{\mathcal{S}}_\pi$ ab), folgt damit (2.9) für $\mathcal{A}$. Also ist $f \cdot \pi$ so sicher wie $f \cdot \rho$. Dies beendet die Beweisskizze für den Fall der komplexitätstheoretischen allgemeinen Sicherheit ohne Auxiliary input bzgl. der Ausgabe der Umgebung.

Die anderen Fälle werden analog gezeigt. Bei der Sicherheit bzgl. der Sicht der Umgebung beachte man, daß sich die Sicht von $\mathcal{Z}$ effizient aus der Sicht von $\mathcal{Z}^R$ bzw. $\mathcal{Z}_\mathcal{A}$ extrahieren läßt. □

Man mag sich an dieser Stelle fragen, ob die Einschränkung auf *allgemeine* Sicherheit für die Gültigkeit des nebenläufigen Kompositionstheorems unbedingt notwendig ist. Ein Blick auf die oben präsentierte Beweisskizze zeigt uns, daß zumindest die obige Beweistechnik im Falle der speziellen Sicherheit nicht funktioniert. So hängt in diesem Fall der Simulator $\tilde{\mathcal{S}}_\rho$ von der betrachteten Umgebung $\mathcal{Z}_l$ ab, diese aber ist wiederum in Abhängigkeit von $\tilde{\mathcal{S}}_\rho$ definiert (denn sie simuliert Kopien von $\tilde{\mathcal{S}}_\rho$). Und in Kapitel 6 beweisen wir, daß das nebenläufige Kompositionstheorem im Falle der komplexitätstheoretischen speziellen Sicherheit tatsächlich nicht gilt.

2.3.4. Allgemeine Komposition

Wie zu Beginn von Abschnitt 2.3.3 erwähnt, ist der allgemeine Fall der Komposition der folgende: Ein Protokoll σ benutzt *polynomiell viele* Instanzen einer idealen Funktionalität $\mathcal{F}$. Gegeben sei weiter ein Protokoll π, welches so sicher wie $\mathcal{F}$ ist. Man stellt sich jetzt die Frage, ob man gefahrlos jede Instanz von $\mathcal{F}$ durch σ ersetzen kann, sprich ob ein Kopien von π benutzendes σ so sicher wie ein Kopien von ρ benutzendes σ ist. Am Anfang von Abschnitt 2.3.3 hatten wir dafür die etwas ungenaue Notation

$$\sigma^{\pi_1,\ldots,\pi_f} \geq \sigma^{\mathcal{F}_1,\ldots,\mathcal{F}_f} \tag{2.11}$$

verwendet (hierbei ist f eine Funktion im Sicherheitsparameter). Unter Benutzung der im vorangegangenen Abschnitt eingeführten Mittel können wir dies nun genauer fassen. Anstatt z. B. davon zu sprechen, daß σ f Instanzen von π aufrufe, können wir ebensogut sagen, daß σ *eine* Instanz des Protokolls $f \cdot \pi$ benutze (welches wiederum aus f Instanzen von π besteht). Dafür können wir also einfach $\sigma^{f\cdot\pi}$ schreiben. Wir können nun (2.11) als

$$\sigma^{f\cdot\pi} \geq \sigma^{f\cdot\mathcal{F}}$$

schreiben (wobei $\geq$ „so sicher wie“ bedeute), die allgemeinen Komposition ist also nichts anderes als die Kombination der einfachen Komposition (Definition 2.18) und der nebenläufigen Komposition (Definition 2.24).

Das folgende Theorem sagt uns, daß im Falle der allgemeinen Sicherheit allgemeine Komposition ohne Verlust an Sicherheit möglich ist.

Theorem 2.27 (Allgemeines Kompositionstheorem)
Es bedeute $\pi \geq \rho$, daß π so sicher wie ρ ist bezüglich perfekter/statistischer/komplexitätstheoretischer *allgemeiner* Sicherheit mit/ohne Auxiliary input bzgl. der Ausgabe/der Sicht der Umgebung.

Seien π, ρ und σ Protokolle, so daß π und ρ in σ einbettbar sind, und so daß $\pi \geq \rho$ gilt. Weiter sei f eine polynomiell-beschränkte Funktion.

Im Falle der komplexitätstheoretischen Sicherheit seien außerdem π, ρ und σ polynomiell-beschränkt und f effizient berechenbar.

Dann ist $\sigma^{f \cdot \pi} \geq \sigma^{f \cdot \rho}$.

Beweisskizze: Das nebenläufigen Kompositionstheorem 2.26 liefert $f \cdot \pi \geq f \cdot \rho$. Daraus folgt mit dem einfachen Kompositionstheorem 2.19 unmittelbar $\sigma^{f \cdot \pi} \geq \sigma^{f \cdot \rho}$.[18] □

In Abschnitt 2.3.2 haben wir uns die Frage gestellt, inwiefern spezielle Sicherheit nicht nur hinreichend, sondern auch *notwendig* für einfache Komposition ist. Eine ähnliche Frage können wir uns natürlich auch bei der allgemeinen Komposition stellen. Dazu müssen wir zunächst definieren, was es für ein Protokoll bedeutet, allgemein komponierbar zu sein (vgl. die Diskussion in Abschnitt 2.3.2). [Lin03] folgend, führen wir den Begriff der *F-beschränkten allgemeinen Komponierbarkeit* ein. Ein Protokoll π ist so sicher wie ein anderes Protokoll ρ bezüglich *F-beschränkter allgemeiner Komponierbarkeit*, wenn für jede polynomiell-beschränkte Funktion $f \in F$ und jedes Protokoll σ gilt, daß $\sigma^{f \cdot \pi}$ so sicher wie $\sigma^{f \cdot \rho}$ ist. In anderen Worten, bis zu f Instanzen von ρ können ohne Verlust an Sicherheit durch Instanzen von π ersetzt werden. (Im Falle der komplexitätstheoretischen Sicherheit müssen dabei f effizient berechenbar und σ polynomiell-beschränkt sein.) Wie schon in Abschnitt 2.3.2 verwenden wir dabei als zugrundeliegenden Sicherheitsbegriff für diese Definition die Sicherheit ohne Umgebung aus Definition 2.21. Das ergibt die folgende Definition (analog zu Definition 2.22):

[18] Bei einer genauen Betrachtung der Details wird man feststellen, daß im komplexitätstheoretischen Fall das einfache Kompositionstheorem wie in Theorem 2.19 formuliert hier nicht ohne weiteres anwendbar ist, da $f \cdot \pi$ und $f \cdot \rho$ strenggenommen nicht polynomiell-beschränkt sind (vgl. die Diskussion nach Definition 2.25). Konstruiert man aber Protokolle $f \cdot \pi'$ und $f \cdot \rho'$, die sich wie $f \cdot \pi$ und $f \cdot \rho$ verhalten, aber nach einer hinreichen großen (aber polynomiell-beschränkten) Anzahl von Nachrichten auf einem Port diesen schließen, so sieht man leicht $f \cdot \pi' \geq f \cdot \rho'$ ein. Darauf kann man dann das einfache Kompositionstheorem 2.19 anwenden.

Definition 2.28 (Allgemeine Komponierbarkeit)
Es sei F eine Menge von Funktionen von $\mathbb{N}$ nach $\mathbb{N}_0$, und es seien π und ρ prinzipiell einbettbare Protokolle.

Dann heißt π so sicher wie ρ bezüglich perfekter/statistischer/komplexitätstheoretischer allgemeiner F-beschränkter Komponierbarkeit mit/ohne Auxiliary input, wenn für jedes (im komplexitätstheoretischen Falle polynomiell-beschränkte) Protokoll σ und jede (im komplexitätstheoretischen Falle effizient berechenbare) Funktion $f \in F$ gilt, daß $\sigma^{f \cdot \pi}$ so sicher wie $\sigma^{f \cdot \rho}$ ist bezüglich perfekter/statistischer/komplexitätstheoretischer Sicherheit ohne Umgebung mit/ohne Auxiliary input.

Wichtige Fälle der allgemeinen Komponierbarkeit sind:

- Polynomiell-beschränkte allgemeine Komponierbarkeit. Diesen Fall meint man meistens, wenn man einfach nur von allgemeiner Komponierbarkeit spricht, da wir für superpolynomielles f im allgemeinen keine Komponierbarkeit mehr erwarten dürfen. Im folgenden werden wir einfach kurz *allgemeine Komponierbarkeit* zur polynomiell-beschränkten allgemeinen Komponierbarkeit sagen.

- 1-beschränkte Komponierbarkeit. Es handelt sich hierbei im Prinzip um einfache Komponierbarkeit.

- $O(1)$-beschränkte Komponierbarkeit. Hier lassen wir das Ersetzen einer variablen Anzahl von Instanzen zu, jedoch muß es eine feste (vom Sicherheitsparameter unabhängige) Schranke für deren Anzahl geben. Dieser Begriff ist insofern wichtig, als er leicht als zur speziellen Sicherheit äquivalent erkannt werden kann (s. u.).

Die folgenden Zusammenhänge ergeben sich nun sehr leicht aus den bisher gewonnenen Erkenntnissen:

Lemma 2.29 (Allgemeine Komponierbarkeit)
Folgendes gilt im perfekten/statistischen/komplexitätstheoretischen Fall mit/ohne Auxiliary input bzgl. der Ausgabe der Umgebung:

Es seien π und ρ prinzipiell einbettbare Protokolle. Im komplexitätstheoretischen Falle seien π und ρ außerdem polynomiell-beschränkt.

Dann gilt (i) $\Rightarrow$ (ii) $\Rightarrow$ (iii) für die folgenden Aussagen:

(Fortsetzung nächste Seite)

(Fortsetzung)

(i) π ist so sicher wie ρ bezüglich allgemeiner Sicherheit.
(ii) π ist so sicher wie ρ bezüglich polynomiell-beschränkter allgemeiner Komponierbarkeit.
(iii) π ist so sicher wie ρ bezüglich spezieller Sicherheit.

Weiterhin sind die folgenden Aussagen äquivalent:
(iv) π ist so sicher wie ρ bezüglich spezieller Sicherheit.
(v) π ist so sicher wie ρ bezüglich $O(1)$-beschränkter allgemeiner Komponierbarkeit.
(vi) π ist so sicher wie ρ bezüglich 1-beschränkter allgemeiner Komponierbarkeit.

Beweisskizze: Wir beweisen zunächst die Äquivalenz der Aussagen (iv,v,vi). Durch wiederholte Anwendung des einfachen Kompositionstheorems 2.19 und unter Ausnutzung der Transitivität der Sicherheit (Lemma 2.20) erhalten wir (iv) $\Rightarrow$ (v). Trivialerweise gilt (v) $\Rightarrow$ (vi). Und da man leicht einsieht, daß 1-beschränkte allgemeine Komponierbarkeit das gleiche ist wie einfache Komponierbarkeit, folgt mit Theorem 2.23 (vi) $\Rightarrow$ (iv). Es folgt die Äquivalenz von (iv,v,vi).

Wir zeigen nun (i) $\Rightarrow$ (ii) $\Rightarrow$ (iii). Aus dem allgemeinen Kompositionstheorem 2.27 folgt (i) $\Rightarrow$ (ii). Weiterhin gilt trivialerweise (ii) $\Rightarrow$ (v) und (iv) $\Leftrightarrow$ (iii), woraus sich mit den oben bewiesenen Äquivalenzen (ii) $\Rightarrow$ (iii) ergibt. □

Angesichts der Äquivalenz von spezieller Sicherheit und $O(1)$-beschränkter Komponierbarkeit kann man sich fragen, ob ein vergleichbares Resultat auch für die polynomiell-beschränkte allgemeine Komponierbarkeit gilt. Ist z. B. die polynomiell-beschränkte allgemeine Komponierbarkeit äquivalent zur allgemeinen Sicherheit? Oder wird sie schon von der speziellen Sicherheit impliziert? Fallen vielleicht sogar alle betrachteten Begriffe zusammen? Oder sind allgemeine Sicherheit, polynomiell-beschränkte allgemeine Komponierbarkeit und spezielle Sicherheit alle verschieden? Wir werden in den folgenden Kapiteln sehen, daß die Antworten auf diese Fragen von der verwendeten Variante des Sicherheitsbegriffs abhängen. So kann jede der Fragen sowohl mit ja als auch mit nein beantwortet werden, je nachdem, ob wir perfekte, statistische oder komplexitätstheoretische Sicherheit untersuchen.

3. Die Mächtigkeit zufälliger Angriffe

In diesem Kapitel wollen widmen wir uns den Varianten der perfekten Sicherheit. Es stellt sich heraus, daß hier die Klassifikation sehr einfach ist, alle Varianten sind äquivalent. Die Technik die diesen Ergebnissen zugrundeliegt ist die des zufälligen Angriffes; wir zeigen daß Umgebungen, die zufällig unter allen möglichen Angriffsstrategien wählen, ebenso mächtig sind wie speziell auf das Protokoll zugeschnittene.

3.1. Spezielle und allgemeine Sicherheit

Um zu zeigen, daß spezielle und allgemeine Sicherheit äquivalent sind, werden wir im folgenden eine Umgebung konstruieren, die ein zufälliges Verhalten zeigt, d. h. eine Umgebung, die bei jeder Aktivierung zufällige Ausgaben hat. Dann werden wir einsehen, daß – wenn überhaupt eine erfolgreiche, also reales und ideales Protokoll unterscheidende Umgebung existiert – schon diese zufällige Umgebung unterscheidet. Der entscheidende Punkt ist hier, daß bei der perfekten Sicherheit schon beliebig kleine Wahrscheinlichkeiten eine Rolle spielen. Da jeder mögliche Angriff der zufälligen Umgebung eine extrem kleine, aber nicht völlig verschwindende Wahrscheinlichkeit hat, hat auch der erfolgreiche, auf das Protokoll zugeschnittene Angriff eine Wahrscheinlichkeit größer als Null und wird deshalb zur Unterscheidung führen.

Zunächst definieren wir die „zufällige Umgebung“. Im wesentlichen gibt es nur eine zufällige Umgebung, jedoch muß die Umgebung zum Protokoll und zum Angreifer passende Ports haben, um mit ihnen kommunizieren zu können.

Definition 3.1 (Zufällige Umgebung)
Es sei $\mathcal{D}$ eine Wahrscheinlichkeitsverteilung auf Σ^*, so daß jedes Wort in Σ^* eine Wahrscheinlichkeit größer 0 hat.[1]

Es seien π und ρ Protokolle und $\mathcal{A}$ ein Angreifer. Die *zu π, ρ und $\mathcal{A}$ passende zufällige Umgebung* ist die Maschine $\mathcal{Z}_?$, die die folgenden Eigenschaften hat:

- Es seien $self, loose \in \Sigma^*$ Umgebungsports, die nicht Ports von π, ρ oder $\mathcal{A}$ sind.

(Fortsetzung nächste Seite)

(Fortsetzung)

- Der Name $name_{\mathcal{Z}_?}$ der Maschine ist `env`.
- Die Menge $in(\mathcal{Z}_?)$ der *eingehenden* Ports besteht aus (i) den *ausgehenden* Umgebungsports von $\mathcal{A}$ und den Protokoll*ausgabe*ports von π und den Protokoll*ausgabe*ports von ρ, (ii) dem Umgebungsport *self* und (iii) dem Spezialport `input`.
- Die Menge $out(\mathcal{Z}_?)$ der *ausgehenden* Ports besteht aus (i) den *eingehenden* Umgebungsports von $\mathcal{A}$ und den Protokoll*eingabe*ports von π und den Protokoll*eingabe*ports von ρ, (ii) dem Umgebungsport *self* und (iii) dem Umgebungsport *loose*.
- Bei jeder Aktivierung von $\mathcal{Z}_?$ wird eine zufällige Nachricht m gemäß der Verteilung $\mathcal{D}$ gewählt, sowie ein gleichverteilt zufälliger ausgehender Port p (wobei auch $p = \lambda$ erlaubt ist, d. h. keine Nachricht wird gesandt). Dann sendet $\mathcal{Z}_?$ die Nachricht m über den Port p. Der Zustand von $\mathcal{Z}_?$ nach der Aktivierung wird zufällig gemäß $\mathcal{D}$ gewählt (unabhängig von m und p).
  Der vorherige Zustand und die eingehende Nachricht werden hierbei ignoriert.
  Formal bedeutet dies, daß die Zustandsübergangsfunktion $\mathcal{Z}_?$ wie folgt definiert ist:

  $$\mathcal{Z}_?(k, s, p, m) := \mathcal{D} \times \mathcal{U}(out(\mathcal{Z}_?) \cup \{\lambda\}) \times \mathcal{D}.$$

  Hierbei bezeichnet $\mathcal{U}(X)$ die Gleichverteilung auf der Menge X und $\times$ das Produkt von Wahrscheinlichkeitsmaßen.

In obiger Definition sind der Name und die Menge der ein- und ausgehenden Ports gerade so gewählt, daß $\mathcal{Z}_?$ eine zulässige Umgebung für $\mathcal{A}$ ist (gemäß Definition 2.12) und Verbindungen zu allen Protokollports der Protokolle und allen Umgebungsports des Angreifers hat (also zu allen Ports, zu denen eine Umgebung überhaupt Verbindungen haben darf). Aus beweistechnischen Gründen hat $\mathcal{Z}_?$ zusätzlich noch eine Verbindung *self* zu sich selbst, und einen unverbundenen Port *loose*. Die zufällige Umgebung $\mathcal{Z}_?$ schickt dann in jeder Aktivierung eine zufällige Nachricht über einen zufälligen Port und geht in einen zufälligen Zustand über. Man kann also sagen, daß $\mathcal{Z}_?$ ein im wesentlichen von Protokoll und Angreifer unabhängiges Verhalten zeigt.

Das folgende Lemma zeigt, daß diese Umgebung in allen Situationen unterscheidet, in denen es überhaupt eine unterscheidende Umgebung gibt:

[1] Da Σ^* abzählbar ist, existiert eine solche Verteilung. Ein Beispiel für eine solche Verteilung ist die, die jedem Wort $x \in \Sigma^*$ die Wahrscheinlichkeit $\#\Sigma^{-|x|}2^{-|x|-1}$ zuordnet.

Lemma 3.2 (Universalität von $\mathcal{Z}_?$)
Es seien π und ρ Protokolle, $\mathcal{A}$ ein zulässiger Angreifer, $\mathcal{S}$ ein zulässiger Simulator und $\mathcal{Z}^*$ eine zulässige Umgebung. Weiterhin seien der Sicherheitsparameter $k \in \mathbb{N}$ und der Auxiliary input $z \in \Sigma^*$ gegeben. Es bezeichne $\mathcal{Z}_?$ die zu π, ρ und $\mathcal{A}$ passende zufällige Umgebung.

Ist

$$\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}^*}(k,z) \neq \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}^*}(k,z),$$

so ist auch

$$\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}_?}(k,z) \neq \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}_?}(k,z).$$

In anderen Worten, wenn die Sicht der ursprünglichen Umgebung $\mathcal{Z}^*$ im realen und idealen Modell verschieden ist, so ist es auch die Sicht der zufälligen Umgebung $\mathcal{Z}_?$.

Bevor wir das Lemma beweisen, sei noch angemerkt, daß dies natürlich nur gelten kann, wenn wir die Sicht der Umgebung betrachten; die *Ausgabe* der zufälligen Umgebung liefert keine Information über die empfangenen Nachrichten und kann deshalb nur schlecht zur Unterscheidung herangezogen werden. Um eine ähnliche Aussage über die Ausgabe der Umgebung zu machen, müßten wir die zufällige Umgebung anders definieren; die zufällige Umgebung würde dann zu einem zufälligen Zeitpunkt Ausgabe generieren, aber diese Ausgabe hätte nicht zufälligen Inhalt, sondern bestünde aus der Sicht der Umgebung bis zu diesem Zeitpunkt. Dann könnte man ein analoges Lemma auch bezüglich der Ausgabe (anstelle der Sicht) der Umgebung zeigen. Da wir aber die in diesem Abschnitt gewonnenen Erkenntnisse mit Korollar 3.7 aus Abschnitt 3.2.2 direkt auf den Fall der Ausgabe übertragen können, verzichten wir darauf, in diesem Abschnitt den komplizierteren Fall der Ausgabe mit zu berücksichtigen.

Beweis: Der Beweis gliedert sich in zwei Teile. Zunächst zeigen wir, daß wir o. B. d. A. annehmen können, daß $\mathcal{Z}^*$ die gleichen Ports, also die gleiche äußere Struktur hat wie die zufällige Umgebung $\mathcal{Z}_?$. Danach beweisen wir, daß es ein bestimmtes Präfix der Sicht von $\mathcal{Z}_?$ gibt, dessen Wahrscheinlichkeit im realen und im idealen Modell verschieden sind.

Wir führen die folgenden Abkürzungen ein: $R(\mathcal{Z}^*) := \mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}^*}(k,z)$ (die Sicht von $\mathcal{Z}^*$ im realen Modell), $I(\mathcal{Z}^*) := \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}^*}(k,z)$ (die Sicht von $\mathcal{Z}^*$ im idealen Modell) und analog für $\mathcal{Z}_?$. Die Voraussetzung des Lemmas lautet dann $R(\mathcal{Z}^*) \neq I(\mathcal{Z}^*)$ und zu zeigen ist $R(\mathcal{Z}_?) \neq I(\mathcal{Z}_?)$.

Es seien *self* und *loose* wie in Definition 3.1. Hat $\mathcal{Z}^*$ einen Port mit dem Namen *self*, so benennen wir diesen Port in *new* um, wobei *new* ein in π, ρ, $\mathcal{A}$, $\mathcal{S}$ und $\mathcal{Z}^*$ nicht vorkommender Portname sei (die resultierende Maschine heiße

$\mathcal{Z}'$). Da weder π, ρ, $\mathcal{A}$ noch $\mathcal{S}$ einen Port mit Namen *self* oder *new* haben, bewirkt dies lediglich eine Umbenennung jeden Vorkommens von *self* in der Sicht von $\mathcal{Z}^*$, es gibt also eine deterministische (meßbare) Funktion f, so daß $R(\mathcal{Z}^*) = f(R(\mathcal{Z}'))$ und $I(\mathcal{Z}^*) = f(I(\mathcal{Z}'))$. Damit ist auch $R(\mathcal{Z}') \neq I(\mathcal{Z}')$. Wir können also o. B. d. A. annehmen, daß $\mathcal{Z}^*$ keinen Port mit dem Namen *self* hat. Einem analogen Argument folgend können wir ebenfalls annehmen, daß $\mathcal{Z}^*$ keinen Port mit dem Namen *loose* hat.

Unter eine Selbstleitung von $\mathcal{Z}^*$ verstehen wir einen Port, der gleichzeitig ein- als auch ausgehender Port ist (d. h. die Menge S der Selbstleitungen ist $S = in(\mathcal{Z}^*) \cap out(\mathcal{Z}^*)$). Wir modifizieren $\mathcal{Z}^*$ wie folgt (und nennen das Resultat $\mathcal{Z}'$): Wir entfernen die Selbstleitungen von $\mathcal{Z}^*$ und geben $\mathcal{Z}'$ stattdessen eine Selbstleitung mit dem Namen *self*. Wann immer $\mathcal{Z}^*$ eine Nachricht m über einen Port $p \in S$ senden würde, sendet $\mathcal{Z}^*$ die Nachricht (p, m) über den ausgehenden Port *self*. Erhält $\mathcal{Z}'$ eine Nachricht (p, m) über den eingehenden Port *self*, so verhält sich $\mathcal{Z}'$ als hätte $\mathcal{Z}^*$ die Nachricht m über den eingehenden Port p bekommen (Nachrichten mit $p \notin S$ oder von ungültiger Form werden ignoriert). Wir haben also alle Selbstleitungen von $\mathcal{Z}^*$ zu einer Selbstleitung *self* gebündelt. Offensichtlich läßt sich die Sicht von $\mathcal{Z}^*$ aus der Sicht von $\mathcal{Z}'$ rekonstruieren, indem man in allen Einträgen in der Sicht, die eine ein- oder ausgehende Nachricht (p, m) über den Port *self* enthalten, diese Nachricht durch eine entsprechende Nachricht m über den Port p ersetzt. Also resultieren $R(\mathcal{Z}^*)$ und $I(\mathcal{Z}^*)$ wieder deterministisch aus $R(\mathcal{Z}')$ und $I(\mathcal{Z}')$, wir können o. B. d. A. annehmen, wir hätten gleich $\mathcal{Z}'$ statt $\mathcal{Z}^*$ vorliegen gehabt, d. h. daß $\mathcal{Z}^*$ genau eine Selbstleitung hat und daß diese den Namen *self* trägt.

Nun betrachten wir die ausgehenden Ports, die $\mathcal{Z}^*$ hat, $\mathcal{Z}_?$ aber nicht. Nach Konstruktion der zufälligen Umgebung $\mathcal{Z}_?$ können dies keine Ports sein, die π, ρ, $\mathcal{A}$ oder $\mathcal{S}$ hat. Aufgrund des vorangehenden Beweisschrittes kann es sich auch nicht um eine Selbstleitung handeln. Somit handelt es sich um Ports, die sowohl im realen als auch im idealen Modell (in $R(\mathcal{Z}^*)$ und $I(\mathcal{Z}^*)$) kein Gegenstück haben. Jede Nachricht, die über einen solchen Port p gesandt wird, führt zur sofortigen Aktivierung des Schedulers (hier $\mathcal{A}$). Wir können also – ähnlich wie oben – $\mathcal{Z}^*$ so ändern, daß statt einer Nachricht m über p eine Nachricht (p, m) über einen neuen Port *loose* gesandt wird (denn dieser hat ebenfalls kein Gegenstück). Wie oben kann die Sicht des ursprünglichen $\mathcal{Z}^*$ aus der des veränderten berechnet werden, also können wir o. B. d. A. annehmen, daß $\mathcal{Z}^*$ keine ausgehenden Ports hat außer denen, die auch $\mathcal{Z}_?$ hat.

Ähnlich sehen wir ein, daß jeder eingehende Port von $\mathcal{Z}^*$, den $\mathcal{Z}_?$ nicht hat, kein Gegenstück hat. Da über einen solchen Port nie Nachrichten ankommen können, enthält die Sicht von $\mathcal{Z}^*$ auch keine Einträge mit diesem Port. Wir können also $\mathcal{Z}^*$ durch eine Maschine ersetzen, die diesen Port gar nicht hat, ohne daß sich die Sicht ändert. Also können wir o. B. d. A. annehmen, daß $\mathcal{Z}^*$ keine ausgehenden Ports hat außer denen, die auch $\mathcal{Z}_?$ hat.

Zusammenfassend hat also $\mathcal{Z}^*$ eine Teilmenge der Ports von $\mathcal{Z}_?$. Wir können jetzt zu $\mathcal{Z}^*$ die Ports hinzufügen, die $\mathcal{Z}_?$ hat, $\mathcal{Z}^*$ aber nicht. Nachrichten auf den neuen eingehenden Ports ignorieren wir, auf den neuen ausgehenden Ports senden wir nichts. Wieder läßt sich die Sicht des ursprünglichen $\mathcal{Z}^*$ aus der des veränderten berechnen, indem wir die Aktivierungen aus der Sicht streichen, die durch eine Nachricht über einen neuen Port zustandegekommen sind.

Insgesamt können wir also o. B. d. A. annehmen, daß $\mathcal{Z}^*$ die gleichen ein- und die gleichen ausgehenden Ports wie $\mathcal{Z}_?$ hat.

Wir zeigen nun, daß die Sicht von $\mathcal{Z}_?$ im realen und im idealen Modell verschieden ist. Für eine endliche oder unendliche Folge x bezeichne $\mathrm{pfx}_n(x)$ das Präfix der Länge n von x, d. h. $\mathrm{pfx}_n(x) = (x_1, x_2, \ldots, x_n)$. (Falls $|x| < n$ setzen wir $\mathrm{pfx}_n(x) = x$.)

Wir erinnern uns, daß die Sicht der Umgebung eine Folge von Tupeln ist, wobei jedes Tupel eine Aktivierung der Umgebung repräsentiert (vgl. Definition 2.6). Wenn wir T als die Menge aller dieser Tupel bezeichnen, dann ist also $T^{\mathbb{N}} \cup T^*$ die Menge aller möglichen Sichten von $\mathcal{Z}_?$ (und auch von $\mathcal{Z}^*$). Man beachte, daß T abzählbar ist. Für jede endliche Folge $\beta \in T^*$ und jedes $n \geq |\beta|$ betrachten wir dann die Menge $P^m_\beta := \{v : \mathrm{pfx}_m(v) = \beta\}$, also die Menge aller Sichten, deren erste m Einträge β sind. Da die Mengen P_β mit $\beta \in T^*$ die σ-Algebra aller meßbaren Mengen über $T^{\mathbb{N}} \cup T^*$ erzeugen,[2] muß es, da $R(Z^*) \neq I(Z^*)$ gilt, ein $\beta \in T^*$ und ein $m \geq |\beta|$ geben, so daß

$$P\big(R(Z^*) \in P^m_\beta\big) \neq P\big(I(Z^*) \in P^m_\beta\big). \tag{3.1}$$

Um einen Widerspruch zu erzeugen, nehmen wir nun an, daß die zufällige Umgebung $\mathcal{Z}_?$ nicht unterscheide, also daß $R(\mathcal{Z}_?) = I(\mathcal{Z}_?)$. Wir wollen daraus nun folgern, daß für jedes $n \in \mathbb{N}$ gilt, daß

$$\mathrm{pfx}_n(R(\mathcal{Z}^*)) = \mathrm{pfx}_n(I(\mathcal{Z}^*)). \tag{3.2}$$

Im Falle $n = m$ stünde dies im Widerspruch zu (3.1), es gälte $R(\mathcal{Z}_?) \neq I(\mathcal{Z}_?)$, und das Lemma wäre gezeigt.

Um (3.2) zu beweisen, zeigen wir induktiv für alle $n \in \mathbb{N}_0$ die folgenden zwei Aussagen gleichzeitig:

$$A(n): \qquad \mathrm{pfx}_n(R(\mathcal{Z}^*)) = \mathrm{pfx}_n(I(\mathcal{Z}^*))$$

$$B(n): \qquad \forall \alpha \in T^n : P\big(\mathrm{pfx}_n(R(\mathcal{Z}^*)) = \alpha\big) > 0 \Rightarrow P\big(\mathrm{pfx}_n(R(\mathcal{Z}_?)) = \alpha\big) > 0$$

Dabei ist $A(n)$ gerade (3.2), und $B(n)$ brauchen wir, um im Induktionsschritt zu garantieren, daß alle betrachteten bedingten Wahrscheinlichkeiten definiert sind.

[2] Denn so ist der kanonische Meßraum über $T^{\mathbb{N}} \cup T^*$ gerade definiert, vgl. z. B. [Bau02, § 9].

Für $n = 0$ sind $A(0)$ und $B(0)$ trivial (da pfx_0 immer das leere Wort liefert). Es gelte also $A(n)$ und $B(n)$ für ein $n \in \mathbb{N}_0$, und wir zeigen nun $A(n+1)$ und $B(n+1)$.

Es sei $\alpha \in T^*$ mit $|\alpha| \leq n+1$ ein Präfix, für den gilt

$$\gamma := P\big(\text{pfx}_{n+1}(R(\mathcal{Z}^*)) = \alpha\big) > 0, \tag{3.3}$$

der also mit positiver Wahrscheinlichkeit im realen Modell mit $\mathcal{Z}^*$ vorkommen kann. Im folgenden schreiben wir $\tilde{\alpha}$ für das um eine Komponente gekürzte α, also $\tilde{\alpha} := \text{pfx}_n(\alpha)$.

Um $A(n+1)$ zu beweisen, genügt es zu zeigen, daß

$$P\big(\text{pfx}_{n+1}(I(\mathcal{Z}^*)) = \alpha\big) = \gamma, \tag{3.4}$$

also daß das Präfix α im idealen Modell mit $\mathcal{Z}^*$ genauso wahrscheinlich ist. (Die Gleichheit für Präfixe mit Wahrscheinlichkeit 0 ist mit abgedeckt, da die Summe der Wahrscheinlichkeiten über alle Präfixe in beiden Fällen 1 sein muß.)

Dazu führen wir zunächst folgende Konvention ein: Ein Tupel $t = (\mathit{name}, s, p, m, s', p', m') \in T$, das eine Aktivierung der Umgebung beschreibt, besteht aus dem Namen *name* (welcher immer `env` ist), dem Zustand s vor der Aktivierung, dem eingehenden Port p, über den die Nachricht m empfangen wird, dem Zustand s' nach der Aktivierung, und der Nachricht m' die über den ausgehenden Port p' geschickt wird. Wir bezeichnen nun das Tupel (name, s, p, m) als den *passiven Teil* $\mathit{passive}(t)$ von t und (s', p', m') als den *aktiven Teil* von t. Die dahinterstehende Intuition ist, daß der passive Teil bei einer Aktivierung nicht vom Verhalten der Umgebung in dieser Aktivierung abhängt (höchstens von vorangegangenen Aktivierungen), während der aktive Teil durch die Zustandsübergangsfunktion der Umgebung aus dem passiven Teil berechnet wird.

Aus (3.2) folgt nun, daß für $p := \mathit{passive}(\alpha_{n+1})$ gilt:

$$\begin{aligned} &P\big(\mathit{passive}(R_{n+1}(\mathcal{Z}_?)) = p \;\big|\; \text{pfx}_n(R(\mathcal{Z}_?)) = \tilde{\alpha}\big) \\ &= P\big(\mathit{passive}(I_{n+1}(\mathcal{Z}_?)) = p \;\big|\; \text{pfx}_n(I(\mathcal{Z}_?)) = \tilde{\alpha}\big), \end{aligned} \tag{3.5}$$

wobei $R_i(\mathcal{Z}_?)$ das i-te Tupel in der Sicht $R(\mathcal{Z}_?)$ bezeichnet und $I_i(\mathcal{Z}_?)$ analog. (Und für $|\alpha| < n+1$ setzen wir $\mathit{passive}(a_{n+1}) := \bot$.) Wir verwenden dabei noch (3.3) und $B(n)$, um zu sehen, daß die Wahrscheinlichkeit für die Bedingungen nicht verschwinden. Intuitiv heißt dies, daß die bedingte Wahrscheinlichkeit, daß die zufällige Umgebung $\mathcal{Z}_?$ sich bei der $(n+1)$-ten Aktivierung in einer Situation p findet, für gegebene Vorgeschichte α, im realen und im idealen Modell gleich ist.

Weiterhin gilt:

$$\begin{aligned} &P\big(\mathit{passive}(R_{n+1}(\mathcal{Z}_?)) = p \;\big|\; \text{pfx}_n(R(\mathcal{Z}_?)) = \tilde{\alpha}\big) \\ &= P\big(\mathit{passive}(R_{n+1}(\mathcal{Z}^*)) = p \;\big|\; \text{pfx}_n(R(\mathcal{Z}^*)) = \tilde{\alpha}\big) > 0. \end{aligned} \tag{3.6}$$

Intuitiv bedeutet dies, daß (gegeben eine feste Vorgeschichte α) die Wahrscheinlichkeit, daß die zufällige Umgebung $\mathcal{Z}_?$ sich im realen Modell in der $(n+1)$-ten Aktivierung in einer Situation p befindet, die gleiche ist wie die Wahrscheinlichkeit (gegeben die gleiche Vorgeschichte α), daß daß die ursprüngliche Umgebung $\mathcal{Z}_?$ sich im realen Modell in der $(n+1)$-ten Aktivierung in derselben Situation p befindet Dies liegt daran, daß die Verteilung des *passiven* Teils von $R_{n+1}(\mathcal{Z}^*)$ nicht von der Zustandsübergangsfunktion der Umgebung abhängt, wenn die Ein- und Ausgaben und Zustände der Umgebung in allen vorangegangenen Aktivierungen festgelegt sind. (Sehr wohl hängt diese Verteilung vom restlichen Protokoll und von den Ports der Umgebung ab, aber diese sind in beiden Fällen gleich). Die Bedingungen haben positive Wahrscheinlichkeit nach (3.3) und $B(n)$. Daß die rechte Seite der Gleichung positiv ist, folgt aus (3.3), wenn man beachtet, daß $\mathrm{pfx}_n(R(\mathcal{Z}^*)) = \alpha$ nach Definition von p auch $\mathit{passive}(R_{n+1}(\mathcal{Z}^*)) = p$ impliziert.

Ebenso erhalten wir die zu (3.6) analoge Gleichung im idealen Modell (wobei wir $R(\mathcal{Z}_?) = I(\mathcal{Z}_?)$ und $A(n)$ verwenden, um $B(n)$ auf $I(\mathcal{Z}^*)$ und $I(\mathcal{Z}_?)$ zu übertragen):

$$\begin{aligned} &P\big(\mathit{passive}(I_{n+1}(\mathcal{Z}_?)) = p \;\big|\; \mathrm{pfx}_n(I(\mathcal{Z}_?)) = \tilde{\alpha}\big) \\ &= P\big(\mathit{passive}(I_{n+1}(\mathcal{Z}^*)) = p \;\big|\; \mathrm{pfx}_n(I(\mathcal{Z}^*)) = \tilde{\alpha}\big). \end{aligned} \tag{3.7}$$

Aus (3.5–3.7) folgt die zu (3.5) analoge Gleichung für die ursprüngliche Umgebung $\mathcal{Z}^*$:

$$\begin{aligned} &P\big(\mathit{passive}(R_{n+1}(\mathcal{Z}^*)) = p \;\big|\; \mathrm{pfx}_n(R(\mathcal{Z}^*)) = \tilde{\alpha}\big) \\ &= P\big(\mathit{passive}(I_{n+1}(\mathcal{Z}^*)) = p \;\big|\; \mathrm{pfx}_n(I(\mathcal{Z}^*)) = \tilde{\alpha}\big), \end{aligned} \tag{3.8}$$

Weil aber der aktive Teil einer Aktivierung bei gegebenem passivem Teil nur von der Zustandübergangsfunktion der Umgebung abhängt (und weder vom restlichen Protokoll noch von den vorangegangenen Aktivierungen), folgt daraus

$$\begin{aligned} &P\big(R_{n+1}(\mathcal{Z}^*) = \alpha_{n+1} \;\big|\; \mathrm{pfx}_n(R(\mathcal{Z}^*)) = \tilde{\alpha}\big) \\ &= P\big(I_{n+1}(\mathcal{Z}^*) = \alpha_{n+1} \;\big|\; \mathrm{pfx}_n(I(\mathcal{Z}^*)) = \tilde{\alpha}\big), \end{aligned} \tag{3.9}$$

und durch Multiplikation mit der aus $A(n)$ folgenden Gleichung

$$P\big(\mathrm{pfx}_n(R(\mathcal{Z}^*)) = \tilde{\alpha}\big) \;=\; P\big(\mathrm{pfx}_n(I(\mathcal{Z}^*)) = \tilde{\alpha}\big)$$

erhalten wir

$$P\big(\mathrm{pfx}_{n+1}(R(\mathcal{Z}^*)) = \alpha\big) \;=\; P\big(\mathrm{pfx}_{n+1}(I(\mathcal{Z}^*)) = \alpha\big),$$

woraus (3.4) folgt. Damit ist $A(n+1)$ gezeigt.

Nun müssen wir noch $B(n+1)$ einsehen. Nach Konstruktion der zufälligen Umgebung $\mathcal{Z}_?$ (vgl. Definition 3.1) hat in jeder Aktivierung von $\mathcal{Z}_?$ jeder aktive Teil eine positive Wahrscheinlichkeit. Insbesondere ist also

$$P\big(R_{n+1}(\mathcal{Z}_?) = \alpha_{n+1} \;\big|\; \mathit{passive}(R_{n+1}(\mathcal{Z}_?)) = p \text{ und } \mathrm{pfx}_n(R(\mathcal{Z}_?)) = \tilde{\alpha}\big) > 0.$$

Multiplizieren wir diese Ungleichung mit (3.6) und mit der aus (3.3) und $B(n)$ folgenden Ungleichung $P\big(\mathrm{pfx}_n(R(\mathcal{Z}_?)) = \tilde{\alpha}\big) > 0$, so erhalten wir

$$P\big(\mathrm{pfx}_{n+1}(R(\mathcal{Z}_?)) = \alpha\big) > 0.$$

Da dies für alle α gilt, die (3.3) erfüllen, folgt $B(n+1)$.

Damit ist die Induktion abgeschlossen, und (3.2) gilt für alle n. Wie bei (3.2) bemerkt, folgt daraus das Lemma. □

Aus Lemma 3.2 können wir nun sehr einfach die Äquivalenz der verschiedenen perfekten Sicherheitsbegriffe bzgl. der Sicht der Umgebung folgern:

Korollar 3.3 (Äquivalenz perfekter Sicherheitsbegriffe bzgl. der Sicht)

Es seien π und ρ Protokolle. Dann sind die folgenden Aussagen äquivalent:

- π ist so sicher wie ρ bezüglich perfekter *allgemeiner* Sicherheit *mit Auxiliary input* bzgl. der Sicht der Umgebung.
- π ist so sicher wie ρ bezüglich perfekter *spezieller* Sicherheit *mit Auxiliary input* bzgl. der Sicht der Umgebung.
- π ist so sicher wie ρ bezüglich perfekter *allgemeiner* Sicherheit *ohne Auxiliary input* bzgl. der Sicht der Umgebung.
- π ist so sicher wie ρ bezüglich perfekter *spezieller* Sicherheit *ohne Auxiliary input* bzgl. der Sicht der Umgebung.

Beweis: Im folgenden bezeichne Sicherheit immer perfekte Sicherheit bzgl. der Sicht der Umgebung.

Da die Varianten der allgemeinen Sicherheit die entsprechenden Varianten der speziellen Sicherheit implizieren (Lemma A.4), und allgemeine Sicherheit mit und ohne Auxiliary input nach Lemma A.2 äquivalent sind, genügt es folgendes zu zeigen:

(i) Spezielle Sicherheit *mit* Auxiliary input impliziert allgemeine Sicherheit *mit* Auxiliary input.

(ii) Spezielle Sicherheit *ohne* Auxiliary input impliziert allgemeine Sicherheit *ohne* Auxiliary input.

Wir wollen nun (i) zeigen. Angenommen, π sei *nicht* so sicher wie ρ bezüglich *allgemeiner* Sicherheit mit Auxiliary input. Dann existieren ein zulässiger Angreifer $\mathcal{A}$ und zu jedem zulässigen Simulator $\mathcal{S}$ eine zulässige Umgebung $\mathcal{Z}_\mathcal{S}$, ein Sicherheitsparameter $k_\mathcal{S} \in \mathbb{N}$ und ein Auxiliary input $z_\mathcal{S} \in \Sigma^*$, so daß:

$$\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}_\mathcal{S}}(k_\mathcal{S}, z_\mathcal{S}) \neq \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}_\mathcal{S}}(k_\mathcal{S}, z_\mathcal{Z}),$$

sprich die (vom Simulator $\mathcal{S}$ abhängige) Umgebung $\mathcal{Z}_\mathcal{S}$ unterscheidet.

Es sei nun $\mathcal{Z}_?$ die zu π, ρ und $\mathcal{A}$ passende zufällige Umgebung. Dann ist nach Lemma 3.2

$$\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}_?}(k_\mathcal{S}, z_\mathcal{S}) \neq \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}_?}(k_\mathcal{S}, z_\mathcal{Z}),$$

sprich die (vom Simulator unabhängige) zufällige Umgebung $\mathcal{Z}_?$ unterscheidet. Damit ist nach Definition 2.16 auch bezüglich spezieller Sicherheit mit Auxiliary input π *nicht* so sicher wie ρ. Es folgt die Implikation (i).

Der Beweis der Implikation (ii) ist wörtlich gleich, mit dem einzigen Unterschied, daß der Auxiliary input $z_\mathcal{S} \in \{\lambda\}$ statt $z_\mathcal{S} \in \{\Sigma^*\}$ gewählt wird. □

3.2. Sicht und Ausgabe

Im vorangegangenen Abschnitt haben wir gezeigt, daß die verschiedenen Varianten der perfekten Sicherheit bzgl. der *Sicht* der Umgebung äquivalent sind. Es stellt sich die Frage, ob dies auch für die Varianten bzgl. der *Ausgabe* der Umgebung gilt. Diese Frage werden wir im folgenden positiv beantworten, indem wir zeigen, daß Sicherheit bzgl. der Ausgabe und bzgl. der Sicht der Umgebung äquivalent sind.

Es ist eine verbreitete Ansicht, daß diese Äquivalenz trivial folgt, wenn man Umgebungen betrachtet, die ihre gesamte Sicht ausgeben. Um zu zeigen, daß dies im allgemeinen nicht korrekt ist und zu demonstrieren, worin das Problem besteht, geben wir zunächst in Abschnitt 3.2.1 ein Beispiel an, das im *statistischen* Fall die beiden Begriffe trennt. Dann zeigen wir in Abschnitt 3.2.2, wie das Problem im *perfekten* Fall umgangen und die Äquivalenz doch gezeigt werden kann. (Und zum komplexitätstheoretischen Fall mit *polynomiell-beschränkten Protokollen* siehe Lemma A.10.)

3.2.1. Der statistische Fall

Der naive Ansatz, um die Äquivalenz der Sicherheit bzgl. der Sicht und bzgl. der Ausgabe der Umgebung zu zeigen, ist der, daß man eine bzgl. der Sicht unterscheidende Umgebung $\mathcal{Z}$ in eine Umgebung $\mathcal{Z}'$ umwandelt, die ihre Sicht ausgibt. Bei genauerer Betrachtung stellen sich aber zwei Fragen:

- Wenn $\mathcal{Z}$ nie terminiert, sondern bei jeder Aktivierung wieder eine neue Nachricht an das Protokoll (oder den Angreifer) schickt, dann können wir die Umgebung $\mathcal{Z}'$ nicht so definieren, daß sie erst dann Ausgabe liefert, wenn $\mathcal{Z}$ terminiert. Es muß also $\mathcal{Z}'$ nach einer gewissen Anzahl von Aktivierungen abbrechen, selbst wenn $\mathcal{Z}$ dies nicht tut. Im Falle der allgemeinen Sicherheit kann man leicht einsehen, daß es eine Zahl von Aktivierungen gibt (abhängig vom Sicherheitsparameter), so daß der statistische Abstand zwischen realer und idealer Sicht nach dieser Anzahl von Aktivierungen beliebig nahe an den statistischen Abstand der vollständigen Sicht herankommt (Lemma 1.3 (ix)). Läßt man $\mathcal{Z}'$ nach dieser Anzahl von Aktivierungen terminieren, so liefert auch $\mathcal{Z}'$ unterscheidbare Sichten.

  Im Falle der speziellen Sicherheit ohne Auxiliary input jedoch ist die Situation nicht so einfach. So kann die Anzahl der Aktivierungen, die notwendig ist, um eine Unterscheidung zu ermöglichen, vom Simulator abhängen. Da dieser erst *nach* der Umgebung gewählt wird, können wir nicht schon bei der Konstruktion der Umgebung diese Schranke einbauen.

- Ein weiteres Problem ist die Tatsache, daß es passieren kann, daß die Umgebung $\mathcal{Z}$ ab einem bestimmten Zeitpunkt nicht mehr aktiviert wird, weil z. B. das Protokoll oder der Angreifer bzw. Simulator in einer Endlosschleife verharren. In diesem Fall wird $\mathcal{Z}$ keine Ausgabe liefern können. Wir dürfen also die Anzahl der Aktivierungen, die $\mathcal{Z}$ durchläuft, bevor es eine Ausgabe liefert, auch nicht zu hoch ansetzen. Diese Problematik besteht in allen Varianten der statistischen Sicherheit (also nicht nur bei der speziellen ohne Auxiliary input). Im Beweis des folgenden Lemmas geben wir ein Paar von Protokollen an, welches diese Tatsache ausnutzt, um Sicherheit bzgl. der Sicht und der Ausgabe der Umgebung zu trennen.

Lemma 3.4 (Trennung der statistischen Sicherheit bzgl. der Sicht und der Ausgabe der Umgebung)
Es existieren Protokolle π und ρ, für die folgendes gilt:

- π ist so sicher wie ρ bezüglich statistischer spezieller und allgemeiner Sicherheit mit und ohne Auxiliary input bzgl. der *Ausgabe* der Umgebung.
- π ist nicht so sicher wie ρ bezüglich statistischer spezieller oder allgemeiner Sicherheit mit oder ohne Auxiliary input bzgl. der *Sicht* der Umgebung.

Beweis: Sicherheit steht im folgenden für statistische spezielle/allgemeine Sicherheit mit/ohne Auxiliary input.

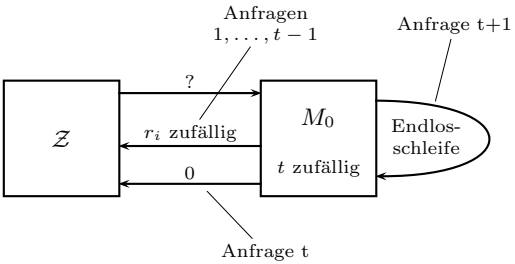


Abbildung 3.1.: Schematische Darstellung des Protokolls π bestehend aus der Maschine M_0. Dargestellt ist zusätzlich die Umgebung $\mathcal{Z}$.

Die Maschine M_0 wählt ein zufälliges $t \in \{1, \ldots, 2^k\}$. Bei jeder Anfrage der Umgebung (symbolisiert durch ein ?) geschieht folgendes: Handelt es sich um die erste bis $(t-1)$-te Anfrage, so wird mit einem Zufallsbit r_i geantwortet. Bei der t-ten Anfrage sendet M_0 ihre Identität (also 0). Bei der $(t+1)$-ten Anfrage geht M_0 in eine Endlosschleife über.

Das Protokoll ρ unterscheidet sich von dem hier dargestellten nur darin, daß es aus einer Maschine M_1 besteht, welche bei der t-ten Anfrage *ihre* Identität, also 1 statt 0, überträgt.

Wir geben zunächst die Protokolle π und ρ an und zeigen dann, daß diese tatsächlich ein trennendes Beispiel darstellen.

Das Protokoll π besteht aus einer Maschine M_0, die sich wie folgt verhält: M_0 wählt zunächst ein gleichverteilt zufälliges $t \in \{1, \ldots, 2^k\}$ (wobei k den Sicherheitsparameter darstelle). Bei der i-ten Aktivierung durch die Umgebung $\mathcal{Z}$ unterscheiden wir nun drei Fälle:

- Ist $i < t$, so sendet M_0 ein Zufallsbit r_i an die Umgebung.
- Ist $i = t$, so sendet M_0 das Bit $r_t := 0$ an die Umgebung.
- Ist $i > t$, so geht M_0 in eine Endlosschleife über (d. h. M_0 beginnt sich über eine Verbindung zu sich selbst Nachrichten zu senden, und hört damit nie auf).

M_0 hat keine Verbindungen zum Angreifer/Simulator. Das Protokoll π ist in Abbildung 3.1 skizziert.

Das Protokoll ρ hingegen besteht aus einer Maschine M_1, welche sich wie M_0 verhält, außer daß im Falle $i = t$ das Bit $r_t := 1$ an die Umgebung übermittelt wird.

Intuitiv wählen also π und ρ eine zufällige große Zahl t, und verraten der Umgebung bei der t-ten Anfrage ihre Identität. Vor der t-ten Anfrage werden nur zufällige Antworten geliefert (die aber von der richtigen Antwort nicht zu

unterscheiden sind). Fragt die Umgebung jedoch mehr als t-mal, so wird sie nie wieder aktiviert. Zu beachten ist, daß t der Umgebung nicht mitgeteilt wird.

Es ist sehr einfach zu erkennen, daß π nicht so sicher wie ρ bezüglich Sicherheit bzgl. der Sicht der Umgebung ist. Denn die Umgebung $\mathcal{Z}$, welche bei jeder Aktivierung eine Nachricht an π bzw. ρ schickt, wird zunächst $t-1$ zufällige Bits r_i erhalten, dann ein Bit r_t, welches kundtut, ob $\mathcal{Z}$ mit π oder mit ρ kommuniziert, und danach wird $\mathcal{Z}$ nicht wieder aktiviert werden. Die Sicht von $\mathcal{Z}$ enthält also eine abbrechende Folge von Bits, von denen im realen Model (mit π) das letzte Bit immer eine 0 ist, im idealen Modell (mit ρ) aber immer eine 1. Somit ist der statistische Abstand zwischen der Sicht im realen und der Sicht im idealen Modell 1, es ist also π nicht so sicher wie ρ bzgl. der Sicht der Umgebung.

Interessanter ist der Fall der Sicherheit bzgl. der Ausgabe der Umgebung. Wir skizzieren zunächst intuitiv, warum wir erwarten, daß hier π so sicher wie ρ ist, und geben dann einen Beweis für diese Behauptung. Wir unterscheiden intuitiv drei Fälle: Im ersten Fall stellt $\mathcal{Z}$ weniger als t Anfragen an das Protokoll und erhält nur Zufallsbits, die unabhängig von der Identität des Protokolls sind. In diesem Fall kann $\mathcal{Z}$ also nichts darüber lernen, ob es sich im realen oder im idealen Modell befindet. Im zweiten Fall stellt $\mathcal{Z}$ mehr als t Anfragen. Dann wird das Protokoll in eine Endlosschleife übergehen, und die von $\mathcal{Z}$ bisher erhaltenen Informationen gehen verloren, weil $\mathcal{Z}$ nicht mehr aktiviert wird, um diese auszugeben. Im dritten Fall stellt $\mathcal{Z}$ genau t Anfragen. In diesem Fall ist die letzte Antwort, die $\mathcal{Z}$ erhält, die Identität des Protokolls und kann zur Unterscheidung herangezogen werden. Wir erwarten jedoch, daß dieser Fall sehr selten auftritt (in der Größenordnung von 2^{-k}), da t nicht bekannt ist und von $\mathcal{Z}$ nur geraten werden kann (hier ist zu beachten, daß die Antwort auf die t-te Anfrage wie die vorangegangenen Zufallsbits aussieht). Also kann auch dieser Fall zur Unterscheidung der Protokolle nur vernachlässigbar viel beitragen. Da dies für alle Umgebungen $\mathcal{Z}$ gilt (unabhängig vom Angreifer und Simulator), ist π so sicher wie ρ bzgl. der Ausgabe der Umgebung.

Daß diese Intuition korrekt ist, wollen wir nun zeigen: Es sei also $\mathcal{A}$ ein zulässiger Angreifer. Als Simulator wählen wir $\mathcal{S} := \mathcal{A}$. Es genügt nun zu zeigen, daß für jede Umgebung $\mathcal{Z}$, alle Sicherheitsparameter $k \in \mathbb{N}$ und alle Auxiliary inputs $z \in \Sigma^*$ gilt:

$$\Delta\big(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\big) \leq 2^{-k+2} \tag{3.10}$$

(Daß in dieser Formel zweimal $\mathcal{A}$ auftaucht, ist kein Versehen, man beachte $\mathcal{S} = \mathcal{A}$.) Wir wollen also einsehen, daß der statistische Abstand zwischen der Ausgabe der Umgebung im realen und im idealen Modell durch 2^{-k+2} beschränkt, insbesondere also vernachlässigbar ist.

Um dies zu zeigen, führen wir zunächst zwei weitere Maschinen M_* und M_∞ ein. M_* unterscheidet sich von M_0 und M_1 dadurch, daß es auch bei der t-

ten Anfrage ein Zufallsbit zurückgibt (und nicht seine Identität). Es ist also hier auch r_t zufällig. Die Maschine M_∞ unterscheidet sich von M_* dadurch, daß zwar ein zufälliges t gewählt wird, aber auf jede Anfrage der Umgebung mit einem Zufallsbit r_i geantwortet wird (insbesondere wird t nicht verwendet, nie etwas anderes als Zufallsbits an $\mathcal{Z}$ geschickt, und M_∞ geht nie in eine Endlosschleife über).

Es bezeichne t_x für $x \in \{0, 1, *, \infty\}$ den Wert t, welchen M_x in einer Ausführung des Netzwerks $\{M_x, \mathcal{A}, \mathcal{Z}\}$ wählt. Insbesondere sind also t_0 und t_1 die Wahl von M_0 bzw. M_1 im realen bzw. idealen Modell. Weiterhin bezeichne n_x die Anzahl der Anfragen, die $\mathcal{Z}$ an M_x stellt.

Nun ist, da M_∞ den Wert t_∞ nur wählt, aber nie verwendet, n_∞ unabhängig von t_∞. Der Wert t_∞ ist weiterhin gleichverteilt auf $\{1, \ldots, 2^k\}$, also ist $P(t_\infty = n_\infty) \leq 2^{-k}$.

Da die Maschine M_* bis einschließlich der t-ten Anfrage das gleiche Programm wie M_∞ hat, folgt, daß die Wahrscheinlichkeit, daß $\mathcal{Z}$ genau t Anfragen stellt, für M_* und M_∞ gleich ist, also $P(t_* = n_*) = P(t_\infty = n_\infty) \leq 2^{-k}$.

Weiterhin ist die Maschine M_* so konstruiert, daß sie sich mit Wahrscheinlichkeit $\frac{1}{2}$ so verhält wie M_0 und mit Wahrscheinlichkeit $\frac{1}{2}$ wie M_1. Es folgt daraus

$$\tfrac{1}{2}\big(P(t_0 = n_0) + P(t_1 = n_1)\big) = P(t_* = n_*) \leq 2^{-k},$$

woraus wiederum folgt, daß

$$P(t_i = n_i) \leq 2^{-k+1} \text{ für } i = 0, 1. \tag{3.11}$$

Wir wissen nun also, daß die Wahrscheinlichkeit, daß die Umgebung genau t Anfragen stellt (und damit die Identität des Protokolls erfährt und ausgeben kann) durch 2^{-k+1} beschränkt ist.

Dies ermöglicht es uns, den statistischen Abstand zwischen der realen und der idealen Ausgabe von $\mathcal{Z}$ wie in (3.10) gefordert abzuschätzen. Dazu sei abkürzend $R := \text{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ die Ausgabe im realen (mit dem Protokoll π) und $I := \text{OUTPUT}_{\rho,\mathcal{A},\mathcal{Z}}(k, z)$ die im idealen Modell (mit dem Protokoll ρ). Dann

ist nach Lemma 1.3 (iii) (es sei dabei $O := \Sigma^* \cup \{\perp\}$):

$$\begin{aligned}
2\Delta(R;I) &= \sum_{o\in O} \big|P(R=o) - P(I=o)\big| \\
&= \sum_{o\in O} \Big|\big(P(R=o, n_0 < t_0) + P(R=o, n_0 = t_0) + P(R=o, n_0 > t_0)\big) \\
&\qquad - \big(P(I=o, n_1 < t_1) + P(I=o, n_1 = t_1) + P(I=o, n_1 > t_1)\big)\Big| \\
&\le \sum_{o\in O} \big|P(R=o, n_0 < t_0) - P(I=o, n_1 < t_1)\big| \\
&\qquad + \sum_{o\in O} \big|P(R=o, n_0 = t_0) - P(I=o, n_1 = t_1)\big| \\
&\qquad + \sum_{o\in O} \big|P(R=o, n_0 > t_0) - P(I=o, n_1 > t_1)\big| \qquad (3.12)
\end{aligned}$$

Es bleibt, die drei Summen auf der rechten Seite dieser Ungleichung abzuschätzen.

Es ist $P(R = o, n_0 < t_0)$ die Wahrscheinlichkeit, daß die Umgebung $\mathcal{Z}$ in einem Lauf von π, $\mathcal{A}$ und $\mathcal{Z}$ weniger als t Anfragen stellt und die Ausgabe o hat (wobei o auch $\perp$ sein kann, wenn $\mathcal{Z}$ keine Ausgabe liefert). Entsprechend ist $P(I = o, n_1 < t_1)$ die entsprechende Wahrscheinlichkeit, wenn man π durch ρ ersetzt. Da π und ρ (bzw. die Maschinen M_0 und M_1 aus denen sie bestehen) sich bis einschließlich der $(t-1)$-ten Anfrage identisch verhalten, müssen diese beiden Wahrscheinlichkeiten gleich sein, also ist

$$\sum_{o\in O} \big|P(R=o, n_0 < t_0) - P(I=o, n_1 < t_1)\big| = 0.$$

Die zweite Summe in (3.12) läßt sich wie folgt abschätzen:

$$\begin{aligned}
&\sum_{o\in O} \big|P(R=o, n_0 = t_0) - P(I=o, n_1 = t_1)\big| \\
&\quad \le \sum_{o\in O} P(R=o, n_0 = t_0) + \sum_{o\in O} P(I=o, n_1 = t_1) \\
&\quad = P(n_0 = t_0) + P(n_1 = t_1) \overset{(3.11)}{\le} 2^{-k+2}.
\end{aligned}$$

Wir betrachten nun die dritte Summe in (3.12). Wenn die Umgebung $\mathcal{Z}$ mehr als t Anfragen an die Maschine M_0 oder M_1 stellt, so geht letztere nach Konstruktion in eine Endlosschleife über, und $\mathcal{Z}$ kann keine Ausgabe generieren. Somit ist $P(R \neq \perp, n_0 > t_0) = P(I \neq \perp, n_1 > t_1) = 0$. Weiterhin ist, da sich M_0 und

M_1 bis zur $(t-1)$-ten Anfrage identisch verhalten, $P(n_0 < t_0) = P(n_1 < t_1)$. Damit rechnen wir

$$\begin{aligned}
&\sum_{o \in O} \big| P(R = o, n_0 > t_0) - P(I = o, n_1 > t_1) \big| \\
&\quad = \big| P(R = \perp, n_0 > t_0) - P(I = \perp, n_1 > t_1) \big| \\
&\quad = \big| P(n_0 > t_0) - P(n_1 > t_1) \big| \\
&\quad = \big| 1 - P(n_0 = t_0) - P(n_0 < t_0) - 1 + P(n_1 = t_1) + P(n_1 < t_1) \big| \\
&\quad = \big| P(n_0 = t_0) - P(n_1 = t_1) \big| \\
&\quad \overset{(3.11)}{\leq} 2^{-k+2}.
\end{aligned}$$

Setzen wir diese drei Abschätzungen in (3.12) ein, so erhalten wir

$$2\Delta(R; I) \leq 2^{-k+3},$$

und das ist gerade (3.10). □

3.2.2. Der perfekte Fall

Wie wir im vorangegangenen Abschnitt am Beispiel der statistischen Sicherheit gesehen haben, ist es im allgemeinen nichttrivial oder sogar unmöglich zu zeigen, daß Sicherheit bezüglich der Sicht und der Umgebung äquivalent sind. Erfreulicherweise ist es jedoch im Falle der perfekten Sicherheit möglich, die Äquivalenz dieser beiden Varianten der Sicherheit zu zeigen, und zwar unabhängig davon, ob spezielle oder allgemeine Sicherheit vorliegt.

Die hierzu verwendete Beweismethode ähnelt der aus Abschnitt 3.1. Aber anstatt eine Umgebung zu betrachten, die völlig zufällig handelt, transformieren wir eine Umgebung $\mathcal{Z}$, die bzgl. der Sicht unterscheidet, in eine Umgebung $\mathcal{Z}_{out}$, die bzgl. Ausgabe unterscheidet, wobei $\mathcal{Z}_{out}$ nach einer zufälligen Anzahl von Aktivierungen ihre gesamte bisherige Sicht ausgibt. In der Situation von Lemma 3.4 hätte dies nicht geholfen, da die Wahrscheinlichkeit, zufällig die „richtige“ Aktivierung zu treffen, zu klein sein kann (genau dies haben wir im Beweis von Lemma 3.4 ausgenutzt). Im perfekten Fall aber ist jede noch so kleine Wahrscheinlichkeit relevant, und daher wird die transformierte Umgebung zwar mit einer kleineren, aber dennoch nicht mit Wahrscheinlichkeit 0 unterscheiden.

Wir definieren nun, wie wir die Umgebung $\mathcal{Z}_{out}$ konstruieren.

Definition 3.5 (Zufällig ausgebende Umgebung)

(Fortsetzung nächste Seite)

(Fortsetzung)

Es sei $\mathcal{Z}$ eine Maschine ohne `output`-Port. Dann ist die *zufällig ausgebende Umgebung* $\mathcal{Z}_{out}$ *zu* $\mathcal{Z}$ wie folgt definiert:

- $\mathcal{Z}_{out}$ hat die gleichen eingehenden Ports wie $\mathcal{Z}$ (also $in(\mathcal{Z}_{out}) := in(\mathcal{Z})$).
- $\mathcal{Z}_{out}$ hat die gleichen ausgehenden Ports wie $\mathcal{Z}$ und zusätzlich einen ausgehenden Port `output` (also $in(\mathcal{Z}_{out}) := in(\mathcal{Z}) \cup \{\texttt{output}\}$).

Die Maschine $\mathcal{Z}$ hat folgendes Programm:

1. Initial sei $s := \lambda$ (der Zustand der simulierten Maschine $\mathcal{Z}$) und *view* eine leere Liste (die bisherige Sicht).
2. Wähle ein zufälliges $t \in \mathbb{N}$, so daß $t = i$ mit Wahrscheinlichkeit 2^{-i}.
3. Wenn eine Nachricht m über den Port p empfangen wird, simuliere eine Aktivierung von $\mathcal{Z}$ mit Zustand s. Dann geht $\mathcal{Z}$ in einen Zustand s' über und gibt eine Nachricht m' über den Port p' aus (mit $p' = \lambda$, falls keine Nachricht gesandt wird).
4. Füge der Liste *view* das Tupel $(name_{\mathcal{Z}}, s, p, m, s', p', m')$ an (also genau die zu dieser Aktivierung in der Sicht von $\mathcal{Z}$ verzeichneten Informationen).
5. Setze $s := s'$ (speichere den Zustand von $\mathcal{Z}$).
6. Wenn *view* weniger als t Tupel enthält, gib die Nachricht m' über den Port p' aus (verhalte dich also wie $\mathcal{Z}$).
7. Ansonsten (also in der t-ten Aktivierung) sende *view* über den Port `output` (gibt die Sicht der ersten t Aktivierungen des simulierten $\mathcal{Z}$ aus). Wir gehen hierbei von einer beliebigen, aber festen Kodierung von *view* als Element von Σ^* aus.
8. Gehe wieder zu Schritt 3.

Man beachte, daß im Gegensatz zur Situation in Definition 3.1 hier die zufällig ausgebende Umgebung $\mathcal{Z}_{out}$ in Abhängigkeit von $\mathcal{Z}$ gewählt wird.

Das folgende Lemma sagt uns nun, daß im Falle der perfekten Sicherheit die Umgebung $\mathcal{Z}_{out}$ in all den Situationen bzgl. der *Ausgabe* unterscheidet, in denen $\mathcal{Z}$ bzgl. der *Sicht* unterscheidet:

Lemma 3.6 (Umgebung $\mathcal{Z}_{out}$ unterscheidet)
Es seien π und ρ Protokolle, $\mathcal{A}$ ein zulässiger Angreifer, $\mathcal{S}$ ein zulässiger Simulator und $\mathcal{Z}$ eine zulässige Umgebung. Weiterhin seien der Sicherheitsparameter $k \in \mathbb{N}$ und der Auxiliary input $z \in \Sigma^*$ gegeben. Es bezeichne

(Fortsetzung nächste Seite)

(Fortsetzung)

$\mathcal{Z}_{out}$ die zufällig ausgebende Umgebung zu $\mathcal{Z}$.

Ist

$$\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) \neq \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}}(k,z),$$

so ist auch

$$\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}_{out}}(k,z) \neq \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}_{out}}(k,z).$$

In anderen Worten, wenn die *Sicht* der ursprünglichen Umgebung $\mathcal{Z}$ im realen und idealen Modell verschieden ist, so ist es auch die *Ausgabe* der zufällig ausgebenden Umgebung $\mathcal{Z}_{out}$.

Beweis: Wir führen die folgenden Abkürzungen ein: Es sei $R(\mathcal{Z}) := \mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ (die *Sicht* von $\mathcal{Z}$ im realen Modell), $I(\mathcal{Z}) := \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)$ (die *Sicht* von $\mathcal{Z}$ im idealen Modell). Weiter sei $RO(\mathcal{Z}_{out}) := \mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}_{out}}$ die *Ausgabe* von $\mathcal{Z}_{out}$ im realen Modell und $IO(\mathcal{Z}_{out}) := \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}_{out}}$ die *Ausgabe* von $\mathcal{Z}_{out}$ im idealen Modell.

Es sei wie im Beweis von Lemma 3.2 $T^{\mathbb{N}} \cup T^*$ die Menge aller möglichen Sichten einer Umgebung (dabei sind die Sichten, bei der die Umgebung nur endlich oft aktiviert wird, gerade die Elemente von T^*).

In dieser Notation ist die Prämisse des Lemmas $R(\mathcal{Z}) \neq I(\mathcal{Z})$, und zu zeigen ist $RO(\mathcal{Z}_{out}) \neq IO(\mathcal{Z}_{out})$.

Da $R(\mathcal{Z}) \neq I(\mathcal{Z})$, existiert ein $n \in \mathbb{N}$ und ein Präfix $\beta \in T^*$, so daß

$$P\big(\mathrm{pfx}_n\big(R(\mathcal{Z})\big) = \beta\big) \neq P\big(\mathrm{pfx}_n\big(I(\mathcal{Z})\big) = \beta\big). \tag{3.13}$$

(Dabei ist pfx_n wie im Beweis von Lemma 3.2 definiert.) Insbesondere gibt es n und β, so daß n minimal ist unter allen die Ungleichung erfüllenden n. Im folgenden seien n und β solch ein minimales Paar.

Wir unterscheiden nun drei Fälle. Es kann $|\beta| = n$ sein (d. h. die Umgebung wird mindestens n-mal aktiviert), $|\beta| = n-1$ (d. h. die Umgebung wird genau $(n-1)$-mal aktiviert und danach geht das Protokoll oder der Angreifer in eine Endlosschleife), und $|\beta| \leq n-2$ (die Umgebung wird weniger als $(n-1)$-mal aktiviert).

Betrachten wir zunächst den Fall $|\beta| \leq n-2$. Eine Sicht v mit $\mathrm{pfx}_{n-1}(v) = \beta$ muß dann Länge $|v| \leq n-2$ haben, und somit ist auch $\mathrm{pfx}_n(v) = \mathrm{pfx}_{n-1}(v) = \beta$. Umgekehrt folgt aus $\mathrm{pfx}_n(v) = \beta$ direkt $\mathrm{pfx}_{n-1}(v) = \beta$. Somit ist

$$\begin{aligned} P\big(\mathrm{pfx}_n(R(\mathcal{Z})) = \beta\big) &= P\big(\mathrm{pfx}_{n-1}(R(\mathcal{Z})) = \beta\big) \\ \text{und} \quad P\big(\mathrm{pfx}_n(I(\mathcal{Z})) = \beta\big) &= P\big(\mathrm{pfx}_{n-1}(I(\mathcal{Z})) = \beta\big), \end{aligned}$$

woraus mit (3.13)

$$P\big(\mathrm{pfx}_{n-1}\big(R(\mathcal{Z})\big) = \beta\big) \neq P\big(\mathrm{pfx}_{n-1}\big(I(\mathcal{Z})\big) = \beta\big)$$

folgt, im Widerspruch zur Minimalität von n.

Es bleiben also nur die Fälle $|\beta| = n-1$ und $|\beta| = n$.

Wir betrachten nun den Fall $|\beta| = n-1$. Wegen der Minimalität von n ist

$$\begin{aligned} 0 &= P\big(\mathrm{pfx}_{n-1}(R(\mathcal{Z})) = \beta\big) - P\big(\mathrm{pfx}_{n-1}(I(\mathcal{Z})) = \beta\big) \\ &= \sum_{x\in T} P\big(\mathrm{pfx}_n(R(\mathcal{Z})) = \beta x\big) - \sum_{x\in T} P\big(\mathrm{pfx}_n(I(\mathcal{Z})) = \beta x\big) \\ &\quad + P\big(\mathrm{pfx}_n(R(\mathcal{Z})) = \beta\big) - P\big(\mathrm{pfx}_n(I(\mathcal{Z})) = \beta\big) \end{aligned}$$

Da die beiden letzten Summanden nach (3.13) ungleich sind, muß auch

$$\sum_{x\in T} P\big(\mathrm{pfx}_n(R(\mathcal{Z})) = \beta x\big) \neq \sum_{x\in T} P\big(\mathrm{pfx}_n(I(\mathcal{Z})) = \beta x\big)$$

sein, also gibt es ein $x \in T$, so daß $P\big(\mathrm{pfx}_n(R(\mathcal{Z})) = \beta x\big) \neq P\big(\mathrm{pfx}_n(I(\mathcal{Z})) = \beta x\big)$. Es existiert also ein $\beta' := \beta x$, so daß (3.13) auch für β' gilt. Da $|\beta'| = n$, können wir o. B. d. A. immer $|\beta| = n$ annehmen.

Die Zufallsvariable t_R bezeichne die von $\mathcal{Z}_{\mathtt{out}}$ in Schritt 2 in Definition 3.5 gewählte Zahl in einem Protokollauf von π, $\mathcal{Z}_{\mathtt{out}}$ und $\mathcal{A}$. Analog sei t_I die von $\mathcal{Z}_{\mathtt{out}}$ in einem Protokollauf von ρ, $\mathcal{Z}_{\mathtt{out}}$ und $\mathcal{S}$ gewählte Zahl. Es ist $P(t_R = n) = P(t_I = n) = 2^{-n}$.

Nach Konstruktion von $\mathcal{Z}_{out}$ gilt nun für die Wahrscheinlichkeit, daß $\mathcal{Z}_{out}$ die Ausgabe $\beta \in T^n$ hat, folgendes:

$$\begin{aligned} P\big(RO(\mathcal{Z}_{out}) = \beta\big) &= P(t_R = |\beta|) \cdot P\big(\mathrm{pfx}_{|\beta|}(R(\mathcal{Z})) = \beta\big) \\ \text{und} \quad P\big(IO(\mathcal{Z}_{out}) = \beta\big) &= P(t_I = |\beta|) \cdot P\big(\mathrm{pfx}_{|\beta|}(I(\mathcal{Z})) = \beta\big) \end{aligned}$$

Wir haben hierbei β mit seiner Kodierung als Element von Σ^* identifiziert, vgl. Schritt 7 in Definition 3.5. Wegen $|\beta| = n$ folgt daraus mit (3.13)

$$P\big(RO(\mathcal{Z}_{out}) = \beta\big) \neq P\big(IO(\mathcal{Z}_{out}) = \beta\big),$$

und das Lemma ist bewiesen. □

Mit Lemma 3.6 ist es nun einfach, die Äquivalenz der Sicherheit bzgl. der Sicht und der Ausgabe der Umgebung zu zeigen:

Korollar 3.7 (Perfekte Sicherheit bzgl. Sicht und Ausgabe der Umgebung sind äquivalent)
Sicherheit bedeute eines von „perfekte allgemeine Sicherheit mit Auxiliary input“, „perfekte allgemeine Sicherheit ohne Auxiliary input“, „perfekte spezielle Sicherheit mit Auxiliary input“ und „perfekte spezielle Sicherheit ohne Auxiliary input“.

Es seien π und ρ Protokolle. In diesem Falle ist π genau dann so sicher wie ρ bezüglich Sicherheit bzgl. der *Sicht* der Umgebung, wenn π so sicher wie ρ bezüglich Sicherheit bzgl. der *Ausgabe* der Umgebung ist.

Beweis: Da nach Lemma A.9 die Sicherheit bzgl. der Ausgabe aus der Sicherheit bzgl. der Sicht folgt, müssen wir nur noch zeigen, daß die Sicherheit bzgl. der Ausgabe die Sicherheit bzgl. der Sicht impliziert.

Wir betrachten zunächst den Fall der perfekten speziellen Sicherheit mit Auxiliary input. Wir nehmen an, π sei *nicht* so sicher wie ρ bezüglich der Sicherheit bzgl. der *Sicht* der Umgebung, d. h. es existieren ein zulässiger Angreifer $\mathcal{A}$ und eine zulässige Umgebung $\mathcal{Z}$, so daß für alle zulässigen Simulatoren $\mathcal{S}$ ein Sicherheitsparameter $k_{\mathcal{S}} \in \mathbb{N}$ und ein Auxiliary input $z_{\mathcal{S}} \in \Sigma^*$ existieren, so daß

$$\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k_{\mathcal{S}}, z_{\mathcal{S}}) \neq \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}}(k_{\mathcal{S}}, z_{\mathcal{S}}),$$

die Umgebung $\mathcal{Z}$ also bzgl. der Sicht unterscheidet.

Nach Lemma 3.2 ist dann auch

$$\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}_{out}}(k_{\mathcal{S}}, z_{\mathcal{S}}) \neq \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}_{out}}(k_{\mathcal{S}}, z_{\mathcal{S}}), \tag{3.14}$$

woraus folgt, daß π nicht so sicher wie ρ ist bezüglich der Sicherheit bzgl. der Ausgabe der Umgebung.

Für die Variante der perfekten speziellen Sicherheit mit Auxiliary input ist das Korollar damit gezeigt. Im Falle der Sicherheit mit Auxiliary input ist $z_{\mathcal{S}} \in \{\lambda\}$ statt in Σ^*, ansonsten ist der Beweis unverändert. Im Falle der allgemeinen Sicherheit hängt die Umgebung $\mathcal{Z}$ vom Simulator $\mathcal{S}$ ab, so daß auch $\mathcal{Z}_{out}$ von $\mathcal{S}$ abhängt. In diesem Fall zeigt (3.14) die Unsicherheit bzgl. allgemeiner Sicherheit und das Korollar ist auch für diesen Fall bewiesen. □

Aus Korollar 3.3 und Korollar 3.7 folgt ohne weiteren Beweis die Äquivalenz aller Varianten der perfekten Sicherheit:

Satz 3.8 (Äquivalenz der perfekten Sicherheitsbegriffe)
Die folgenden acht Sicherheitsbegriffe sind äquivalent: Perfekte allgemeine/spezielle Sicherheit mit/ohne Auxiliary input bzgl. der Sicht/Ausgabe der Umgebung.

4. Ein Kompositionstheorem für statistische Sicherheit

In diesem Kapitel zeigen wir, daß statistische spezielle Sicherheit nebenläufige Komposition erlaubt.

4.1. Das Problem der nebenläufigen Komposition bei spezieller Sicherheit

In Abschnitt 2.3 haben wir zwei Typen der Komposition kennengelernt, die einfache Komposition (Abschnitt 2.3.1) und die nebenläufige Komposition (Abschnitt 2.3.3). Bei der einfachen Komposition ersetzen wir in einem Protokoll σ^ρ das Unterprotokoll ρ durch ein anderes Unterprotokoll π. Das einfache Kompositionstheorem 2.19 garantiert uns dann, daß das resultierende Protokoll σ^π so sicher wie das ursprüngliche Protokoll σ^ρ ist, vorausgesetzt, π ist so sicher wie ρ. Dieses Resultat galt für alle betrachteten Varianten der Sicherheit.

Bei der nebenläufigen Komposition ist die Situation eine andere. Gegeben sind zwei Protokolle π und ρ, wobei π so sicher wie ρ ist. Wir fragen uns nun, ob die Sicherheit erhalten bleibt, wenn wir nicht nur eine Instanz von π ausführen, sondern polynomiell viele Kopien. In anderen Worten, wir fragen uns, ob $f \cdot \pi$, die f-fache nebenläufige Komposition von π, so sicher ist wie $f \cdot \rho$. Im Falle der allgemeinen Sicherheit gibt das nebenläufige Kompositionstheorem 2.26 eine positive Antwort.

Wir wollen in diesem Kapitel die Frage (positiv) beantworten, ob auch die *spezielle statistische* Sicherheit nebenläufige Komposition erlaubt (und damit auch allgemeine, vgl. Abschnitt 2.3.4). In Kapitel 3 haben wir dies bereits für den Fall der perfekten speziellen Sicherheit geklärt, indem wir gezeigt haben, daß spezielle und allgemeine Sicherheit im perfekten Fall zusammenfallen. Den komplexitätstheoretischen Fall werden wir in Kapitel 6 betrachten.

Das Ziel dieses Kapitels ist also der Beweis des folgenden Theorems:

Theorem 4.1 (Nebenläufiges Kompositionstheorem für spezielle statistische Sicherheit)

(Fortsetzung nächste Seite)

(Fortsetzung)

Es bedeute $\pi \geq \rho$, daß π so sicher wie ρ ist bezüglich statistischer spezieller Sicherheit mit/ohne Auxiliary input bzgl. der Ausgabe/der Sicht der Umgebung.

Seien π und ρ Protokolle, und f eine polynomiell-beschränkte Funktion. Dann ist $f \cdot \pi \geq f \cdot \rho$.

Bevor wir mit dem Beweis dieses Theorems beginnen, wollen wir uns zunächst kurz in Erinnerung rufen, woran der in Abschnitt 2.3.3 skizzierte Beweis des nebenläufigen Kompositionstheorems 2.26 im Falle der *speziellen* Sicherheit gescheitert ist. Wir empfehlen es dem Leser jedoch, sich im Zweifelsfalle die Beweisskizze noch einmal anzuschauen, da die Methodik in diesem Kapitel darauf aufbaut.

Die Grundidee des Beweises bestand darin, im realen Modell (also bei der Ausführung des komponierten Protokolls $f \cdot \pi$) davon auszugehen, daß neben der Umgebung $\mathcal{Z}_f$ und den f Instanzen des Protokolls π statt *eines* Angreifers noch f Kopien eines „kleinen“ Dummy-Angreifers $\tilde{\mathcal{A}}_\pi$ vorliegen. Jeder dieser Dummy-Angreifer hat einfach nur Nachrichten zwischen „seiner“ Protokollinstanz und der Umgebung durchgeleitet (vgl. Abbildung 2.8a auf Seite 90).

Da wir annehmen, daß das einzelne Protokoll π so sicher wie ρ ist, gibt es nun (im Falle der allgemeinen Sicherheit zumindest) einen zu $\tilde{\mathcal{A}}_\pi$ gehörigen Simulator $\tilde{\mathcal{S}}_\rho$, so daß π mit $\tilde{\mathcal{A}}_\pi$ und ρ mit $\tilde{\mathcal{S}}_\rho$ ununterscheidbar sind. Es liegt daher nahe, jedes Vorkommen von π und $\tilde{\mathcal{A}}_\pi$ in Abbildung 2.8a durch ρ und $\tilde{\mathcal{S}}_\rho$ zu ersetzen (vgl. Abbildung 2.8b), in der Hoffnung, daß diese simultane Ersetzung vieler ununterscheidbarer Subnetze von $\mathcal{Z}_f$ nicht bemerkt werden kann.

Dies konnten wir tatsächlich zeigen, indem wir Zwischenstufen zwischen der Konstruktion aus Abbildungen 2.8a und 2.8b betrachten, in denen die realen Protokollinstanzen und Angreifer eine nach der anderen durch ideale Protokollinstanzen und Simulatoren ersetzt werden (Abbildung 2.8c). Wir haben die Tatsache ausgenutzt, daß, wenn wir ein reales Protokoll π mit Angreifer $\tilde{\mathcal{A}}_\pi$ durch ein ideales Protokoll ρ mit Simulator $\tilde{\mathcal{S}}_\rho$ ersetzen, wir den Rest des Netzwerkes als eine große Umgebung $\mathcal{Z}_l$ auffassen können (diese Umgebung $\mathcal{Z}_l$ ist in Abbildung 2.8c durch eine gestrichelte Linie gekennzeichnet). Diese darf nach Annahme das Netzwerk vor und nach der Ersetzung nicht unterscheiden können, und insbesondere kann es dann auch nicht die Umgebung $\mathcal{Z}_f$, die ein Teil dieser großen Umgebung ist.[1] Dies ermöglicht es dann zu schließen, daß die f Instanzen von $\tilde{\mathcal{S}}_\rho$ – zu einer Maschine zusammengefaßt – tatsächlich einen guten

[1] Dies ist eine vereinfachte Darstellung. Aus technischen Gründen haben wir eine Umgebung $\mathcal{Z}^R$ konstruiert, welche die Anzahl der realen und der idealen Instanzen *zufällig* wählt, und ausgenutzt, daß diese Umgebung die Ersetzung nicht bemerkt, s. u.

Simulator darstellen, womit die Sicherheit des komponierten Protokolls gezeigt ist.

Im Falle der speziellen Sicherheit haben wir aber das folgende Problem: Anders als bei der allgemeinen Sicherheit gibt es nicht *den* Simulator $\tilde{\mathcal{S}}_\rho$, sondern der Simulator hängt von der betrachteten Umgebung ab. Wir müssen also eine Umgebung festlegen, relativ zu der der Simulator $\tilde{\mathcal{S}}_\rho$ gewählt werden soll. Wir haben im Beweis die Tatsache ausgenutzt, daß die Umgebung $\mathcal{Z}_R$ (die eine zufällige Anzahl von realen und idealen Instanzen simuliert) eine Ersetzung einer realen Protokollinstanz mit Angreifer durch eine ideale Protokollinstanz mit Simulator nicht bemerkt. Also müßte der Simulator $\tilde{\mathcal{S}}_\rho$ als Simulator für diese Umgebung $\mathcal{Z}^R$ gewählt werden. Unglücklicherweise ist diese Umgebung wiederum abhängig von $\tilde{\mathcal{S}}_\rho$ konstruiert, sie simuliert nämlich Kopien von $\tilde{\mathcal{S}}_\rho$ zusammen mit den Instanzen des idealen Protokolls ρ. Es ist nicht klar, ob diese zyklische Definition einen Fixpunkt hat.

Ein Ansatz, um dieses Problem zu lösen, wäre, die l-te Instanz $\tilde{\mathcal{S}}_l$ des Simulators in Abbildung 2.8b gerade als Simulator für die Umgebung $\mathcal{Z}_l$ zu wählen. Diese simuliert nur die Instanzen $\tilde{\mathcal{S}}_{l+1}, \ldots, \tilde{\mathcal{S}}_f$, die Konstruktion wäre also nicht zyklisch. Leider treten dabei die folgenden zwei Probleme auf:

- Sollte die Komplexität eines Simulators von der der Umgebung abhängen (z. B. doppelt so groß sein), so wird jeder Simulator doppelt so aufwendig sein wie alle zuvor konstruierten zusammen, der zuletzt konstruierte hat dann eine Komplexität, die exponentiell in f ist. (Im Falle der statistischen Sicherheit stört uns dies nicht, aber im Falle der komplexitätstheoretischen führt dies zu einem ungültigen Simulator. Eine genaue Betrachtung des in Kapitel 6 konstruierten Gegenbeispiels zeigt, daß dort genau dieses Problem auftritt.)

- Die Tatsache, daß bei jeder Ersetzung eines Subnetzes die Ausgabe der Umgebung $\mathcal{Z}_f$ vor und nach der Ersetzung ununterscheidbar ist, erlaubt es noch nicht zu schließen, daß auch die Ausgabe vor der ersten Ersetzung und die nach der letzten Ersetzung (also in Abbildungen 2.8a und 2.8b) ununterscheidbar sind. Dies werde durch folgendes Zahlenbeispiel erläutert: Bei der i-ten Ersetzung eines Simulators trete ein statistischer Abstand von 2^{i-k-1} zwischen der Ausgabe der Umgebung vor und nach der Ersetzung auf (als Funktion im Sicherheitsparameter k). Dies ist denkbar, da für festes i diese Funktion vernachlässigbar ist. Der statistische Abstand zwischen der Ausgabe vor der ersten und nach der letzten Ersetzung ist dann durch $\delta := \sum_{i=1}^{f(k)} 2^{i-k-1}$ beschränkt. Ist beispielsweise $f(k) = k$, so ist $\delta \approx 1$, was keineswegs vernachlässigbar ist.

  Im Beweis des nebenläufigen Kompositionstheorems 2.26 haben wir dies Problem umgangen, indem wir die Umgebung $\mathcal{Z}^R$ eingeführt haben, wel-

che einen *zufälligen* Zwischenschritt zwischen den Situationen in Abbildungen 2.8a und 2.8b simuliert hat, und damit bewirkt hat, daß wir eine *uniforme* Schranke für den statistischen Abstand vor und nach einer Ersetzung eines Subnetzes erhalten haben. Diese steigt dann insbesondere nicht mehr exponentiell mit der Nummer der Ersetzung.

Dieser Ansatz steht uns jedoch nicht in dieser Form zur Verfügung, da die Umgebung $\mathcal{Z}^R$ potentiell alle Simulatoren simulieren muß, und damit wieder die oben beschriebene zyklische Abhängigkeit zwischen Umgebung und Simulator besteht.

Im nächsten Abschnitt werden wir zeigen, wie man das Problem dieser zyklischen Abhängigkeit lösen kann, ohne daß sich der statistische Abstand zwischen der Ausgabe der Umgebung im realen und im idealen Modell exponentiell aufschaukelt.

4.2. Der Beweis des Kompositionstheorems

Bevor wir den Beweis des nebenläufigen Kompositionstheorems 4.1 im Detail präsentieren, skizzieren wir zunächst die verwendete Methode. Dazu beginnen wir mit einer bereits im vorangegangenen Abschnitt vorgestellten Technik, die wir dann schrittweise verfeinern, bis sie zum Erfolg führt. Die im folgenden gebrachten Argumente sind nicht als rigorose Argumentation, nicht einmal als Beweisskizze zu sehen, sondern sollen eine erste Intuition vermitteln, die das Verständnis des eigentlichen Beweises vereinfacht.

1. Um eine zyklische Abhängigkeit zwischen der Umgebung und dem Simulator zu verhindern, definieren wir die Umgebung $\mathcal{Z}_l$ und den Simulator $\mathcal{S}_l$ induktiv wie folgt: Die Umgebung $\mathcal{Z}_l$ simuliert $l-1$ Kopien des idealen Modells bestehend aus ρ und $\mathcal{S}_i$ für $i = 1, \ldots, l-1$ und $f-l-1$ Kopien des realen Modells bestehend aus π und $\tilde{\mathcal{A}}_\pi$.

   $\mathcal{S}_l$ wiederum ist definiert als ein für die Umgebung $\mathcal{Z}_l$ geeigneter Simulator.[2] Wir bezeichnen den statistischen Abstand der Ausgabe der Umgebung $\mathcal{Z}_l$ zwischen realem und idealem Modell mit diesem Simulator als μ_l.

2. Wir können nun ähnlich wie im Beweis des nebenläufigen Kompositionstheorems 2.26 mit der Umgebung $\mathcal{Z}_1$ beginnen (welche nur Dummy-Angreifer und Instanzen des realen Protokolls enthält), und dann schritt-

[2] Hier und im folgenden verstehen wir unter einem für eine Umgebung $\mathcal{Z}$ geeigneten Simulator einen Simulator $\mathcal{S}$, so daß die Umgebung $\mathcal{Z}$ nicht unterscheiden kann zwischen dem realen Protokoll π mit dem Dummy-Angreifer $\tilde{\mathcal{A}}_\pi$ und dem idealen Protkoll ρ mit dem Simulator $\mathcal{S}$. Ein geeigneter Simulator existiert für jede Umgebung nach der Definition der speziellen Sicherheit.

weise jeweils eine Instanz von Dummy-Angreifer und realem Protokoll durch eine Instanz von $\mathcal{S}_l$ und idealem Protokoll ersetzen (wobei wir mit $l = 1$ beginnen). Schließlich erhalten wir die Umgebung $\mathcal{Z}_{f(k)+1}$, welche nur noch Simulatoren und ideale Protokollinstanzen enthält. Dabei summiert sich der statistische Abstand auf, wir erhalten einen statistischen Abstand von $\mu := \sum_{l=1}^{f(k)} \mu_l$ zwischen der Ausgabe von $\mathcal{Z}_0$ und $\mathcal{Z}_{f(k)}$. Wie bereits im vorangegangen Abschnitt demonstriert, ist μ nicht notwendig vernachlässigbar, selbst wenn alle μ_l dies sind. Unser Ziel ist es nun also, die $\mathcal{S}_i$ so zu wählen, daß $\sum_{l=1}^{f(k)} \mu_l$ vernachlässigbar wird.

3. Um dies zu erreichen, verlangen wir, daß die Simulatoren $\mathcal{S}_l$ in folgendem Sinne optimal sind: Für jeden Simulator $\mathcal{S}'$ gilt, daß der statistische Abstand der Ausgabe von $\mathcal{Z}_l$ mit $\mathcal{S}_l$ kleiner-gleich dem statistischen Abstand der Ausgabe von $\mathcal{Z}_l$ mit $\mathcal{S}'$ ist.[3] Man beachte, daß diese Konstruktion benutzt, daß wir unbeschränkte Simulatoren zulassen.

4. Um zu zeigen, daß für diese optimalen Simulatoren $\sum_{l=1}^{f(k)} \mu_l$ vernachlässigbar ist, konstruieren wir zunächst eine weitere Umgebung $\mathcal{Z}_\infty$. Diese Umgebung soll das Verhalten aller Umgebungen $\mathcal{Z}_l$ imitieren können, so daß ein für $\mathcal{Z}_\infty$ geeigneter Simulator $\mathcal{S}_\infty$ zugleich ein geeigneter Simulator für alle $\mathcal{Z}_l$ ist. Dies erreichen wir wie folgt: Es sei $\mathcal{D}$ die Verteilung, die dem Wert l die Wahrscheinlichkeit $1/l^2\gamma$ mit $\gamma := \sum_{i=1}^{\infty} 1/i^2$ zuordnet. Bei ihrer ersten Aktivierung wählt $\mathcal{Z}_\infty$ ein l gemäß der Verteilung $\mathcal{D}$ und verhält sich dann wie $\mathcal{Z}_l$.

   Es sei $\mathcal{S}_\infty$ ein für $\mathcal{Z}_\infty$ geeigneter Simulator, so daß der statistische Abstand der Ausgabe von $\mathcal{Z}$ durch die vernachlässigbare Funktion μ_∞ beschränkt ist.

5. Da sich $\mathcal{Z}_\infty$ mit Wahrscheinlichkeit $P_l := 1/l^2\gamma$ wie $\mathcal{Z}_l$ verhält, ist $\mathcal{S}_\infty$ auch ein für $\mathcal{Z}_l$ geeigneter Simulator. Genauer ist der statistische Abstand der Ausgabe von $\mathcal{Z}_l$ mit $\mathcal{S}_\infty$ durch $\mu_\infty/P_l = l^2\gamma\mu_\infty$ beschränkt. Da wir gefordert hatten, daß $\mathcal{S}_l$ ein für $\mathcal{Z}_l$ optimaler Simulator ist, muß $\mathcal{S}_l$ mindestens ebenso gut sein, d. h. $\mu_l \leq l^2\gamma\mu_\infty$. Es folgt $\sum_{l=1}^{f(k)} \mu_l \leq \sum_{l=1}^{f(k)} l^2\gamma\mu_\infty \leq f(k)^3\gamma\mu_\infty$, was für polynomiell-beschränktes f vernachlässigbar ist.

Gerüstet mit diesen Vorbemerkungen können wir nun den Beweis des nebenläufigen Kompositionstheorems 4.1 betrachten. Dazu müssen wir zunächst den Dummy-Angreifer definieren, da wir diesen in der Beweisskizze zu Theorem 2.26 nur informell spezifiziert hatten.

[3] Genaugenommen erlauben wir in dieser Abschätzung einen Fehler von 2^{-k}, da sonst die Existenz eines optimalen Simulators nicht garantiert ist.

Definition 4.2 (Unbeschränkter Dummy-Angreifer)
Es sei ein Protokoll π gegeben. Der *unbeschränkte*[4] *Dummy-Angreifer* $\tilde{\mathcal{A}}$ *zu* π hat den Namen `adv` und die folgenden Ports:

- Für jeden *ausgehenden* Angreiferport `adv_`p des Protokolls π hat $\tilde{\mathcal{A}}$ einen *eingehenden* Angreiferport `adv_`p und einen *ausgehenden* Umgebungsport `env_adv_`p.
- Für jeden *eingehenden* Angreiferport `adv_`p des Protokolls π hat $\tilde{\mathcal{A}}$ einen *ausgehenden* Angreiferport `adv_`p und einen *eingehenden* Umgebungsport `env_adv_`p.
- Weiterhin hat $\tilde{\mathcal{A}}$ den Port `clk` ($\tilde{\mathcal{A}}$ ist also Scheduler), und einen ausgehenden Umgebungsport `env_clk`.

Bei jeder Aktivierung verhält sich der Dummy-Angreifer $\tilde{\mathcal{A}}$ wie folgt: Erhält er eine Nachricht m über einen eingehenden Angreiferport `adv_`p, so wird m über den ausgehenden Umgebungsport `env_adv_`p geschickt. Erhält er eine Nachricht m über einen eingehenden Umgebungsport `env_adv_`p, so wird m über den ausgehenden Angreiferport `adv_`p geschickt. Erhält er eine Nachricht m über den Port `clk`, so wird m über `env_clk` geschickt.

Da folgende Lemma garantiert uns, daß es genügt, Sicherheit bzgl. des Dummy-Angreifers zu betrachten:

Lemma 4.3 (Vollständigkeit des Dummy-Angreifers)
Es seien π und ρ Protokolle. Es sei $\tilde{\mathcal{A}}$ der unbeschränkte Dummy-Angreifers zu π. Dann ist π so sicher wie ρ bezüglich perfekter/statistischer allgemeiner/spezieller Sicherheit mit/ohne Auxiliary input bzgl. der Sicht/Ausgabe der Umgebung, wenn π so sicher wie ρ ist bezüglich ebendieser Sicherheit mit Angreifern in $\{\tilde{\mathcal{A}}\}$.

Da es sich hierbei um eine wohlbekannte Tatsache handelt, skizzieren wir den Beweis nur grob für den Fall der statistischen speziellen Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung.

Beweisskizze: Im folgenden bezeichne Sicherheit die statistische spezielle Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung.

Trivialerweise impliziert Sicherheit ohne Beschränkung der Angreifermenge Sicherheit mit Angreifern in $\{\tilde{\mathcal{A}}\}$. Daher müssen wir nur die umgekehrte Richtung zeigen. Sei also π so sicher wie ρ bezüglich der Sicherheit mit Angreifern

[4] Wir nennen diesen Dummy-Angreifer *unbeschränkt*, weil er – im Gegensatz zum p-Dummy-Angreifer aus Definition 7.13 – Anzahl und Länge der weitergeleiteten Nachrichten nicht beschränkt.

in $\{\tilde{\mathcal{A}}\}$. Dann existiert für jede Umgebung $\mathcal{Z}$ ein Simulator $\tilde{\mathcal{S}}_{\mathcal{Z}}$, so daß

$$\text{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}} \approx \text{OUTPUT}_{\rho,\tilde{\mathcal{S}}_{\mathcal{Z}},\mathcal{Z}}. \tag{4.1}$$

Hierbei bezeichnet $\approx$ die statistische Ununterscheidbarkeit. Die Angabe des Sicherheitsparameters k und des Auxiliary inputs z lassen wir an dieser Stelle weg.

Es sei nun ein Angreifer $\mathcal{A}$ gegeben. Um zu zeigen, daß π so sicher wie ρ ist bezüglich der Sicherheit ohne Einschränkung der Angreifer, müssen wir zeigen, daß für jede Umgebung ein Simulator $\mathcal{S}_{\mathcal{Z}}$ existiert, so daß

$$\text{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{OUTPUT}_{\rho,\mathcal{S}_{\mathcal{Z}},\mathcal{Z}}. \tag{4.2}$$

Es sei nun $\mathcal{Z}_{\mathcal{A}}$ die Umgebung, die die Maschinen $\mathcal{Z}$ und $\mathcal{A}$ simuliert, und alle Kommunikation zwischen diesen beiden durchleitet. Weiterhin leite $\mathcal{Z}_{\mathcal{A}}$ die Kommunikation zwischen $\mathcal{Z}$ und dem Protokoll durch, und die Kommunikation zwischen $\mathcal{A}$ und dem Protokoll durch eine Instanz des Dummy-Angreifers $\tilde{\mathcal{A}}$ (vgl. Abbildung 4.1b). Dann ist

$$\text{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}} = \text{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_{\mathcal{A}}}.$$

Mit (4.1) erhalten wir daraus

$$\begin{aligned}\text{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}} &= \text{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_{\mathcal{A}}} \\ &\overset{(4.1)}{\approx} \text{OUTPUT}_{\rho,\tilde{\mathcal{S}}_{\mathcal{Z}_{\mathcal{A}}},\mathcal{Z}_{\mathcal{A}}} = \text{OUTPUT}_{\rho,\mathcal{S}_{\mathcal{Z}},\mathcal{Z}}\end{aligned}$$

wenn wir $\mathcal{S}_{\mathcal{Z}}$ als den Simulator konstruieren, der $\mathcal{A}$ und $\tilde{\mathcal{S}}_{\mathcal{Z}_{\mathcal{A}}}$ simuliert (vgl. Abbildung 4.1d). Damit ist (4.2) gezeigt.

Die anderen Fälle werden analog gezeigt. Bei der Sicherheit bzgl. der Sicht der Umgebung muß man beachten, daß die Sicht von $\mathcal{Z}$ eine deterministische Funktion der Sicht von $\mathcal{Z}_{\mathcal{A}}$ ist. Bei der allgemeinen Sicherheit ist $\mathcal{S}_{\mathcal{Z}}$ von $\mathcal{Z}$ unabhängig, da $\tilde{\mathcal{S}}_{\mathcal{Z}_{\mathcal{A}}}$ von $\mathcal{Z}$ unabhängig ist. □

Wir können uns nun dem zentralen Beweis dieses Kapitels zuwenden:

Beweis von Theorem 4.1: Wir zeigen zunächst den Fall der statistischen speziellen Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung.

Es sei also π so sicher wie ρ bezüglich dieses Sicherheitsbegriffs. Wir müssen nun zeigen, daß auch $f \cdot \pi \geq f \cdot \rho$. Sei $\tilde{\mathcal{A}}^*$ der unbeschränkte Dummy-Angreifer für $f \cdot \pi$. Nach Lemma 4.3 genügt es zu zeigen, daß für jede zulässige Umgebung $\mathcal{Z}^*$ ein zulässiger Simulator $\mathcal{S}^*$ existiert, so daß

$$\text{OUTPUT}_{f\cdot\pi,\tilde{\mathcal{A}}^*,\mathcal{Z}^*} \approx \text{OUTPUT}_{f\cdot\rho,\mathcal{S}^*,\mathcal{Z}^*} \tag{4.3}$$

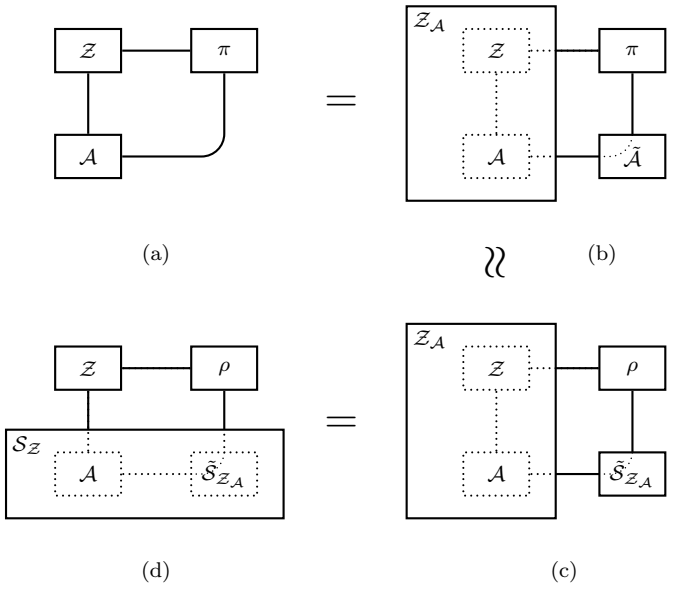


Abbildung 4.1.: Vollständigkeit des Dummy-Angreifers.
(a) Wir beginnen mit einem Netzwerk bestehend aus Umgebung $\mathcal{Z}$, Angreifer $\mathcal{A}$ und Protokoll π.
(b) Wir fügen den Dummy-Angreifer $\tilde{\mathcal{A}}$ zwischen π und $\mathcal{A}$ ein, welcher die Kommunikation einfach nur weiterleitet und fassen $\mathcal{Z}$ und $\mathcal{A}$ zu einer neuen Umgebung $\mathcal{Z}_{\mathcal{A}}$ zusammen.
(c) Weil π bezüglich des Dummy-Angreifers sicher ist, können wir π durch ρ ersetzen, wenn wir $\tilde{\mathcal{A}}$ durch den zugehörigen Simulator $\tilde{\mathcal{S}}_{\mathcal{Z}_{\mathcal{A}}}$ ersetzen.
(d) Ersetzen wir $\mathcal{Z}_{\mathcal{A}}$ wieder durch die enthaltenen Maschinen $\mathcal{Z}$ und $\mathcal{A}$, und fassen dann $\mathcal{A}$ und $\tilde{\mathcal{S}}_{\mathcal{Z}_{\mathcal{A}}}$ zu einem Simulator $\mathcal{S}_{\mathcal{Z}}$ zusammen, so erhalten wir das Netz bestehend aus $\mathcal{Z}, \mathcal{S}_{\mathcal{Z}}, \pi$.

gilt. Hierbei bezeichne $\approx$ die statistische Ununterscheidbarkeit. Weiterhin geben wir der Übersichtlichkeit halber Sicherheitsparameter k und Auxiliary input z nicht explizit an (d. h. $\text{OUTPUT}_{\dots}$ steht hier und im folgenden für die Familie $\{\text{OUTPUT}_{\dots}(k,z)\}_{k\in\mathbb{N},z\in\Sigma^*}$).

Wir nehmen im weiteren o. B. d. A. die folgenden Tatsachen an:

- Die Protokolle π und ρ haben die gleichen Protokollports. Ansonsten fügen wir einfach die fehlenden Ports hinzu und ignorieren Nachrichten auf diesen.
- $\mathcal{Z}^*$ hat keine Leitungen zu sich selbst (d. h. $\mathcal{Z}^*$ hat keinen Port p, der sowohl ein- als auch ausgehender Port ist). Eine solche Umgebung könnten wir immer durch eine ersetzen, die, anstatt sich selbst Nachrichten zu schicken, dies nur simuliert.
- $\mathcal{Z}^*$ hat keine Ports, die nicht mit Angreifer oder Protokoll verbunden sind, analoges gilt für Angreifer, Simulatoren und Protokolle (das Senden einer Nachricht über einen solchen Port können wir dadurch ersetzen, daß einfach keine Nachricht gesandt wird).
- $\mathcal{Z}^*$ sendet nur Nachrichten der Form (sid, m) mit $sid \in \{1, \dots, f(k)\}$. Andere Nachrichten würden sowieso von den Protokollen $f \cdot \pi$ und $f \cdot \rho$ und damit effektiv auch vom Dummy-Angreifer $\tilde{\mathcal{A}}^*$ ignoriert werden.

Es sei $\tilde{\mathcal{A}}$ der unbeschränkte Dummy-Angreifer zu π.

Gegeben Simulatoren $\mathcal{S}_1, \dots, \mathcal{S}_{l-1}$ (die für ρ und $\tilde{\mathcal{A}}$ zulässig sind) definieren wir für $l \geq 1$ die Umgebung $\mathcal{Z}_l = \mathcal{Z}_l(\mathcal{S}_1, \dots, \mathcal{S}_{l-1})$ wie folgt (vgl. Abbildung 4.2a): $\mathcal{Z}_l$ simuliert $\mathcal{Z}^*$ und für $sid = 1, \dots, l-1$ je eine Instanz $\mathcal{S}_{sid}$ und eine Instanz ρ_{sid} des Protokolls ρ, und für $sid = l+1, \dots, f(k)$ je eine Instanz $\tilde{\mathcal{A}}_{sid}$ und π_{sid} von $\tilde{\mathcal{A}}$ und π. Maschinen mit gleichem Index sid (im folgenden Session-ID genannt) werden miteinander verbunden. Nachrichten von Maschinen mit Session-ID sid an $\mathcal{Z}^*$ werden an $\mathcal{Z}^*$ als (sid, m) ausgeliefert. Analog werden Nachrichten von $\mathcal{Z}^*$ der Form (sid, m) an die entsprechenden Maschinen mit Session-ID sid ausgeliefert. Eine Sonderrolle nimmt $sid = l$ ein. Eine Nachricht (l, m) von $\mathcal{Z}^*$ wird von $\mathcal{Z}_l$ nach außen geleitet (d. h. an die mit $\mathcal{Z}_l$ verbundenen Maschinen geschickt, welche auch immer dies sein mögen), und eine von außen kommende Nachricht m wird an $\mathcal{Z}^*$ als (l, m) weitergeleitet. Die Ausgabe von $\mathcal{Z}_l$ ist die vom simulierten $\mathcal{Z}^*$.

Im Detail sieht die Definition von $\mathcal{Z}_l$ wie folgt aus: (Wir empfehlen dem Leser beim ersten Lesen diese Details zu überspringen und sich an die im vorangegangenen Absatz gegebene Kurzbeschreibung und Abbildung 4.2a zu halten.)

- $\mathcal{Z}_l$ hat die gleichen Ports wie $\mathcal{Z}^*$.
- $\mathcal{Z}_l$ simuliert:
  - eine Instanz von $\mathcal{Z}^*$,
  - für $i = 1, \dots, l-1$ je eine Instanz ρ_i des Protokolls ρ (d. h. je eine Instanz jeder Maschine in ρ),
  - für $i = 1, \dots, l-1$ je eine Instanz von $\mathcal{S}_i$,

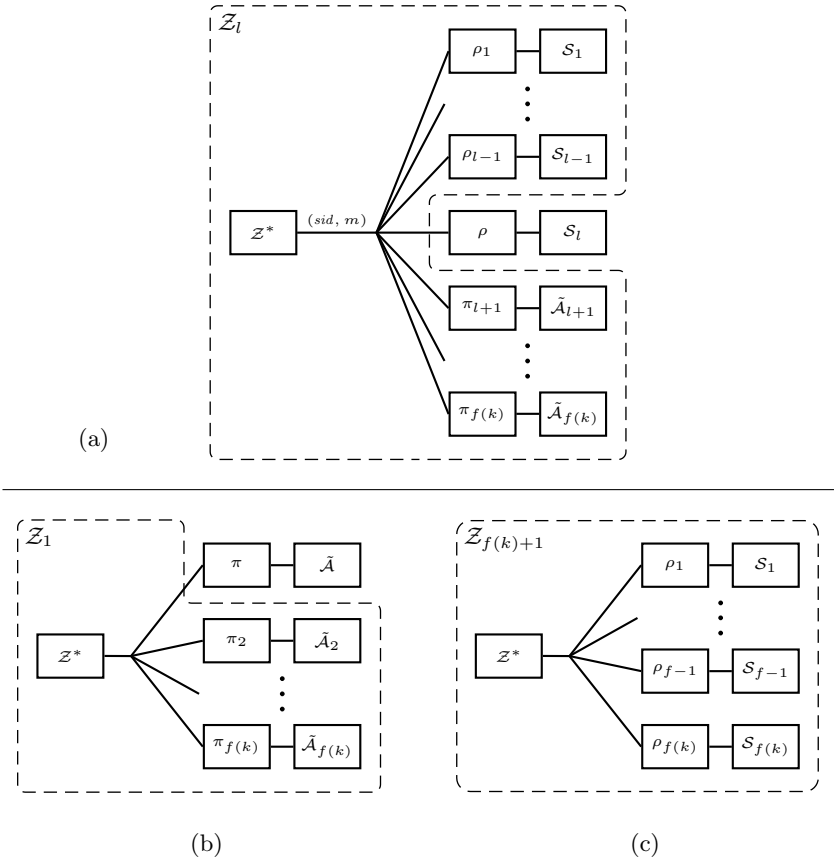


Abbildung 4.2.: Beweisschritte im nebenläufigen Kompositionstheorem.
(a) Konstruktion der Umgebung $\mathcal{Z}_l$ (gestrichelte Umrandung). Die Verbindungen zwischen den Simulatoren/Angreifern und $\mathcal{Z}^*$ wurden der Übersichtlichkeit halber weggelassen. Beispielhaft ist $\mathcal{Z}_l$ mit ρ und $\mathcal{S}_l$ verbunden.
(b) Der Spezialfall $l = 1$.
(c) Der Spezialfall $l = f(k) + 1$.

  - für $i = l+1, \ldots, f(k)$ je eine Instanz π_i des Protokolls π,
  - für $i = l+1, \ldots, f(k)$ je eine Instanz $\tilde{\mathcal{A}}_i$ des unbeschränkten Dummy-Angreifers $\tilde{\mathcal{A}}$ zu π.
- Sendet $\mathcal{Z}^*$ eine Nachricht der Form (sid, m) über den ausgehenden Port p, so unterscheiden wir die folgenden Fälle:
  - $sid \in \{1, \ldots, l-1\}$: Es sei $M \in \{\mathcal{S}_{sid}\} \cup \rho_{sid}$ die simulierte Maschine mit dem eingehenden Port p: Dann aktivieren wir M mit Nachricht m auf Port p.
  - $sid \in \{l+1, \ldots, f(k)\}$ und eine simulierte Maschine $M \in \{\tilde{\mathcal{A}}_{sid}\} \cup \pi_{sid}$ hat einen eingehenden Port p: Dann aktivieren wir M mit Nachricht m auf Port p.
  - $sid = l$: Dann sendet $\mathcal{Z}_l$ die Nachricht m auf ihrem ausgehenden Port p.
- Sendet eine simulierte Maschine $M \in \{\tilde{\mathcal{A}}_{sid}\} \cup \pi_{sid}$ eine Nachricht m auf einem ausgehenden Port $p \neq$ `env_clk`, so unterscheiden wir die folgenden Fälle:
  - Eine simulierte Maschine $M' \in \{\tilde{\mathcal{A}}_{sid}\} \cup \pi_{sid}$ hat einen eingehenden Port p. Dann wird M' mit Nachricht m auf Port p aktiviert. (Dies gilt auch für den Fall $M = M'$).
  - $\mathcal{Z}^*$ hat einen eingehenden Port p. Dann wird $\mathcal{Z}^*$ mit der Nachricht (sid, m) auf Port p aktiviert.
- Analog für $M \in \{\mathcal{S}_{sid}\} \cup \rho_{sid}$.
- Sendet eine simulierte Maschine $M \in \{\tilde{\mathcal{A}}_{sid}\} \cup \pi_{sid}$ oder $M \in \{\mathcal{S}_{sid}\} \cup \rho_{sid}$ nach ihrer Aktivierung keine Nachricht, so wird $\tilde{\mathcal{A}}_{sid}$ bzw. $\mathcal{S}_{sid}$ mit Nachricht λ auf Port `clk` aktiviert (der Scheduler mit Session-ID sid wird aktiviert).
- Die Nachrichten auf den `clk`- und `env_clk`-Ports werden wie folgt durchgereicht:[5]
  - Sendet ein simulierter Dummy-Angreifer $\tilde{\mathcal{A}}_{sid}$ mit $sid > l+1$ eine Nachricht m über `env_clk`, so wird $\tilde{\mathcal{A}}_{sid-1}$ mit m auf Port `clk` aktiviert.
  - Sendet der simulierte Dummy-Angreifer $\tilde{\mathcal{A}}_{l+1}$ eine Nachricht m über `env_clk`, so endet $\mathcal{Z}_l$'s aktuelle Aktivierung, ohne daß eine Nachricht gesandt wird. (Effektiv wird also der (nicht von $\mathcal{Z}_l$ simulierte) Scheduler aufgerufen.)
  - Erhält $\mathcal{Z}_l$ über einen eingehenden Port $p \neq$ `env_clk` eine Nachricht m und ist $l > 1$, so aktivieren wir $\mathcal{S}_{l-1}$ mit Nachricht m auf Port `clk`. Ist jedoch $l = 1$, so wird $\mathcal{Z}^*$ mit Nachricht m auf Port `env_clk`

[5] Der Effekt ist, daß das Token entlang fallender Session-IDs durchgereicht wird (wobei der nicht simulierte Scheduler die Session-ID l erhält). Dies erweist sich als notwendig, da wir anders als bei den anderen Ports hier die Session-ID nicht an die Nachricht anhängen können (denn diese ist immer λ).

aktiviert.
  - Sendet $\mathcal{S}_{sid}$ mit $sid > 1$ eine Nachricht m über Port `env_clk`, so wird $\mathcal{S}_{sid-1}$ mit m über `clk` aktiviert.
  - Sendet $\mathcal{S}_1$ eine Nachricht m über Port `env_clk`, so wird $\mathcal{Z}^*$ mit m auf Port `env_clk` aktiviert.
- Liefert $\mathcal{Z}^*$ Ausgabe o, so liefert auch $\mathcal{Z}_l$ Ausgabe o und terminiert.

Damit die Definition von $\mathcal{Z}_l$ vollständig ist, müssen wir noch die Simulatoren $\mathcal{S}_i$ definieren. Hierzu definieren wir zunächst, was wir unter einem *fast optimalen* Simulator für eine Umgebung verstehen. Für eine Umgebung $\mathcal{Z}$ und einen Simulator $\mathcal{S}$ sei $\delta(\mathcal{S}, \mathcal{Z})$ definiert als

$$\delta_{\mathcal{S},\mathcal{Z}}(k) := \sup_{z \in \Sigma^*} \Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)\big)$$

Wir nennen einen Simulator $\mathcal{S}$ *fast optimal* für eine Umgebung $\mathcal{Z}$, wenn für jeden (zulässigen) Simulator $\mathcal{S}'$ gilt: $\delta_{\mathcal{S},\mathcal{Z}}(k) \leq \delta_{\mathcal{S}',\mathcal{Z}}(k) + 2^{-k}$ für alle $k \in \mathbb{N}$ (nicht nur für hinreichend große k).

Es existiert immer ein zulässiger fast optimaler Simulator $\mathcal{S}_{opt}$ zu einer Umgebung $\mathcal{Z}$: Für jedes k sei $\mathcal{S}_{opt,k}$ ein Simulator mit

$$\delta_{\mathcal{S}_{opt,k},\mathcal{Z}} \leq \inf_{\mathcal{S}'} \delta_{\mathcal{S}',\mathcal{Z}}(k) + 2^{-k}.$$

Dann ist der Simulator $\mathcal{S}_{opt}$, der sich für Sicherheitsparameter k verhält wie $\mathcal{S}_{opt,k}$, fast optimal für $\mathcal{Z}$.[6]

Es sei dann $\mathcal{S}_l$ ein für $\mathcal{Z}_l$ fast optimaler zulässiger Simulator. Man beachte, daß $\mathcal{Z}_l$ und $\mathcal{S}_l$ rekursiv definiert sind: $\mathcal{Z}_l$ ist definiert in Abhängigkeit von $\mathcal{S}_1, \ldots, \mathcal{S}_{l-1}$, und $\mathcal{S}_l$ wiederum ist in Abhängigkeit von $\mathcal{Z}_l$ definiert.

Wir betrachten nun den Fall $l = 1$. In diesem Fall simuliert $\mathcal{Z}_l = \mathcal{Z}_1$ für $sid = 2, \ldots, f(k)$ Instanzen von $\tilde{\mathcal{A}}$ und π. Verbinden wir $\mathcal{Z}_1$ mit $\tilde{\mathcal{A}}$ und π, so erhalten wir eine Instanz von $\mathcal{Z}$ zusammen mit f Instanzen von $\tilde{\mathcal{A}}$ und π (vgl. Abbildung 4.2b). Man überzeugt sich daher leicht, daß

$$\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_1} = \mathrm{OUTPUT}_{f\cdot\pi,\tilde{\mathcal{A}}^*,\mathcal{Z}^*}. \tag{4.4}$$

(Man beachte hierbei, daß $\tilde{\mathcal{A}}^*$ wie $\tilde{\mathcal{A}}$ nichts weiter tut, als Nachrichten durchzuleiten.)[7]

[6] Hier brauchen wir, daß der Simulator $\mathcal{S}_{opt}$ unbeschränkt sein darf.

[7] Einen Sonderfall stellt lediglich die Weiterleitung des Tokens durch die `clk`- und `env_clk`-Ports dar. Hier muß man beachten, daß $\tilde{\mathcal{A}}^*$ bei Erhalt einer Nachricht auf `clk` diese sofort über `env_clk` an $\mathcal{Z}^*$ weiterleitet, wohingegen $\tilde{\mathcal{A}}$ zunächst die Nachricht über `env_clk` an $\mathcal{Z}_l$ sendet, woraufhin die Nachricht durch die simulierten Instanzen von $\tilde{\mathcal{A}}$ bis zur Instanz $\tilde{\mathcal{A}}_1$ durchgeleitet wird, welche schließlich über `env_clk` die Nachricht an die simulierte Instanz von $\mathcal{Z}^*$ schickt. Schließlich landet die Nachricht also in beiden Fällen bei $\mathcal{Z}^*$, so daß die Ausführung daraufhin identisch weiterläuft.

Nun betrachten wir die Umgebung $\mathcal{Z}_{f(k)+1}$ (d. h. die Umgebung, die sich für Sicherheitsparameter k verhält wie $\mathcal{Z}_l$ mit $l := f(k) + 1$). Die Umgebung $\mathcal{Z}_f$ simuliert nur Instanzen von $\mathcal{S}_i$ und ρ und schickt nie Nachrichten nach außen (denn die höchste auftretende Session-ID ist $sid = f(k)$, aber erst bei $sid = f(k)+1$ würde eine Nachricht nach außen geschickt werden), vgl. Abbildung 4.2c. Es existiert also ein zulässiger Simulator (welcher die Simulatoren $\mathcal{S}_1, \ldots, \mathcal{S}_{f(k)}$ simuliert, vgl. Abbildung 4.3a), so daß

$$\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_{f(k)+1}} = \mathrm{OUTPUT}_{f\cdot\rho,\mathcal{S}^*,\mathcal{Z}^*}. \tag{4.5}$$

Die Details der Definition dieses Simulators sind die folgenden (wieder empfehlen wir, diese Details beim ersten Lesen zu überspringen):

- $\mathcal{S}^*$ simuliert die Simulatoren $\mathcal{S}_1, \ldots, \mathcal{S}_{f(k)}$.
- Bei Aktivierung durch eine Nachricht (sid, m) auf Port $p \neq$ `clk`, wird der simulierte Simulator $\mathcal{S}_{sid}$ mit Nachricht m auf Port p aktiviert.
- Sendet der simulierte Simulator $\mathcal{S}_{sid}$ eine Nachricht m über Port $p \neq$ `env_clk`, so sendet $\mathcal{S}^*$ die Nachricht (sid, m) über Port p.
- Erhält $\mathcal{S}^*$ eine Nachricht m auf dem eingehenden Port `clk` (wird also in seiner Eigenschaft als Scheduler aktiviert), so wird $\mathcal{S}_{f(k)}$ mit Eingabe m auf Port `clk` aktiviert.[8]
- Sendet ein $\mathcal{S}_{sid}$ mit $sid > 1$ eine Nachricht m auf dem ausgehenden Port `env_clk`, so wird $\mathcal{S}_{sid-1}$ mit der Nachricht m auf Port `clk` aktiviert.
- Sendet $\mathcal{S}_1$ eine Nachricht m auf `env_clk`, so sendet $\mathcal{S}^*$ diese Nachricht m auf `env_clk`.

Wegen (4.4) und (4.5) brauchen wir, um (4.3) und damit das Theorem zu beweisen, nur zu zeigen, daß

$$\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_1} \approx \mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_{f(k)+1}}. \tag{4.6}$$

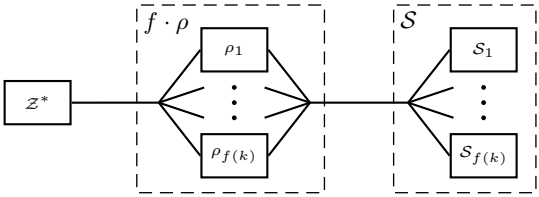
Zunächst folgt aus der Konstruktion von $\mathcal{Z}_l$, daß für alle $k \in \mathbb{N}$ und $z \in \Sigma^*$ gilt:

$$\mathrm{OUTPUT}_{\rho,\mathcal{S}_l,\mathcal{Z}_l} = \mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_{l+1}}. \tag{4.7}$$

(Auf der linken Seite werden l Instanzen des idealen Modells $\mathcal{S}_i$ und ρ von $\mathcal{Z}_l$ simuliert, und eine „externe“ Instanz von $\mathcal{S}_l$, ρ liegt vor. Auf der rechten Seite simuliert $\mathcal{Z}_{l+1}$ nun $l + 1$ Instanzen, dafür ist die „externe“ Instanz eine Instanz des realen Modells. Vgl. Abbildungen 4.2a und 4.3b. Man überzeuge sich, daß auch die `clk`-Nachrichten in beiden Fällen den gleichen Weg nehmen.)

[8] Hier erkennen wir den Grund, warum wir den `clk`-Port auf derart komplizierte Weise durch alle Angreifer und Simulatoren durchleiten. Andernfalls müßten wir nämlich hier entscheiden, welchen der $\mathcal{S}_{sid}$ wir aktivieren sollen. Da $\mathcal{S}^*$ aber nicht wissen kann, welche Instanz von ρ das Token verloren hat, ist dies nicht möglich.

(a)



(b)

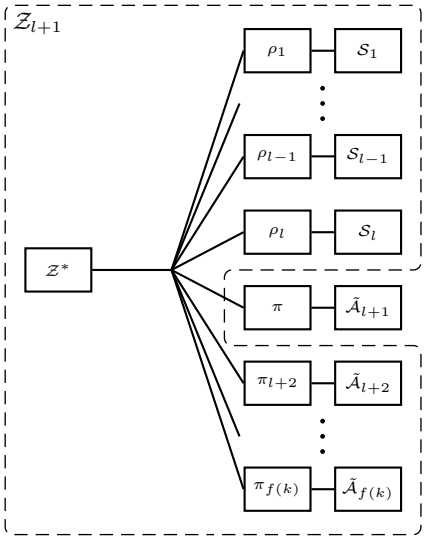


Abbildung 4.3.: Beweisschritte im nebenläufigen Kompositionstheorem.
(a) Konstruktion des Simulators $\mathcal{S}$. Die Verbindungen zwischen $\mathcal{Z}^*$ und den $\mathcal{S}_i$ sind der Übersichtlichkeit halber nicht eingezeichnet.
(b) Die Umgebung $\mathcal{Z}_{l+1}$ mit dem realen Protokoll.

Man ist nun versucht, direkt nutzen zu wollen, daß

$$\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_l} \approx \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}_l},$$

um dann induktiv (4.6) zu folgern. Wie aber bereits in Abschnitt 4.1 gezeigt, führt dieser Ansatz in die Irre, weil auf der rechten Seite von (4.6) f keine Konstante ist, sondern vom Sicherheitsparameter k abhängig.

Stattdessen begeben wir uns zunächst auf einen Umweg und definieren die Umgebung $\mathcal{Z}_\infty$. Es bezeichne $\mathcal{D}$ die Verteilung, die der Zahl $i \in \mathbb{N}$ die Wahrscheinlichkeit $1/i^2\gamma$ zuordnet mit $\gamma := \sum_{i=1}^{\infty} 1/i^2$. $\mathcal{Z}_\infty$ hat die gleichen Ports wie $\mathcal{Z}^*$ (und damit wie $\mathcal{Z}_l$), wählt bei der ersten Aktivierung ein L gemäß $\mathcal{D}$ und verhält sich dann wie $\mathcal{Z}_L$. Weiterhin sei $\mathcal{S}_\infty$ ein für $\mathcal{Z}_\infty$ geeigneter zulässiger Simulator, d. h.

$$\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_\infty} \approx \mathrm{OUTPUT}_{\rho,\mathcal{S}_\infty,\mathcal{Z}_\infty}.$$

Dann ist also

$$\mu(k) := \sup_{z\in\Sigma^*} \Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_\infty}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}_\infty,\mathcal{Z}_\infty}(k,z)\big)$$

vernachlässigbar.

Für $i \in \mathbb{N}$ bezeichne $\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_\infty}(k,z)|(L=l)$ die Verteilung der Zufallsvariable $\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_\infty}(k,z)$ unter der Bedingung, daß das von $\mathcal{Z}_\infty$ gewählte L den Wert l hat. Da $\mathcal{D}$ jedem l eine positive Wahrscheinlichkeit zuordnet, sind diese Verteilungen wohldefiniert. Offensichtlich ist

$$\begin{aligned} &\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_\infty}|(L=l) = \mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_l} \\ \text{und}\quad &\mathrm{OUTPUT}_{\rho,\mathcal{S}_\infty,\mathcal{Z}_\infty}|(L=l) = \mathrm{OUTPUT}_{\rho,\mathcal{S}_\infty,\mathcal{Z}_l} \end{aligned}$$

und mit Lemma 1.3 (vii) und der Tatsache, daß $L = l$ mit Wahrscheinlichkeit $1/l^2\gamma$ eintritt, folgt außerdem

$$\begin{aligned} &\Delta\Big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_\infty}(k,z)|(L=l);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}_\infty,\mathcal{Z}_\infty}(k,z)|(L=l)\Big) \\ &\qquad \le l^2\gamma\,\Delta\Big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_\infty}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}_\infty,\mathcal{Z}_\infty}(k,z)\Big) \end{aligned}$$

Damit erhalten wir

$$\mu_l(k) := \sup_{z\in\Sigma^*} \Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_l}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}_\infty,\mathcal{Z}_l}(k,z)\big) \le \frac{l^2\mu(k)}{\gamma}.$$

Da $\mathcal{S}_l$ nach Wahl ein fast optimaler Simulator für $\mathcal{Z}_l$ ist, folgt

$$\begin{aligned} &\Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_l}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}_l,\mathcal{Z}_l}(k,z)\big) \\ &\qquad \le \mu_l(k) + 2^{-k} \le l^2\gamma\mu(k) + 2^{-k}. \end{aligned} \tag{4.8}$$

Für jedes k gilt dann induktiv unter Benutzung von (4.7), (4.8) und der Tatsache, daß Δ eine Metrik ist (Lemma 1.3 (ii)):

$$\Delta\Big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_1}(k,z);\ \mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}_{f(k)+1}}(k,z)\Big) \leq \sum_{l=1}^{f(k)}\Big(l^2\gamma\mu(k)+2^{-k}\Big) =: \mu^*(k).$$

Da μ vernachlässigbar ist, ist es auch μ^*. Es folgt (4.6) und damit ist das Theorem für den Fall der statistischen speziellen Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung gezeigt.

Der Fall ohne Auxiliary input wird genauso bewiesen, lediglich wird nicht mehr über alle $z \in \Sigma^*$ sondern nur über $z = \lambda$ quantifiziert. Bei Sicherheit bzgl. der Sicht der Umgebung ist die Konstruktion im wesentlichen die gleiche, man muß nur beachten, daß die Sicht von $\mathcal{Z}^*$ deterministisch aus der Sicht von $\mathcal{Z}_l$ bzw. $\mathcal{Z}_\infty$ berechnet werden kann. □

Aus Theorem 4.1 erhalten wir dann (wie bei Lemma 2.29):

Korollar 4.4 (Allgemeines Kompositionstheorem für statistische spezielle Sicherheit)
Es seien π und ρ prinzipiell einbettbare Protokolle.

Wenn π so sicher wie ρ ist bezüglich statistischer spezieller Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung, dann ist π so sicher wie ρ bezüglich polynomiell-beschränkter allgemeiner Komponierbarkeit mit Auxiliary input.

Das gleiche gilt im Falle ohne Auxiliary input.

4.3. Statistische Sicherheit mit Auxiliary input

Wenn wir statistische Sicherheit mit Auxiliary input betrachten, dann ergibt sich auch eine andere Möglichkeit zu zeigen, daß spezielle Sicherheit für allgemeine Komponierbarkeit hinreichend ist. Denn in diesem Falle fallen allgemeine und spezielle Sicherheit zusammen.

Der Grund liegt darin, daß der Auxiliary input selbst bei der speziellen Sicherheit *nach* dem Simulator gewählt wird. Wenn also zu jedem Simulator eine Umgebung existiert, die unterscheidet, so wählen wir einfach für jeden Simulator den Auxiliary input, der diese unterscheidende Umgebung beschreibt. Dann konstruieren wir eine universelle Umgebung, die die durch ihren Auxiliary input beschriebene Umgebung simuliert, und somit für jeden Simulator zu unterscheiden in der Lage ist. So folgt aus spezieller Sicherheit allgemeine Sicherheit.

Ein Problem tritt hier allerdings auf, eine Maschine hat nämlich nicht notwendigerweise eine endliche Beschreibung. Um dieses Problem zu lösen, zeigen wir, daß wir jede Umgebung (gegeben einen festen Simulator) durch eine leicht veränderte Umgebung ersetzen können, so daß (i) der dadurch entstehende Fehler vernachlässigbar ist und (ii) die resultierende Umgebung eine endliche Beschreibung hat. Da diese veränderte Umgebung immer noch unterscheidet, können wir o. B. d. A. eine solche endliche Umgebung annehmen, und obiges Argument funktioniert.

Diese Überlegungen erlauben es, den folgenden Satz zu formulieren:

Satz 4.5 (Allgemeine und spezielle statistische Sicherheit fallen zusammen bei Auxiliary input)
Es seien π und ρ Protokolle. Es ist π genau dann so sicher wie ρ bezüglich *allgemeiner* statistischer Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung, wenn π so sicher wie ρ bezüglich *spezieller* statistischer Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung ist.

Das gleiche gilt für Sicherheit bzgl. der *Sicht* der Umgebung.

Beweis: Da allgemeine Sicherheit spezielle Sicherheit impliziert (Lemma A.4), genügt es zu zeigen, daß aus spezieller Sicherheit allgemeine folgt. Da weiterhin im Falle allgemeiner statistischer Sicherheit die Fälle mit und ohne Auxiliary input zusammenfallen (Lemma A.2), reicht es zu zeigen, daß aus spezieller Sicherheit *mit* Auxiliary input allgemeine Sicherheit *ohne* Auxiliary input folgt.

Wir nehmen also an, π sei *nicht* so sicher wie ρ bezüglich allgemeiner Sicherheit ohne Auxiliary input bzgl. der Ausgabe/Sicht der Umgebung (wir betrachten die beiden Fälle gleichzeitig).

Dann existiert ein zulässiger Angreifer $\mathcal{A}$, eine Funktion $\mathcal{Z}(\mathcal{S})$, die jedem Simulator $\mathcal{S}$ eine Umgebung $\mathcal{Z}$ zuordnet, so daß für jeden Simulator $\mathcal{S}$ gilt:

$$\left\{\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}(\mathcal{S})}(k)\right\}_k \not\approx \left\{\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}(\mathcal{S})}(k)\right\}_k \tag{4.9}$$

bzw.

$$\left\{\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}(\mathcal{S})}(k)\right\}_k \not\approx \left\{\mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}(\mathcal{S})}(k)\right\}_k. \tag{4.10}$$

(Je nach dem, ob Sicherheit bzgl. der Ausgabe oder der Sicht der Umgebung betrachtet wird.) Hierbei bezeichnet $\approx$ die statistische Ununterscheidbarkeit.

Es sei $\mathcal{Z}^n(\mathcal{S})$ die Umgebung, die sich wie $\mathcal{Z}(\mathcal{S})$ verhält, aber nach n Aktivierungen terminiert.

In der diskreten Topologie[9] konvergiert $\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}^n(\mathcal{S})}(k)$ für festes k fast sicher gegen $\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}(\mathcal{S})}(k)$ (in n), da die Ausgabe entweder nach einer

[9] Eine Folge konvergiert in der diskreten Topologie genau dann, wenn sie stationär wird.

endlichen Anzahl von Aktivierungen feststeht, oder nie eine Ausgabe erfolgt. Im letzteren Fall ist für jedes n die Ausgabe $\perp$, und Konvergenz liegt auch hier vor. Nach Lemma 1.3 (x) konvergiert also für $n \to \infty$ der statistische Abstand

$$\Delta\big(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}^n(\mathcal{S})}(k);\ \mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}(\mathcal{S})}(k)\big)$$

gegen 0. Gleiches gilt für $\Delta\big(\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}^n(\mathcal{S})}(k);\ \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}(\mathcal{S})}(k)\big)$. Wählen wir also eine hinreichend große Funktion $n = n(k)$, so folgt mit (4.9)

$$\big\{\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}^n(\mathcal{S})}(k)\big\}_k \not\approx \big\{\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}^n(\mathcal{S})}(k)\big\}_k.$$

Weiterhin ist $\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}^n(\mathcal{S})}(k)$ das Präfix der Länge n von $\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}(\mathcal{S})}(k)$. Analog $\mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}^n(\mathcal{S})}(k)$. Nach Lemma 1.3 (ix) konvergiert also

$$\begin{aligned}&\Delta\big(\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}^n(\mathcal{S})}(k);\ \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}^n(\mathcal{S})}(k)\big)\\ &\overset{n\to\infty}{\longrightarrow} \Delta\big(\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}(\mathcal{S})}(k);\ \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}(\mathcal{S})}(k)\big).\end{aligned}$$

Aus (4.10) folgt damit für eine hinreichend große Funktion $n = n(k)$

$$\big\{\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}^n(\mathcal{S})}(k)\big\}_k \not\approx \big\{\mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}^n(\mathcal{S})}(k)\big\}_k.$$

Im folgenden sei n fest die oben gewählte Funktion.

Es bezeichne M_k die Menge der möglichen Sichten von $\mathcal{Z}^n(\mathcal{S})$, die in einem Protokollauf von $\pi, \mathcal{A}, \mathcal{Z}^n(\mathcal{S})$ oder in einem Protokollauf von $\rho, \mathcal{S}, \mathcal{Z}^n(\mathcal{S})$ vorkommen können. Da $\mathcal{Z}^n(\mathcal{S})$ nach endlich vielen Aktivierungen terminiert, ist jede Sicht von $\mathcal{Z}^n(\mathcal{S})$ endlich, und somit M_k abzählbar. Also existiert eine endliche Menge $M'_k \subseteq M_k$, so daß

$$\begin{aligned}&P\big(\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}^n(\mathcal{S})}(k) \in M'_k\big) \geq 1 - 2^{-k}\\ \text{und}\quad &P\big(\mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}^n(\mathcal{S})}(k) \in M'_k\big) \geq 1 - 2^{-k}.\end{aligned} \tag{4.11}$$

Es sei $\mathcal{Z}'(\mathcal{S})$ die Maschine, die sich wie $\mathcal{Z}^n(\mathcal{S})$ verhält, mit der Änderung, daß $\mathcal{Z}'(\mathcal{S})$ terminiert, wenn ihre bisherige Sicht nicht Präfix einer Sicht aus M'_k ist. Da dies höchstens mit Wahrscheinlichkeit 2^{-k} geschieht, gilt weiterhin

$$\big\{\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}'(\mathcal{S})}(k)\big\}_k \not\approx \big\{\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}'(\mathcal{S})}(k)\big\}_k$$

bzw.

$$\big\{\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}'(\mathcal{S})}(k)\big\}_k \not\approx \big\{\mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}'(\mathcal{S})}(k)\big\}_k.$$

Da die Maschine $\mathcal{Z}'(\mathcal{S})$ nur bei endlich vielen Sichten nicht abbricht, kann die Zustandsübergangsfunktion $\mathcal{Z}'(\mathcal{S})$ durch die Angabe von endlich vielen Wahrscheinlichkeitsverteilungen beschrieben werden, nämlich die Verteilungen μ_v der zu sendenden Nachrichten für die jeweilige (partielle) Sicht v.

Wir können nun jede der Verteilungen μ_v durch eine Verteilung μ'_v ersetzen mit $\Delta(\mu_v, \mu'_v) \leq 2^{-k}/n(k)$, so daß μ'_v endlichen Träger und nur rationale Wahrscheinlichkeiten hat (durch Weglassen unwahrscheinlicher Ereignisse und Runden der Wahrscheinlichkeiten). Dann ist μ'_v durch eine endliche Zeichenkette beschreibbar. Es sei $\mathcal{Z}''(\mathcal{S})$ die Maschine, die die Verteilungen μ'_v statt μ_v verwendet. Damit ist (für gegebenen Sicherheitsparameter k) auch die Zustandsübergangsfunktion von $\mathcal{Z}''(\mathcal{S})$ endlich beschreibbar.

Da $\mathcal{Z}''(\mathcal{S})$ höchstens n-mal aktiviert wird, und in jeder Aktivierung nur ein Fehler von höchstens $2^{-k}/n$ auftritt, gilt:

$$\Delta\big(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}'(\mathcal{S})}(k);\ \mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}''(\mathcal{S})}(k)\big) \leq 2^{-k}$$

und analoges für $\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}''(\mathcal{S})}(k)$, $\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}''(\mathcal{S})}(k)$, $\mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}''(\mathcal{S})}(k)$. Damit folgt

$$\big\{\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}''(\mathcal{S})}(k)\big\}_k \not\approx \big\{\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}''(\mathcal{S})}(k)\big\}_k$$

bzw.

$$\big\{\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}''(\mathcal{S})}(k)\big\}_k \not\approx \big\{\mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}''(\mathcal{S})}(k)\big\}_k.$$

Es sei $z_k(\mathcal{S})$ für gegebenes k die Beschreibung der Zustandsübergangsfunktion von $\mathcal{Z}''(\mathcal{S})$ für Sicherheitsparameter k in einer willkürlichen, aber festen Kodierung (wie oben gesehen, ist die Beschreibung endlich). Es sei weiter $\mathcal{Z}^u$ die Umgebung, die sich bei Auxiliary input $z_k(\mathcal{S})$ wie die Umgebung $\mathcal{Z}''(\mathcal{S})$ verhält.

Damit gilt also für jeden Simulator und für $z_k := z_k(\mathcal{S})$ als Auxiliary input:

$$\big\{\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}^u}(k, z_k)\big\}_k \not\approx \big\{\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}^u}(k, z_k)\big\}_k$$

bzw.

$$\big\{\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}^u}(k, z_k)\big\}_k \not\approx \big\{\mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}^u}(k, z_k)\big\}_k.$$

Also ist π nicht so sicher wie ρ bezüglich spezieller statistischer Sicherheit mit Auxiliary input bzgl. der Ausgabe bzw. der Sicht der Umgebung. □

Man beachte, daß Satz 4.5 nicht auf den Fall ohne Auxiliary input verallgemeinert, wir geben in Abschnitt 5.2 ein Gegenbeispiel an. Auch auf den Fall, daß ein Auxiliary input existiert, dessen Länge aber beschränkt ist (durch eine dem Simulator bekannte Funktion), läßt sich das Gegenbeispiel aus Abschnitt 5.2 verallgemeinern.

5. Laufzeittrennungen und Time-lock puzzles

5.1. Wettkämpfe zwischen Umgebung und Simulator

Im diesem Kapitel wollen wir zeigen, daß – zumindest im Falle der statistischen und der komplexitätstheoretischen Sicherheit – allgemeine und spezielle Sicherheit verschieden sind, daß es also tatsächlich einen Unterschied macht, ob die Umgebung abhängig vom Simulator gewählt wird, oder der Simulator abhängig von der Umgebung.

Da die in den folgenden Abschnitten vorgestellten Beweistechniken alle auf derselben Grundidee fußen, wollen wir diese zunächst auf intuitiver Ebene skizzieren, bevor wir die einzelnen Instantiierungen im Detail vorstellen.

Hierzu rufen wir uns zunächst den Unterschied zwischen spezieller und allgemeiner Sicherheit nochmals ins Gedächtnis. Dieser besteht darin, daß bei der speziellen Sicherheit für jede Umgebung ein Simulator existieren muß, so daß die Umgebung nicht zwischen dem realen (in dem der Simulator nicht vorkommt) und dem idealen Modell (mit Simulator) unterscheiden kann. Der Simulator wird hier also in Abhängigkeit von der Umgebung gewählt. Im Gegensatz dazu verlangen wir bei der allgemeinen Sicherheit, daß ein Simulator existiert, so daß keine Umgebung zwischen realem und idealem Modell unterscheiden kann. Hier wird also die Umgebung in Abhängigkeit vom Simulators gewählt.

Wir können uns die Sicherheitsdefinitionen also als Spiel zwischen Umgebung und Simulator vorstellen.[1] Die Umgebung gewinnt, wenn sie zwischen realem und idealem Modell unterscheiden kann; andernfalls gewinnt der Simulator. Hierbei genießt die Umgebung einen Vorzug: bereits wenn die Umgebung mit einer kleinen, aber nicht vernachlässigbaren Wahrscheinlichkeit unterscheiden kann, hat sie das Spiel gewonnen. Ein Protokoll gilt als sicher, wenn der Simulator gewinnt.

In dieser Sichtweise liegt der Unterschied zwischen spezieller und allgemeiner Sicherheit darin, daß im ersten Fall zunächst die Umgebung ihre Strategie (d. h. ihr Programm) festlegt, und dann der Simulator *unter Kenntnis der Strategie der Umgebung* seine Strategie festlegt. Umgekehrt darf bei der allgemeinen Sicherheit die Umgebung die Wahl ihrer Strategie von der des Simulators abhängig machen.

[1] Den Angreifer können wir zunächst außer acht lassen, da wir hier o. B. d. A. den Dummy-Angreifer annehmen können.

Um also ein trennendes Beispiel für spezielle und allgemeine Sicherheit zu finden, müssen wir zunächst ein Spiel konstruieren, in dem immer der gewinnt, der seine Strategie zuletzt festlegen darf. Dieses Spiel muß natürlich von der Form sein wie es in der Definition der speziellen und allgemeinen Sicherheit vorkommt. In anderen Worten, wir müssen zwei Protokolle, ein reales und ein ideales, finden, so daß unser Spiel darin besteht, daß die Umgebung gewinnt, wenn sie unterscheiden kann, und der Simulator, wenn er dies verhindern kann.

Um eine Vorstellung zu vermitteln, wie die unseren trennenden Beispielen zugrundeliegenden Spiele aussehen, betrachten wie zunächst folgende (stark vereinfachte) Situation: Wie stellen uns vor, jeder Maschine M (insbesondere jeder Umgebung und jedem Simulator) sei eine natürliche Zahl s_M, ihre Stärke zugeordnet. Weiterhin nehmen wir an, wir könnten ein Protokoll ρ konstruieren, welches feststellen kann, ob die Umgebung $\mathcal{Z}$ oder der Simulator $\mathcal{S}$ „stärker“ ist (also ob $s_\mathcal{Z} > s_\mathcal{S}$ oder $s_\mathcal{Z} < s_\mathcal{S}$). Falls die Umgebung stärker ist, soll das Protokoll ρ die Nachricht „ich bin das ideale Protokoll“ an die Umgebung schicken, ansonsten die Nachricht „ich bin das reale Protokoll“. Das Protokoll π hingegen sage immer „ich bin das reale Protokoll“. Ist nun die Umgebung „stärker“ als der Simulator, so erfährt die Umgebung, ob sie mit π oder mit ρ kommuniziert (und gewinnt somit das Spiel). Ist hingegen der Simulator „stärker“, so wird sowohl π als auch ρ immer ausgeben, daß es das reale Protokoll ist (und damit gewinnt der Simulator).

Wird nun zunächst die Umgebung $\mathcal{Z}$ mit Stärke $s_\mathcal{Z}$ gewählt, so kann man abhängig davon einen Simulator mit Stärke $s_\mathcal{S} := s_\mathcal{Z} + 1$ wählen, der somit stärker als die Umgebung ist und gewinnt. Somit ist π so sicher wie ρ bezüglich *spezieller* Sicherheit. Wählt man aber umgekehrt die Umgebung $\mathcal{Z}$ nach dem Simulator, so kann die Umgebung die Stärke $s_\mathcal{Z} := s_\mathcal{S} + 1$ haben, und damit das Spiel gewinnen. Also ist π *nicht* so sicher wie ρ bezüglich *allgemeiner* Sicherheit, π und ρ trennen also die beiden Begriffe.

Unglücklicherweise läßt sich obige Intuition nicht direkt in eine konkrete Konstruktion umsetzen, denn wie genau sollte die Stärke einer Maschine definiert sein, und wie soll ein Protokoll unterscheiden können, welche Maschine die stärkere ist? Die Antwort auf diese Fragen hängt von den Details des jeweiligen Sicherheitsbegriffs ab, und wird in den folgenden Abschnitten erörtert.

5.2. Statistische Sicherheit

In diesem Abschnitt untersuchen wir, wie man die im vorangegangenen Abschnitt skizzierte Beweisidee im Fall der statistischen Sicherheit umsetzt, also auf welche Weise eine unbeschränkte Maschine „ihre Stärke beweisen“ kann. Auf den ersten Blick mag dies paradox klingen, da eine unbeschränkte Maschine intuitiv „unendliche Stärke“ haben müßte. Es zeigt sich aber, daß – vorausgesetzt

die Maschine erhält keinen von der anderen Maschine abhängigen Auxiliary input – wir dennoch Maschinen vergleichen können.

Das Spiel, das Umgebung und Simulator spielen sollen, ist in diesem Szenario das folgende: Sowohl Umgebung $\mathcal{Z}$ als auch Simulator $\mathcal{S}$ wählen je eine (beliebig große) natürliche Zahl $s_{\mathcal{Z}}$ bzw. $s_{\mathcal{S}}$. Die Umgebung gewinnt nun, wenn ihre Zahl größer ist, ansonsten gewinnt der Simulator. Stellen wir uns zunächst vor, die Umgebung und der Simulator müßten diese Zahl deterministisch wählen (sprich: die Zahl wäre fest in ihrem Programm kodiert). Dann hat offensichtlich die zweitgewählte Maschine den Vorteil, weil sie immer eine um 1 größere Zahl verwenden kann.

Doch tatsächlich muß die Zahl, die eine Maschine M wählt, nicht deterministisch von ihrem Programm abhängen. Vielmehr liegt lediglich eine Verteilung S_M auf den natürlichen Zahlen fest. Doch auch dies ist hinreichend, damit die zweitgewählte Maschine das Spiel gewinnen kann: Zu jeder Zufallsvariable S auf den natürlichen Zahlen und jedem Sicherheitsparameter k existiert eine Zahl n_S, so daß $P(S \geq n_S) \leq 2^{-k}$ (diese Zahl n_S ist gerade das $(1-2^{-k})$-Quantil von S). Liegt also eine Maschine M fest, die die Zahl s_M gemäß der Verteilung S_M wählt, so kann eine andere Maschine N mit überwältigender Wahrscheinlichkeit das Spiel gewinnen, indem sie $s_N := n_{S_M}$ wählt.

Wir haben also ein Spiel im Sinne des vorangegangenen Abschnitts gefunden, in dem die „Stärke“ einer Maschine durch die Größe der ausgegebenen Zahlen ausgedrückt wird, und in dem die zuletzt gewählte Maschine immer einen großen Vorteil hat.[2]

Dieses Spiel läßt sich sehr einfach in ein konkretes Gegenbeispiel bestehend aus zwei Protokollen π und ρ umwandeln. Das reale Protokoll π sagt der Umgebung immer (egal welche Zahl die Umgebung eingibt), daß es das reale Protokoll sei, während das ideale Protokoll sowohl von der Umgebung als auch vom Simulator eine Zahl erwartet, und der Umgebung dann wahrheitsgemäß mitteilt, daß es das ideale Protokoll ist, wenn die Umgebung die größere Zahl eingibt. Ansonsten gibt ρ wie π aus, das reale Protokoll zu sein. Wird nun der Simulator zuletzt gewählt, so kann er durch Eingabe einer hinreichend großen Zahl immer erreichen, daß beide Protokolle das gleiche ausgeben (mit überwältigender Wahrscheinlichkeit), π ist also so sicher wie ρ bezüglich *spezieller* Sicherheit. Wird aber die Umgebung als zweites gewählt, so kann sie die größere Zahl eingeben, und die Protokolle unterscheiden.

Wir können nun das genaue Ergebnis und dessen Beweis angeben:

[2] Dieses Spiel ist auch das vermutlich einfachste Beispiel für ein Spiel ohne Nash-Equilibrium. Dies bringt die Konstruktion aus diesem Abschnitt in direkten Zusammenhang mit den Techniken aus Kapitel 7, denn der Grund, daß Korollar 7.20 nicht auf das hier vorgestellte Gegenbeispiel anwendbar ist, liegt gerade an der Nichtexistenz eines solchen Nash-Equilibriums.

Satz 5.1 (Trennung von statistischer allgemeiner und spezieller Sicherheit)
Es existieren Protokolle π und ρ, für die folgendes gilt:

- Das Protokoll π ist so sicher wie ρ bezüglich *spezieller* statistischer Sicherheit *ohne* Auxiliary input sowohl bzgl. der Sicht als auch bzgl. der Ausgabe der Umgebung.
- Das Protokoll π ist *nicht* so sicher wie ρ bezüglich *allgemeiner* statistischer Sicherheit *ohne* Auxiliary input weder bzgl. der Sicht noch bzgl. der Ausgabe der Umgebung.

Beweis: Das Protokoll π besteht aus einer Maschine M_{real}, und das Protokoll ρ aus einer Maschine M_{ideal}.

Für $x \in \{real, ideal\}$ sei M_x die Maschine, die wie folgt definiert ist (vgl. Abbildung 5.1a): M_x hat die eingehenden Ports `in_number`, `adv_number` und den ausgehenden Port `out_identity` (das Protokoll π hat also eine eingehende Verbindung namens `in_number` von der Umgebung, eine eingehende Verbindung namens `adv_number` vom Angreifer/Simulator, und eine ausgehende Verbindung namens `out_identity` zur Umgebung.)

Bei der ersten Aktivierung über den Port `in_number` setzt M_x die Variable $s_{\mathcal{Z}} := |m|$, wobei m die über den Port `in_number` erhaltene Nachricht ist. (Die Zahl $s_{\mathcal{Z}}$ wird also unär übertragen.)

Bei der ersten Aktivierung über den Port `adv_number` setzt M_x die Variable $s_{\mathcal{S}} := |m|$, wobei m die über den Port `adv_number` erhaltene Nachricht ist.

Sobald sowohl $s_{\mathcal{Z}}$ als auch $s_{\mathcal{S}}$ gesetzt sind, testet M_x, ob $s_{\mathcal{Z}} > s_{\mathcal{S}}$. Wenn ja, wird x über `out_identity` geschickt. Ansonsten wird *real* über `out_identity` geschickt. (Man beachte, daß also M_{real} in beiden Fällen *real* sendet.)

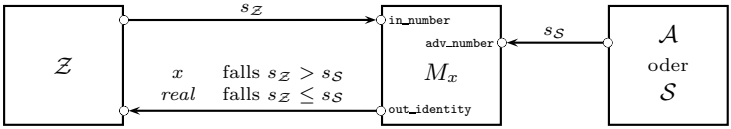
Wir zeigen nun zunächst, daß π so sicher wie ρ bezüglich *spezieller* statistischer Sicherheit ohne Auxiliary input bzgl. der Sicht und der Ausgabe der Umgebung ist.

Es seien also eine zulässige Umgebung $\mathcal{Z}$ und ein zulässiger Angreifer $\mathcal{A}$ gegeben. In einer Ausführung von π, $\mathcal{Z}$ und $\mathcal{A}$ mit Sicherheitsparameter k bezeichne die Zufallsvariable $S^k_{\mathcal{Z}}$ die von M_{real} auf Port `in_number` erhaltene Zahl $s_{\mathcal{Z}}$. (Und $S^k_{\mathcal{Z}} := 0$, falls keine solche Zahl empfangen wurde; hat $\mathcal{Z}$ keinen ausgehenden Port `in_number`, so ist immer $S^k_{\mathcal{Z}} = 0$.)

Für jeden Sicherheitsparameter $k \in \mathbb{N}$ sei $n_k \in \mathbb{N}$ die kleinste Zahl, so daß $P(S^k_{\mathcal{Z}} \geq n_k) \leq 2^{-k}$. Eine solche Zahl muß existieren, da $P(S^k_{\mathcal{Z}} \geq n) \to 0$ für $n \to \infty$.

Wir definieren nun den Simulator $\mathcal{S}$ wie folgt: $\mathcal{S}$ hat die gleichen Ports wie $\mathcal{A}$. Weiterhin verhält sich $\mathcal{S}$ wie $\mathcal{A}$, mit der folgenden Ausnahme: Wann immer $\mathcal{A}$

(a)



(b)

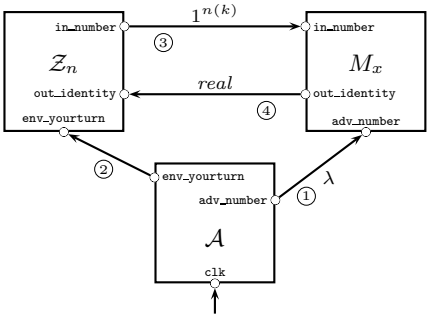


Abbildung 5.1.: Trennung von allgemeiner und spezieller statistischer Sicherheit.
(a) Die Maschine M_x im Zusammenspiel mit der Umgebung $\mathcal{Z}$.
(b) Angreifer $\mathcal{A}$ und Umgebung $\mathcal{Z}_n$ im Zusammenspiel mit dem realen Protokoll $\pi = \{M_{real}\}$.

eine Nachricht m über den ausgehenden Port `adv_number` schicken würde (falls ein solcher Port vorhanden ist), schickt $\mathcal{S}$ stattdessen 1^{n_k}, also einen String der Länge n_k.

Offensichtlich ist $\mathcal{S}$ ein zulässiger Simulator.

Da das Verhalten von M_{real} unabhängig vom Wert von $s_{\mathcal{S}}$ ist (M_{real} sendet in jedem Fall *real* an die Umgebung), können wir $\mathcal{A}$ durch $\mathcal{S}$ ersetzen, ohne daß sich die Sicht der Umgebung $\mathcal{Z}$ ändert, in Formeln:

$$\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k) = \mathrm{VIEW}_{\pi,\mathcal{S},\mathcal{Z}}(k)$$

Insbesondere hat $s_{\mathcal{Z}}$ in beiden Protokolläufen die gleiche Verteilung. Daher ist die Wahrscheinlichkeit, daß $s_{\mathcal{Z}} > n_k = s_{\mathcal{S}}$ auch in einem Protokollauf von π mit $\mathcal{S}$ und $\mathcal{Z}$ durch 2^{-k} beschränkt. Da M_{real} und M_{ideal} nur im Falle $s_{\mathcal{Z}} > s_{\mathcal{S}}$ verschiedenes Verhalten zeigen, können wir M_{real} durch M_{ideal} ersetzen und erhalten

$$\Delta\big(\mathrm{VIEW}_{\pi,\mathcal{S},\mathcal{Z}}(k);\ \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}}(k)\big) \leq 2^{-k}.$$

Somit sind

$$\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k) \qquad \text{und} \qquad \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}}(k)$$

statistisch ununterscheidbar und folglich auch

$$\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k) \qquad \text{und} \qquad \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k).$$

Also ist π so sicher wie ρ bezüglich *spezieller* statistischer Sicherheit ohne Auxiliary input bzgl. der Sicht und der Ausgabe der Umgebung.

Nun zeigen wir, daß π *nicht* so sicher wie ρ ist bezüglich *allgemeiner* statistischer Sicherheit ohne Auxiliary input bzgl. der Sicht und der Ausgabe der Umgebung.

Dazu geben wir einen zulässigen Angreifer $\mathcal{A}$ an, und für jeden Simulator $\mathcal{S}$ eine Umgebung $\mathcal{Z}$, so daß in einem Protokollauf von π mit $\mathcal{A}$ und $\mathcal{Z}$ mit Wahrscheinlichkeit 1 die Nachricht *real* von π zu $\mathcal{Z}$ geschickt wird, wohingegen in einem Protokollauf von ρ mit $\mathcal{S}$ und $\mathcal{Z}$ die Nachricht *real* nur mit vernachlässigbarer Wahrscheinlichkeit gesandt wird.

Wir beginnen mit dem Angreifer $\mathcal{A}$ (vgl. Abbildung 5.1b). Es sei $\mathcal{A}$ der Angreifer mit dem eingehenden Port `clk` (über den er in seiner Eigenschaft als Scheduler aktiviert wird) und den folgenden ausgehenden Ports: dem Angreiferport `adv_number` (über den er den Wert $s_{\mathcal{S}}$ an die Maschine M_{real} schickt) und dem Umgebungsport `env_yourturn` (über den er die Umgebung aktivieren kann).

In seiner ersten Aktivierung schickt $\mathcal{A}$ die Nachricht λ über den ausgehenden Port `adv_number` (an M_{real}). In seiner zweiten Aktivierung schickt $\mathcal{A}$ dann die Nachricht λ über `env_yourturn` (und aktiviert damit die Umgebung). Offensichtlich ist dieser Angreifer zulässig.

Es sei $n : \mathbb{N} \to \mathbb{N}$ eine Funktion. Wir beschreiben nun die Umgebung $\mathcal{Z}_n$ (vgl. Abbildung 5.1b). Diese hat die eingehenden Ports `out_identity` (über den sie von M_x die Nachricht `real` oder `ideal` erhalten kann) und `env_yourturn` (über den sie vom Angreifer aktiviert werden kann). Weiter hat die Umgebung $\mathcal{Z}_n$ die ausgehenden Ports `in_number` (über den sie $s_\mathcal{Z}$ an M_x sendet) und `output` (über den sie ihre finale Ausgabe OUTPUT liefert).

Bei Aktivierung über den Port `env_yourturn` sendet $\mathcal{Z}_n$ die Nachricht $1^{n(k)}$ über `in_number` an M_x. Bei Aktivierung über den Port `out_identity` mit Nachricht m gibt $\mathcal{Z}_n$ m über den Port `output` aus.

In einem Protokollauf von π mit $\mathcal{Z}_n$ und $\mathcal{A}$ wird also mit Wahrscheinlichkeit 1 das folgende geschehen (vgl. Abbildung 5.1b): $\mathcal{A}$ wird über `clk` aktiviert und sendet λ an M_{real}. Da M_{real} keine Nachricht sendet, wird $\mathcal{A}$ nochmals über `clk` aktiviert und aktiviert $\mathcal{Z}_n$ über `env_yourturn`. Darauf schickt $\mathcal{Z}_n$ die Nachricht $1^{n(k)}$ an M_{real}. Da M_{real} nun sowohl $s_\mathcal{Z} = n(k)$ als auch $s_\mathcal{S} = 0$ erhalten hat, sendet M_{real} die Nachricht *real* über `out_identity` and $\mathcal{Z}_n$, welche daraufhin *real* ausgibt. Also ist

$$P\big(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}_n}(k) = \mathit{real}\big) = 1.$$

Nun sei ein zulässiger Simulator $\mathcal{S}$ gegeben. In einer Ausführung von ρ, $\mathcal{Z}_n$ und $\mathcal{S}$ mit Sicherheitsparameter k bezeichne die Zufallsvariable $S_\mathcal{S}^{k,n}$ die von M_{ideal} auf Port `adv_number` erhaltene Zahl $s_\mathcal{S}$. (Und $S_\mathcal{S}^{k,n} := 0$, falls keine solche Zahl empfangen wurde; hat $\mathcal{S}$ keinen ausgehenden Port `adv_number`, so ist immer $S_\mathcal{S}^{k,n} = 0$.)

Nach Konstruktion von M_{ideal} hängt $s_\mathcal{S}$ nicht vom Wert von $s_\mathcal{Z}$ ab (denn selbst wenn $s_\mathcal{Z}$ zuerst gesandt wird, so wird M_{ideal} erst dann eine Entscheidung treffen, die abhängig vom Wert von $s_\mathcal{Z}$ ist, wenn $s_\mathcal{S}$ empfangen wurde). Somit ist $S_\mathcal{S}^{k,0} = S_\mathcal{S}^{k,n}$ für jede Funktion n.

Es sei nun $n(k) \in \mathbb{N}$ die kleinste Zahl mit $P(S_\mathcal{S}^{k,0} \geq n(k)) \leq 2^{-k}$. Dann ist auch $P(S_\mathcal{S}^{k,n} \geq n(k)) \leq 2^{-k}$. Da aber $s_\mathcal{Z}$ nur den Wert $n(k)$ annehmen kann, und M_{ideal} nur dann *real* über `out_identity` an $\mathcal{Z}_n$ sendet, wenn $s_\mathcal{S}^{k,n} = s_\mathcal{S} \geq s_\mathcal{Z} = n(k)$, folgt

$$P\big(\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}_n}(k) = \mathit{real}\big) \leq 2^{-k}$$

Mit $\mathcal{Z}(\mathcal{S}) := \mathcal{Z}_n$ ist also für jeden zulässigen Simulator

$$\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}(\mathcal{S})} \not\approx \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}(\mathcal{S})}$$

und, da die Ausgabe deterministisch aus der Sicht folgt, auch

$$\mathrm{VIEW}_{\pi,\mathcal{A},\mathcal{Z}(\mathcal{S})} \not\approx \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}(\mathcal{S})}$$

Somit ist π *nicht* so sicher wie ρ bezüglich *allgemeiner* statistischer Sicherheit ohne Auxiliary input bzgl. der Sicht und der Ausgabe der Umgebung. □

Man beachte, daß das eben vorgestellte trennende Beispiel nicht funktioniert, wenn wir Sicherheit mit Auxiliary input (der ja vom Simulator abhängt) betrachten. Denn die Umgebung kann in diesem Fall selbst bei spezieller Sicherheit die Zahl $s_{\mathcal{Z}}$ über den Auxiliary input erhalten und somit – ähnlich wie im zweiten Teil des Beweises – mit überwältigender Wahrscheinlichkeit den Simulator überbieten. Dies ist nicht weiter überraschend, da wir in Abschnitt 4.3 gezeigt haben, daß in diesem Fall spezielle und allgemeine Sicherheit zusammenfallen.

Falls aber eine (dem Simulator bekannte) obere Schranke für die Länge des Auxiliary inputs existiert (z. B. wenn der Auxiliary input polynomiell-beschränkt ist), kann man den Beweis anpassen: Der Simulator wählt $s_{\mathcal{S}}$ so groß, daß für jeden der *endlich vielen* möglichen Auxiliary inputs mit überwältigender Wahrscheinlichkeit $s_{\mathcal{S}} \geq s_{\mathcal{Z}}$ gilt.

5.3. Komplexitätstheoretische Sicherheit

Wir untersuchen nun die nächste Anwendung der in Abschnitt 5.1 vorgestellten Beweismethode. Im Falle der komplexitätstheoretischen Sicherheit macht es durchaus Sinn, von der Stärke einer Maschine zu reden, denn einer Maschine läßt sich immer ein ihre Laufzeit beschränkendes Polynom zuordnen. Auch ist es einfach ein Spiel zu entwerfen (ganz ohne Komplexitätsannahmen), bei dem die Maschine mit dem größeren Laufzeitpolynom gewinnt. Das Laufzeitpolynom einer Maschine M beschränkt nämlich nicht nur die Laufzeit, sondern indirekt auch die größtmögliche Länge einer von M gesandten Nachricht. Somit können wir das Spiel vom vorangegangenen Beispiel wiederverwerten, bei dem die Maschine gewinnt, die den längeren String eingibt. Tatsächlich können wir sogar das trennende Gegenbeispiel von Satz 5.1 unverändert wiederverwenden. Lediglich der Beweis verändert sich ein wenig. So wird eine nach dem Simulator gewählte Umgebung einfach eine Nachricht schicken, die länger als das Laufzeitpolynom des polynomiell-beschränkten Simulators ist,[3] und somit immer unterscheiden, während ein nach der Umgebung gewählter Simulator auf analoge Weise das Spiel gewinnt und somit eine Unterscheidung verhindern kann. Erfreulicherweise ist dieser Beweis, anders als der von Satz 5.1, unabhängig davon, ob die Umgebung einen Auxiliary input bekommt oder nicht.

Korollar 5.2 (Trennung von komplexitätstheoretischer allgemeiner und spezieller Sicherheit)
Es existieren Protokolle π und ρ, für die folgendes gilt:

(Fortsetzung nächste Seite)

[3] Eine solche Umgebung ist ebenfalls polynomiell-beschränkt.

(Fortsetzung)

- Das Protokoll π ist so sicher wie ρ bezüglich *spezieller* komplexitätstheoretischer Sicherheit sowohl mit als auch ohne Auxiliary input sowohl bzgl. der Sicht als auch bzgl. der Ausgabe der Umgebung.
- Das Protokoll π ist *nicht* so sicher wie ρ bezüglich *allgemeiner* komplexitätstheoretischer Sicherheit weder mit noch ohne Auxiliary input und weder bzgl. der Sicht noch bzgl. der Ausgabe der Umgebung.

Beweis: Da der Beweis in seinen Grundzügen stark dem von Satz 5.1 ähnelt, beschreiben wir nur die Unterschiede.

Die Protokolle π und ρ sind genauso gewählt wie im anderen Beweis. Im Falle der speziellen Sicherheit wird die Zahl n_k (also die Länge der vom Simulator an M_{ideal} gesandten Nachricht) als $n_k := p(k) + 1$ gewählt, wobei p ein die Laufzeit der Umgebung beschränkendes Polynom ist. Der resultierende Simulator ist polynomiell-beschränkt, da n_k ein Polynom in k ist. Da die Umgebung keine Nachrichten senden kann, die länger als $p(k)$ sind, gilt immer $s_{\mathcal{S}} > s_{\mathcal{Z}}$ (falls beide definiert sind), und es folgt wie im anderen Beweis, daß die Sicht und Ausgabe der Umgebung statistisch ununterscheidbar sind (selbst wenn wir über alle Auxiliary inputs $z \in \Sigma^*$ quantifizieren), und somit π so sicher wie ρ ist bezüglich spezieller Sicherheit.

Im Falle der allgemeinen Sicherheit wird umgekehrt $n(k)$ (die Länge der von der Umgebung gesandten Nachricht) als $p'(k)+1$ gewählt, wobei p' ein die Laufzeit des Simulators beschränkendes Polynom ist. Wie oben ist damit auch der Simulator polynomiell-beschränkt, und wir haben $s_{\mathcal{Z}} > s_{\mathcal{S}}$ (falls beide definiert sind). Es folgt wie im anderen Beweis die statistische Unterscheidbarkeit von Sicht und Ausgabe der Umgebung und damit, daß π *nicht* so sicher wie ρ ist bezüglich allgemeiner Sicherheit. □

5.4. Trennung mit Time-lock puzzles

Das trennende Beispiel in Korollar 5.2 hat einen Nachteil. Das Protokoll ρ ist nicht polynomiell-beschränkt. Zwar macht die Maschine M_{ideal} nichts anderes, als die Längen der eingehenden Nachrichten zu vergleichen, und ist somit polynomiell in ihrer Eingabe. Da aber keine a priori gegebene Schranke für die Länge dieser Eingaben angegeben werden kann, ist die Laufzeit der Maschine M_{ideal} nicht im Sicherheitsparameter polynomiell beschränkt. Dies mag auf den ersten Blick harmlos erscheinen, da M_{ideal} zumindest in einem intuitiven Sinne polynomiell ist, jedoch gelten die Kompositionstheoreme 2.19 und 2.26

nur für im formalen Sinne polynomiell-beschränkte Protokolle, und die Behandlung von „intuitiv polynomiellen“ Protokollen stellt eine noch nicht vollständig zufriedenstellend gelöste Aufgabe dar (vgl. auch die Diskussion am Anfang von Abschnitt 2.2.3 und [HMQU05a]). Man stellt sich daher die Frage, ob nicht auch ein trennendes Beispiel für spezielle und allgemeine komplexitätstheoretische Sicherheit existiert, welches aus polynomiell-beschränkten Protokollen besteht.

Diese Frage werden wir im folgenden positiv beantworten, jedoch benötigen wir dazu einige spezielle Komplexitätsannahmen, die wir im folgenden zunächst vorstellen und untersuchen werden.[4] Danach stellen wir in Abschnitt 5.4.3 das trennende Beispiel vor.

5.4.1. Time-lock puzzles

Um es zu ermöglichen, in einem interaktiven Spiel die bezüglich ihrer Rechenzeit mächtigere Maschine zu identifizieren, brauchen wir zumindest einmal die Annahme, daß eine mächtigere polynomiell-beschränkte Maschine mehr berechnen kann als eine andere, weniger mächtige. Auf den ersten Blick scheint dies trivial, doch ist es nicht unmittelbar klar, daß nicht ein festes Polynom p existiert, so daß alles, was in polynomieller Zeit lösbar ist, bereits mit p Rechenschritten lösbar ist.[5] Insbesondere scheint dies nicht aus den in der Kryptographie üblichen Komplexitätsannahmen wie z. B. Einwegfunktionen, Trapdoor-Permutationen etc. zu folgen, da diese üblicherweise nur zwischen schwachen (üblicherweise polynomiellen oder subexponentiellen) und starken (üblicherweise exponentiellen oder superpolynomiellen) Maschinen unterscheiden, aber keine feinere Hierarchie postulieren. Daher brauchen wir eine neue Komplexitätsannahme, die explizit die Unterscheidbarkeit verschiedener polynomieller Komplexitätsklassen garantiert.

Eine der vermutlich schwächsten in diesem Zusammenhang verwendbaren Komplexitätsannahmen ist die, daß sogenannte Time-lock puzzles (im folgenden TL-Puzzles) existieren. Unter einem TL-Puzzle (laut [RSW96] erstmalig in [May93] vorgeschlagen) verstehen wir intuitiv ein irgendwie geartetes Problem, das von einer Maschine, dem Verifier, generiert wird, welche auch die Lösung des Problems überprüft. Dabei hat der Verifier die Möglichkeit, die Schwierigkeit des Problems in einem gewissen Rahmen frei einzustellen, von sehr einfach bis nicht

[4] Und die Ergebnisse in Kapitel 7 geben starke Evidenz, daß ohne Komplexitätsannahmen kein trennendes Beispiel gefunden werden kann: Dort wird gezeigt (Korollar 7.20) daß bei statistischer Sicherheit und polynomiell-beschränkten Protokollen spezielle und allgemeine Sicherheit zusammenfallen.

[5] Tatsächlich sind Ergebnisse in dieser Richtung bekannt, sog. Time Hierarchy Theorems (z. B. [HS65, CS99, Bar02]). Diese sind hier aber leider nicht anwendbar, da sie zwar die Existenz von Sprachen garantieren, die für ein Polynom p nicht lösbar, für ein größeres Polynom p' aber doch lösbar sind, nicht aber, daß die harten Instanzen in diesen Sprachen effizient gefunden werden können. Vgl. auch Abschnitt 5.4.2.3.

effizient lösbar. Damit hat der Verifier die Möglichkeit zu testen, wie mächtig eine polynomiell-beschränkte Maschine M tatsächlich ist, indem er überprüft, Probleme welchen Schwierigkeitsgrads M noch lösen kann. Die lösende Maschine nennen wir im folgenden Prover, da es sich bei den TL-Puzzles in gewissem Sinn um einen Beweis der Rechenmächtigkeit handelt.

Es gibt nun verschiedene Möglichkeiten, diese Intuition mit Details zu versehen:

- Zunächst einmal kann man sich die Frage stellen, von welcher Form das Problem ist: Handelt es sich z. B. um eine Nachricht, deren Lösung eine zweite Nachricht ist, oder muß der Prover in Interaktion mit dem Verifier beweisen, daß er in der Lage ist, das Problem zu lösen? Gibt es nur eine gültige Lösung, oder gibt es potentiell viele korrekte Antworten? Muß der Verifier selbst in der Lage sein, eine Lösung anzugeben, oder muß er nur in der Lage sein, die Lösung zu überprüfen? Können Außenstehende (die nur das Problem und die Lösung sehen) entscheiden, ob die Antwort korrekt ist?

  So genügt z. B. das in [RSW96] vorgestellte TL-Puzzle relativ strikten Anforderungen: Das Problem besteht aus einer Nachricht, hat nur eine korrekte Lösung, und diese kann der Verifier selbst effizient finden. Allerdings ist keine effiziente Möglichkeit bekannt, wie man die Lösung des TL-Puzzles überprüfen kann, ohne es selbst zu lösen. Da eine eindeutige, dem Verifier bekannte Lösung existiert, hat dieses TL-Puzzle einige zusätzliche potentielle Anwendungen. So kann man z. B. eine Nachricht mit der Lösung verschlüsseln, und erhält somit eine Art Zeitkapsel, die der Empfänger erst nach einer gewissen Anzahl an Rechenschritten öffnen kann.[6]

  Da wir hier aber möglichst schwache Annahmen machen wollen, verwenden wir eine Definition von TL-Puzzles, die es dem Prover erlaubt, dem Verifier interaktiv seine Mächtigkeit zu beweisen, und verlangen weder, daß der Verifier selbst in der Lage wäre, das TL-Puzzle zu lösen, noch daß ein Außenstehender die Lösung überprüfen kann.

- Darüber hinaus muß festgelegt werden, wie genau der Verifier die Schwierigkeit des TL-Puzzles „einstellen“ kann. Eine mögliche Forderung wäre, daß der Verifier exakt die Anzahl der Rechenschritte festlegen kann, die zur Lösung des Puzzles nötig sind. Mit dieser Anzahl von Schritten soll es dann möglich sein, das Puzzle zu lösen, mit weniger Schritten nicht. Diese

[6] [RSW96] gibt allerdings keine Definition eines TL-Puzzles, vielmehr werden die verschiedenen vermuteten Eigenschaften des dort vorgestellten TL-Puzzles nur auf intuitiver Ebene diskutiert.

Forderung ist natürlich extrem stark, so würde selbst eine leichte Optimierung des Lösungsalgorithmus die Annahme verletzen. Außerdem ist diese Definition sehr stark vom verwendeten Maschinenmodell abhängig.

Eine schwächere Forderung wäre es, anzunehmen, daß ein linearer Zusammenhang zwischen dem vom Verifier gewünschten Schwierigkeitsgrad $s \in \mathbb{N}$ und der zur Lösung notwendigen Laufzeit besteht.[7] Eine solche Lösung ist u. a. im Zusammenhang mit einer Modellierung der Maschinen als Turing-Maschinen interessant, da eine Turing-Maschine immer um einen konstanten Faktor beschleunigt werden kann, und somit ein strengerer Begriff nicht erfüllbar sein kann. Aber auch dieser Begriff verlangt bei formaler Definition noch eine genaue Spezifikation des zugrundeliegenden Maschinenmodells.

Etwas mit den obigen zwei Varianten Vergleichbares wird man vermutlich für praktische Anwendungen haben wollen, da man i. a. mehr oder minder konkrete Aussagen darüber treffen können will, wie lange ein Angreifer mindestens braucht, um die TL-Puzzles zu lösen (davon kann z. B. die maximale Wartezeit in einem Protokoll abhängen). Darüber hinaus wird man noch zusätzliche Forderungen stellen, beispielsweise daß das Lösen des Puzzles nicht parallelisierbar ist, da man dann bessere Aussagen über die schnellste existierende Hardware treffen kann. Solche Anforderungen sind beispielsweise bei dem TL-Puzzle von [RSW96] berücksichtigt.

Im Gegensatz dazu ist es für die Konstruktion unserer Gegenbeispiele ausreichend, daß der vom Verifier eingestellte Schwierigkeitsgrad s und die notwendige Laufzeit t des Provers nur in sehr schwacher Weise gekoppelt sind. So ist es ausreichend, daß sowohl s in t polynomial ist, als auch t in s. Dies ist für praktische Anwendungen zwar ungeeignet, ergibt aber eine sehr schwache und vom Maschinenmodell weitgehend unabhängige Definition.

- Einen weiteren subtilen Unterschied zwischen verschiedenen Interpretationen der Idee des TL-Puzzles illustrieren wir zunächst durch ein Beispiel. Es sei ein TL-Puzzle der folgenden Natur gegeben: Das Puzzle des Schwierigkeitsgrads s ist durch ein Prädikat P gegeben, welches die Eigenschaft hat, daß $\frac{1}{s}$ aller Belegungen von P erfüllend sind. Eine Lösung des Puzzles ist eine erfüllende Belegung von P. Wir nehmen an, die beste Möglichkeit, das Puzzle zu lösen, ist, zufällige Belegungen zu testen. Weiterhin dauere das Testen einer Belegung $1\,\mu sec$.

[7] Man mag noch einen logarithmischen oder polylogarithmischen Faktor zulassen, da viele natürliche Maschinenmodelle (insbesondere verschiedene Typen von RAM-Maschinen) bis auf einen logarithmischen Faktor in der Laufzeit äquivalent sind, vgl. [vEB90].

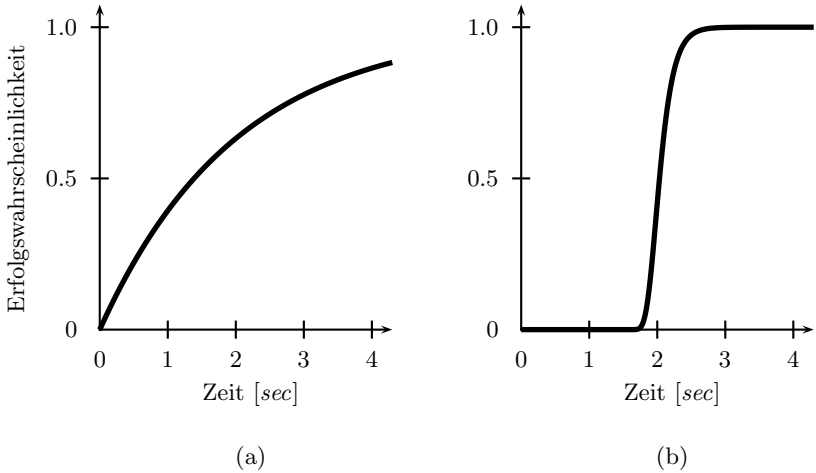


Abbildung 5.2.: Die Erfolgswahrscheinlichkeit beim Lösen eines Time-lock puzzles in Abhängigkeit der investierten Zeit: (a) bei einem schlechten TL-Puzzle und (b) bei einem guten TL-Puzzle.

Auf den ersten Blick handelt es sich hierbei um ein TL-Puzzle, das nichts zu wünschen läßt (abgesehen davon, daß der Lösungsalgorithmus sehr gut parallelisierbar ist).

Nun wird dieses TL-Puzzle in einem Protokoll verwendet: Zum Zeitpunkt $0\,sec$ wird ein TL-Puzzle des Schwierigkeitsgrads $s := 2 \cdot 10^6$ gestellt. Zum Zeitpunkt $1\,sec$ wird ein weiterer Protokollschritt durchgeführt, dessen Sicherheit darauf basiert, daß das TL-Puzzle zu diesem Zeitpunkt noch nicht gelöst wurde. Zwar garantiert uns das TL-Puzzle, daß man im Schnitt $s\,\mu sec = 2\,sec$ braucht, um ein Puzzle der Schwierigkeit s zu lösen, doch auch in $1\,sec$ findet man mit Wahrscheinlichkeit etwa 0,39 eine Lösung.[8] Man erkennt leicht, daß auch mit anderen Parametern dieses TL-Puzzle in diesem Zusammenhang nicht verwendbar ist. Entweder s ist polynomial, und dann ist die Wahrscheinlichkeit, das Puzzle nach $1\,sec$ zu lösen, nicht vernachlässigbar, oder s ist nicht mehr polynomiell, und damit das TL-Puzzle für keine polynomielle Maschine mehr lösbar.

Der Grund für dieses Problem liegt darin, daß die Wahrscheinlichkeit, das Puzzle zu lösen, relativ gleichmäßig mit dem investierten Aufwand ansteigt (Abbildung 5.2a), und daß selbst für geringen Aufwand bereits ei-

[8] $1 - \left(1 - 1/(2 \cdot 10^{-6})\right)^{10^6} \approx 0{,}39.$

ne kleine, aber nicht vernachlässigbare Erfolgswahrscheinlichkeit auftritt. Wir wollen aber, daß unterhalb einer gewissen Aufwandsschranke die Erfolgswahrscheinlichkeit vernachlässigbar ist, wohingegen sie ab einem gewissen Aufwand überwältigend wird, die Erfolgswahrscheinlichkeit muß also mehr oder weniger sprunghaft ansteigen. Ein Beispiel für einen geeigneten Zusammenhang zwischen Zeit und Erfolgswahrscheinlichkeit zeigt Abbildung 5.2b.

Wir werden im folgenden nur TL-Puzzles mit einem solchen sprunghaften Anstieg betrachten. Allerdings ist die andere Variante auch nicht ohne Relevanz, so existieren beispielsweise Vorschläge zur Spam-Vermeidung (z. B. [Bac02a, DNW05]), die auf sog. *Proofs of work* basieren, welche im wesentlichen TL-Puzzles mit einer Erfolgskurve wie in Abbildung 5.2a sind.

Zusammenfassend verlangen wir also von einem TL-Puzzle in etwa folgendes: Der Verifier ist eine interaktive Maschine, die in polynomieller Zeit läuft (im Sicherheitsparameter, aber nicht im Schwierigkeitsgrad). Das Puzzle soll einfach sein in folgendem Sinne: Ist der Schwierigkeitsgrad s durch ein Polynom beschränkt, so gibt es eine polynomiell-beschränkte Maschine, die mit überwältigender Wahrscheinlichkeit den Verifier zum Akzeptieren (zur Ausgabe 1) bringt. Andererseits soll das Puzzle auch in folgendem Sinne schwierig sein: Für jede polynomiell-beschränkte Maschine M existiert ein Polynom p, so daß M für Schwierigkeitsgrade $s \geq p$ nur mit vernachlässigbarer Wahrscheinlichkeit den Verifier dazu bringt, zu akzeptieren.

Dies fixieren wir formal wie folgt:

Definition 5.3 (Time-lock puzzle)
Ein *Time-lock puzzle* besteht aus einer polynomiell-beschränkten interaktiven Turing-Maschine (PITM)[9] $\mathcal{V}$, dem *Verifier*, der die folgenden zwei Eigenschaften erfüllt:

- *Einfachheit.* Für jedes Polynom p existiert eine PITM C, so daß

$$\min_{s \leq p(k)} P\big(\langle C(1^k, s), \mathcal{V}(1^k, s)\rangle = 1\big)$$

  überwältigend in k ist (wie nennen s den Schwierigkeitsgrad und k den Sicherheitsparameter des TL-Puzzles).
- *Schwierigkeit.* Für jede PITM B existiert ein Polynom p, so daß

$$\sup_{\substack{s \geq p(k) \\ z \in \Sigma^*}} P\big(\langle B(1^k, s, z), \mathcal{V}(1^k, s)\rangle = 1\big)$$

(Fortsetzung nächste Seite)

(Fortsetzung)

vernachlässigbar in k ist.

In den Grundzügen wurde diese Definition bereits oben erläutert. Erwähnenswert ist lediglich die Tatsache, daß die Maschine B in der Schwierigkeitsbedingung noch einen Auxiliary input z polynomieller Länge erhält. Dies hat zwei Gründe: Zum einen brauchen wir einen Begriff, der selbst in Präsenz eines Auxiliary inputs sicher ist, wenn wir die TL-Puzzles zum Trennen von komplexitätstheoretischer spezieller und allgemeiner Sicherheit *mit* Auxiliary input verwenden wollen. Zum anderen aber ist der Begriff des TL-Puzzles ohne Auxiliary input[10] unabhängig von unserem Anwendungszweck schon für sich problematisch, wie folgendes Beispiel zeigen soll: Man stelle sich vor, eine Maschine A wolle einer Maschine $\mathcal{V}$ beweisen, wie mächtig sie ist. Dazu wählt A einen Schwierigkeitsgrad s und löst dann das von $\mathcal{V}$ gestellte TL-Puzzle mit Schwierigkeitsgrad s. Ein Definition von TL-Puzzles ohne Auxiliary input gibt uns nun keine Sicherheitsgarantien: Die Maschine A könnte s und eine Hilfsinformation z gemeinsam konstruieren, so daß z beim Lösen von TL-Puzzles der Schwierigkeit s (und nur dieser Schwierigkeit) hilft. Wenn ein solches z nur gemeinsam mit s effizient konstruiert werden kann, nicht aber für vorgegebenes s, so widerspricht dies nicht der Definition von TL-Puzzles ohne Auxiliary input. Definition 5.3 hingegen garantiert, daß eine solche Hilfsinformation nicht helfen kann.

Nachdem wir nun die Definition von Time-lock puzzles präsentiert haben, sind wir in der Lage, das trennende Beispiel für komplexitätstheoretische spezielle und allgemeine Sicherheit zu präsentieren. Wir verschieben dies aber auf Abschnitt 5.4.3, und stellen zunächst einige Konstruktionen für TL-Puzzles vor, um dem Leser die Möglichkeit zu geben, einen Eindruck zu erlangen, wie realistisch die Existenz von TL-Puzzles ist. Diese Abschnitte sind aber nicht für das weitere Verständnis nötig, der geneigte Leser kann direkt zu Abschnitt 5.4.3 springen, falls er bereit ist, die Existenz von TL-Puzzles als Annahme zu akzeptieren.

5.4.2. Konstruktion von Time-lock puzzles

Im folgenden werden wir einige Kandidaten für Time-lock puzzles diskutieren. Zunächst (Abschnitt 5.4.2.1) präsentieren wir ein TL-Puzzle, welches in [RSW96] vorgestellt wurde. In Abschnitt 5.4.2.2 zeigen wir, wie man im Random-Oracle-Modell beweisbare TL-Puzzles konstruiert. Und in Abschnitt 5.4.2.3 ge-

[9] Man erinnere sich, daß bei uns eine PITM in der Länge ihres ersten Arguments (hier immer k) polynomiell-beschränkte Laufzeit hat.

[10] Wie Definition 5.3, nur erhält B das Argument z nicht.

ben wir einige Techniken an, die es ermöglichen, TL-Puzzles zu konstruieren, bei denen die Lösung die iterierte Anwendung einer vorgegebenen Funktion auf einer festen Eingabe ist.

5.4.2.1. Das Time-lock puzzle nach Rivest, Shamir, Wagner

Eine der Herausforderungen bei der Konstruktion von TL-Puzzles ist, daß nicht nur der Aufwand, das Puzzle zu lösen, mit dem Schwierigkeitsgrad zunehmen muß, sondern daß trotzdem der Verifier weiterhin effizient die Lösung des Puzzles überprüfen können muß. [RSW96] zeigen eine Möglichkeit, wie man dieses Problem auf elegante Weise lösen kann. Das dort vorgestellte TL-Puzzle bezeichnen wir im folgenden als das RSW-Puzzle.

Der Verifier des RSW-Puzzles geht wie folgt vor: Es bezeichne $s \leq 2^k$ den gewünschten Schwierigkeitsgrad. Zunächst wird ein zufälliger Blum-Integer[11] $n = pq$ der Länge k gewählt und an den Prover geschickt. Eine Lösung des RSW-Puzzles ist nun die Zahl $2^{2^s} \mod n$.

Offensichtlich ist es möglich durch s-faches Quadrieren modulo n die Lösung des RSW-Puzzles in s Iterationen zu finden. Das Puzzle erfüllt also die Einfachheitsbedingung. Andererseits ist keine schnellere Methode bekannt, um die Lösung zu finden. Somit scheint auch die Schwierigkeitsbedingung erfüllt zu sein.

Allerdings ist es auf den ersten Blick nicht ersichtlich, wie der Verifier die Lösung des Puzzles in polynomieller Zeit überprüfen soll. Hier aber nutzen wir die Tatsache, daß die Faktorisierung von n bekannt ist. Es ist

$$2^{2^s} \equiv 2^{2^s \bmod \varphi(n)} \bmod n,$$

wobei φ die Eulersche-Phi-Funktion bezeichne. Unter Kenntnis der Faktorisierung von n kann man $\varphi(n) = (p-1)(q-1)$ berechnen, und somit auch $s' := 2^s \bmod \varphi(n)$. Da aber $\varphi(n) \leq n < 2^k$, hat s' höchstens die Länge k, so daß $2^{s'} \bmod n$ in polynomieller Zeit in k berechenbar ist. Somit kann der Verifier nicht nur die Lösung überprüfen, sondern sie sogar selbst angeben, was es z. B. erlauben kann, die Lösung zur Verschlüsselung von Daten zu verwenden, die vom Empfänger erst nach einer gewissen Zeit gelesen werden dürfen (eine Art Zeitkapsel).[12]

[11] Ein Blum-Integer ist das Produkt aus zwei Primzahlen p und q mit $p \equiv q \equiv 3 \bmod 4$.

[12] Allerdings folgt aus der Annahme, daß das RSW-Puzzle ein TL-Puzzle nach Definition 5.3 ist, nicht notwendigerweise, daß eine derart konstruierte Zeitkapsel auch sicher ist. Im sog. Random-Oracle-Modell jedoch kann man eine solche Zeitkapsel leicht beweisbar aus dem RSW-Puzzle konstruieren (unter der Annahme, daß das RSW-Puzzle ein TL-Puzzle ist), indem man die Lösung des TL-Puzzles als Eingabe des Random-Oracles nimmt, und die resultierende Ausgabe als Schlüssel verwendet, um die Daten zu verschlüsseln.

Weitere Anwendungen des RSW-Puzzles werden in [RSW96] diskutiert. Das Hauptproblem des RSW-Puzzles ist, daß es auf einer sehr speziellen Komplexitätsannahme basiert (die Komplexitätsannahme ist gerade, daß das RSW-Puzzle ein TL-Puzzle ist), und nicht bekannt ist, in welcher Beziehung diese Komplexitätsannahme zu anderen, gebräuchlicheren oder schwächeren Komplexitätsannahmen steht.

5.4.2.2. Im Random-Oracle-Modell

Eine beliebte Heuristik in der Kryptographie ist die Random-Oracle-Heuristik. Hier wird zunächst angenommen, daß alle Parteien (einschließlich des/der Angreifer) Zugriff auf ein gemeinsames sog. Random-Oracle haben. Dieses Orakel stellt eine gleichverteilt zufällig gezogene Funktion $\mathcal{O} : \{0,1\}^k \to \{0,1\}^k$ dar. Viele Protokolle, die ein solches Random-Oracle benutzen, lassen sich ohne Benutzung von Komplexitätsannahmen als sicher beweisen, unter Benutzung der Tatsache, daß eine polynomiell-beschränkte Maschine nur polynomiell viele Zugriffe auf das Orakel durchführen kann. Andere Protokolle benötigen weiterhin Komplexitätsannahmen, sind aber wesentlich einfacher zu beweisen als vergleichbare Protokolle ohne Random-Oracle. In manchen Fällen sind sogar keine oder keine effizienten beweisbar sicheren Protokolle ohne Benutzung des Random-Oracles bekannt. Hat man nun ein Protokoll im Random-Oracle gefunden, ist dies zwar vielleicht beweisbar sicher, aber für sich stehend erst einmal nutzlos, da in der Realität keine Random-Oracles zur Verfügung stehen. Die Random-Oracle-Heuristik besagt nun, daß, wenn man ein im Random-Oracle-Modell sicheres Protokoll nimmt, und jeden Aufruf des Random-Oracles durch die Auswertung einer hinreichend komplexen und strukturlosen Funktion ersetzt (ein typischer Kandidat wäre z. B. die SHA-Funktionenfamilie [SHS02]), das resultierende Protokoll auch sicher ist. (Die dem zugrundeliegende Idee ist die, daß eine hinreichend undurchschaubare Funktion keinen Angriff zuläßt, der nicht auch bei einer zufälligen Funktion möglich wäre.)

Leider wurde die Random-Oracle-Heuristik als falsch bewiesen: In [CGH98] wurde gezeigt, daß im Random-Oracle-Modell sichere asymmetrische Verschlüsselungs- und Signaturverfahren existieren, die bei *jeder* Instantiierung des Random-Oracles durch eine konkrete Funktion (oder eine parametrisierte Funktionenschar) unsicher werden. Die entsprechenden Gegenbeispiele sind jedoch sehr konstruiert, daher wird oft angenommen, daß für „natürliche“ Protokolle die Heuristik weiterhin gilt. Die Random-Oracle-Heuristik erfreut sich daher weiterhin großer Beliebtheit, so ist z. B. das in der Praxis weit verbreitete und im PKCS #1-Standard [PKC02] fixierte asymmetrische Verschlüsselungsverfahren RSA-OAEP nur im Random-Oracle-Modell bewiesen [BR95, FOPS04].

Im Lichte der Random-Oracle-Heuristik stellen wir nun ein TL-Puzzle vor, welches wir *im Random-Oracle-Modell* als sicher beweisen. Wir sehen darin kei-

nen Beweis, aber ein starkes Indiz, daß mittels geeigneter Instantiierungen des Random-Oracles TL-Puzzles ohne Random-Oracle konstruiert werden können.

Um ein TL-Puzzle des Schwierigkeitsgrads s zu erstellen, wählt der Verifier zufällige, unabhängige Zahlen $y_1, \ldots, y_k \in \{0, \ldots, s-1\}$. Diese sendet der Verifier an den Prover. Eine gültige Lösung des TL-Puzzles sind Werte $x_1, \ldots, x_k \in \{0, \ldots, 2^{k/2} - 1\}$ mit $\mathcal{O}(i \| x_i) \equiv y_i \bmod s$ für alle i. Hierbei werden i und x_i als $\frac{k}{2}$-Bit-Strings aufgefaßt, $i \| x_i$ bezeichne die Konkatenation von i und x_i, und die Ausgabe des Random-Oracles fassen wir als Zahl in $\{0, \ldots, 2^k - 1\}$ auf.

Die Einfachheitsbedingung ist für dieses TL-Puzzle gegeben: Ein zufälliges x_i erfüllt mit Wahrscheinlichkeit ungefähr $\frac{1}{s}$ die Bedingung $\mathcal{O}(i \| x_i) \equiv y_i \bmod s$, so daß man k solche x_i mit überwältigender Wahrscheinlichkeit findet, wenn man $\mathcal{O}$ an $\Theta(k^2 s)$ zufälligen Stellen auswertet. $\Theta(k^2 s)$ ist polynomiell-beschränkt, wenn s dies ist.

Auch die Härtebedingung ist nicht schwierig zu zeigen: Es sei eine polynomiell-beschränkte ITM B gegeben, die versucht, das Puzzle zu lösen. Da B nur eine polynomiell-beschränkte Anzahl von Orakelzugriffen durchführt, können wir o. B. d. A. annehmen, daß ein Polynom q existiert, so daß folgendes gilt: (i) B gibt immer eine potentielle Lösung der Form $(x_1, \ldots, x_n)$ aus, für die gilt, daß B die Orakelaufrufe $\mathcal{O}(1 \| x_1), \ldots, \mathcal{O}(k \| x_k)$ durchgeführt hat (ansonsten verändern wir B einfach dahingehend, daß es vor der Ausgabe von $(x_1, \ldots, x_n)$ zusätzlich diese Orakelaufrufe ausführt), (ii) keine Orakelanfrage wird mehr als einmal durchgeführt (denn B kann sich die Antworten merken), (iii) für jedes $i = 1, \ldots, k$ ruft B das Orakel $\mathcal{O}$ genau $q(k)$-mal mit einer Eingabe der Form $i \| x$ auf (andernfalls lassen wir B noch zusätzliche Aufrufe durchführen). Es sei nun für den Sicherheitsparameter k der Schwierigkeitsgrad $s \geq 2q(k)$. Die Wahrscheinlichkeit, daß B eine vom Verifier akzeptierte Lösung ausgibt, ist beschränkt durch die Wahrscheinlichkeit, daß B für jedes i einen Orakelaufruf $\mathcal{O}(i \| x)$ mit $\mathcal{O}(i \| x) \equiv y_i \bmod s$ durchführt. Da alle Antworten des Orakels stochastisch unabhängig sind, gilt für festes i, daß die Wahrscheinlichkeit, daß B einen solchen Orakelaufruf für dieses i durchführt, durch $\frac{q(k)}{s} \leq \frac{1}{2}$ beschränkt ist (denn B führt $q(k)$ solche Aufrufe durch, und jeder hat eine Erfolgswahrscheinlichkeit von $\frac{1}{s}$). Damit ist die Wahrscheinlichkeit, daß B für jedes $i = 1, \ldots, k$ erfolgreich ist, durch $(\frac{1}{2})^k$ beschränkt (da der Erfolg für verschiedene i stochastisch unabhängig ist). Damit ist auch die Wahrscheinlichkeit für eine korrekte Lösung durch $(\frac{1}{2})^k$ beschränkt und die Härtebedingung gezeigt.

Im Random-Oracle-Modell gibt es also Time-lock puzzles, somit ist es, wenn man die Random-Oracle-Heuristik akzeptiert, sehr wahrscheinlich, daß es auch außerhalb des Random-Oracle-Modells TL-Puzzles gibt.

Dieses TL-Puzzle hat die Eigenschaft, daß Außenstehende (die nur s und die übertragenen Nachrichten erfahren) ebenfalls überprüfen können, ob das Puzzle

korrekt gelöst ist (denn der Verifier selbst benutzt zur Überprüfung nur diese öffentlich verfügbaren Informationen).

Im Gegensatz zum TL-Puzzle von Rivest, Shamir und Wagner (vergleiche Abschnitt 5.4.2.1) ist es dem Verifier nicht möglich, die Lösung des Puzzles vorherzusagen, was man wiederum brauchen würde, wollte man z. B. Zeitkapseln realisieren. Um zu zeigen, daß dies kein prinzipielles Manko des Random-Oracle-Ansatzes ist, geben wir noch (ohne Beweis) eine andere Konstruktion an, die die Eigenschaft hat, daß der Verifier die Lösung des TL-Puzzles bereits beim Erstellen des Puzzles lernt: Der Verifier wählt zufällige Zahlen $r_i \in \{0, \ldots, 2^k - 1\}$, $x_i \in \{0, \ldots, s-1\}$ für $i = 1, \ldots, k$ und setzt $y_i := \mathcal{O}(r_i + x_i \bmod 2^k)$. An den Prover wird $r_1, y_1, \ldots, r_k, y_k$ geschickt, und eine gültige Lösung für das Puzzle sind $\tilde{x}_1, \ldots, \tilde{x}_k \in \{0, \ldots, s-1\}$ mit $y_i = \mathcal{O}(r_i + \tilde{x}_i \bmod 2^k)$. Die Einfachheits- und die Schwierigkeitsbedingung lassen sich ähnlich wie oben zeigen. Und mit überwältigender Wahrscheinlichkeit sind die Lösungen $x_1, \ldots, x_k$, die der Verifier kennt, eindeutig (vorausgesetzt, s ist subexponentiell).

Die beiden eben vorgestellten TL-Puzzles beruhten auf der gleichen Technik: Man beginnt mit einem Puzzle, bei dem die *erwartete* Laufzeit, die nötig ist, um das Puzzle zu lösen, hinreichend groß ist. (Im Falle des ersten TL-Puzzles war dies z. B. die Aufgabe, zu gegebenem y ein Urbild x zu finden, so daß $\mathcal{O}(x) \equiv y \bmod s$.) Dies ist natürlich noch kein TL-Puzzle im Sinne von Definition 5.3, da dies nicht ausschließt, daß auch bei kurzer Laufzeit mit nicht vernachlässigbarer Wahrscheinlichkeit eine Lösung gefunden wird (vgl. die Diskussion am Anfang von Abschnitt 5.4.1 und Abbildung 5.2). Von diesem Puzzle haben wir k unabhängige Instanzen erstellt, und vom Prover verlangt, daß er *alle* k Instanzen korrekt löst. Dann haben wir ausgenutzt, daß wenn wir k unabhängige Wartezeiten haben (die Laufzeiten für die einzelnen Instanzen), deren Summe sich mit hoher Wahrscheinlichkeit nahe an der Summe der Erwartungswerte aufhält. Dieses Verfahren scheint relativ unabhängig davon zu sein, wie das zugrundeliegende Puzzle aussieht, und es drängt sich die Vermutung auf, daß dieses Verfahren eine allgemeingültige Transformation liefern könnte. Genauer, es sei ein Puzzle gegeben, welches die folgenden Eigenschaften hat (wir beschränken uns auf eine grobe Skizze der Eigenschaften):

- Das Puzzle ist effizient stell- und überprüfbar.
- Um das Puzzle zu lösen, braucht man eine *erwartete* Laufzeit in der Größenordnung von s.
- Das Puzzle ist nicht amortisierbar: Um n unabhängige Puzzles zu lösen, benötigt man eine *erwartete* Laufzeit in der Größenordnung von $n \cdot s$.

(Diese Eigenschaften wurden in [DN93] unter dem Namen *Pricing function* zur Spamvermeidung vorgeschlagen.) Man kann nun vermuten, daß aus einem Verfahren, welches diese Eigenschaften erfüllt, durch Parallelisierung ein TL-Puzzle wird. Dies können wir aber bereits durch folgendes einfaches Beispiel widerlegen (wobei wir den Beweis des Gegenbeispiels nur grob andeuten): Ein Puzzle mit dem Schwierigkeitsgrad s sei gegeben durch einen k-Bit-String q. Eine akzeptierte Lösung ist ein Tupel $(m_1, \ldots, m_l, t)$ mit $t, m_i \in \{0,1\}^k$, $l \geq 1$, $q \in \{m_1, \ldots, m_l\}$ und $\mathcal{O}(m_1 \| \ldots \| m_l \| t) \equiv 0 \bmod ls$.[13] Man sieht leicht ein, daß die erwartete Laufzeit, um eine Lösung zu finden, in $\Theta(s)$ liegt. Weiterhin läßt sich einsehen, daß dieses Puzzle nicht amortisierbar ist. Selbst wenn man z. B. n Puzzles zugleich zu lösen versucht, indem ein t mit $\mathcal{O}(q_1 \| \ldots \| q_n \| t) \equiv 0 \bmod ns$ gesucht wird, braucht man $\Theta(ns)$ Schritte.[14]

Sind jedoch k Puzzles $q_1, \ldots, q_k$ mit dem gleichen Schwierigkeitsgrad s gegeben, so gilt für zufälliges t mit Wahrscheinlichkeit $\frac{1}{sk}$, daß $\mathcal{O}(q_1 \| \ldots \| q_k \| t) \equiv 0 \bmod ks$ ist. Dann ist $(q_1, \ldots, q_k, t)$ eine Lösung für *jedes* der k Puzzles. Somit kann man alle Puzzles zusammen mit nicht vernachlässigbarer Wahrscheinlichkeit lösen, die Parallelisierung dieses – zugegebenermaßen nicht sehr natürlichen – Puzzles ist also kein TL-Puzzle.

5.4.2.3. Mit schwer iterierbaren Funktionen

In den vorangegangenen Abschnitten haben wir die Konstruktion von Timelock puzzles unter Benutzung (a) einer speziellen RSA-Komplexitätsannahme und (b) der Random-Oracle-Heuristik betrachtet. Nun wollen wir untersuchen, inwiefern wir die Existenz von TL-Puzzles auf allgemeine kryptographische Annahmen reduzieren können.

Dazu betrachten wir zunächst nochmal die Eigenschaften, die ein TL-Puzzle haben soll (vgl. die Diskussion zu Beginn von Abschnitt 5.4.1). Hier fällt uns auf, daß zwei wesentliche Aspekte ein TL-Puzzle ausmachen: Zunächst einmal verlangen wir, daß die Laufzeit, die benötigt wird, um das TL-Puzzle zu lösen, in einem hinreichend engen Zusammenhang mit dem Schwierigkeitsgrad des Puzzles steht. Und zum anderen erwarten wir, daß die Lösung durch den Verifier auch effizient überprüft werden kann. Obwohl die erste Eigenschaft die ist, die ein TL-Puzzle erst ausmacht, ist es oft die zweite, die die Suche nach einem TL-Puzzle erschwert. So scheint es nicht unrealistisch, daß viele Funktionen g die Eigenschaft haben, daß $g^s(x)$ nicht in weniger als s Schritten zu berechnen ist. Doch nur in wenigen Fällen ist es möglich, z. B. einen zahlentheoretischen

[13] Hier verwenden wir ein Random-Oracle, bei dem der Urbildbereich Strings beliebiger Länge enthält. Das Beispiel läßt sich aber leicht in eines umwandeln, bei dem der Urbildbereich aus k-Bit-Strings besteht.

[14] Ein genauer Beweis kann sich natürlich nicht auf diesen Spezialfall zurückziehen, sondern muß auch „Mischstrategien“ berücksichtigen, bei denen mehrere, aber nicht alle Puzzles gleichzeitig bearbeitet werden.

Trick wie beim TL-Puzzle von Rivest, Shamir und Wagner (Abschnitt 5.4.2.1) zu finden, der es dem Verifier ermöglicht, $g^s(x)$ selbst wesentlich schneller zu berechnen.

Aus diesem Grunde ist es wünschenswert, die beiden Eigenschaften getrennt betrachten zu können. Dies ist tatsächlich möglich, und wir werden im folgenden eine allgemeine Konstruktion vorstellen, um die Überprüfbarkeit durch den Verifier „nachträglich“ hinzuzufügen. Dazu müssen wir zunächst definieren, was wir uns unter einem „TL-Puzzle ohne effiziente Überprüfbarkeit“ vorstellen. Anstatt zu versuchen, die Definition 5.3 in voller Allgemeinheit nachzuvollziehen, beschränken wir uns auf eine speziellere Definition, die im wesentlichen einem „*nichtinteraktivem* TL-Puzzle ohne effiziente Überprüfbarkeit *mit eindeutiger Lösung*“ entspricht: die *härteregulierbaren Funktionen*.

Definition 5.4 (Härteregulierbare Funktionen)
Eine Funktion $f : \mathbb{N} \times \Sigma^* \to \Sigma^*$ zusammen mit einem probabilistischen polynomiellen Algorithmus G (dem *Instanzgenerator*) heißt *härteregulierbar*, wenn folgendes gilt:

- *Längenbeschränkung.* Es existiert ein Polynom p, so daß für alle $s \in \mathbb{N}, q \in \Sigma^*$ gilt: $|f(s,q)| \leq p(\log s + |q|)$.[15]
- *Einfachheit.* Es gibt einen probabilistischen Algorithmus C, so daß gilt:
$$\inf_{s \in \mathbb{N}} P\big(q \leftarrow G(1^k, s) : C(1^k, s, q) = f(s,q)\big)$$
ist überwältigend in k. Weiterhin soll die Laufzeit von C polynomiell in s und k beschränkt sein.[16]
- *Schwierigkeit.* Für jeden (in k) polynomiell-beschränkten probabilistischen Algorithmus B existiert ein Polynom p, so daß
$$\sup_{\substack{s \geq p(k) \\ z \in \Sigma^*}} P\big(q \leftarrow G(1^k, s) : B(1^k, s, q, z) = f(s,q)\big)$$
vernachlässigbar in k ist.

Wir nennen eine härteregulierbare Funktion f *deterministisch*, wenn der Algorithmus C aus der Einfachheitsbedingung deterministisch ist.

Die Annahme, daß härteregulierbare Funktionen existieren ist nun vermutlich wesentlich einfacher zu akzeptieren als die, daß TL-Puzzles existieren. Im folgen-

[15] Andernfalls könnte der Verifier die Antwort nicht einmal lesen. Härteregulierbare Funktionen *ohne* Längenbeschränkung existieren trivialerweise: $f(s,q) := 1^s$ ist deterministisch härteregulierbar. In der Tat beruht das trennende Beispiel aus Abschnitt 5.3 gerade auf dieser Tatsache.

[16] Also insbesondere polynomiell in s und nicht in der Länge von s.

den werden wir zunächst zeigen, daß man aus einer härteregulierbaren Funktion unter Benutzung schwacher kryptographischer Annahmen ein TL-Puzzle konstruieren kann.

Bevor wir unsere Konstruktion vorstellen können, müssen wir erst einmal das Konzept der *Universal arguments* [BG02] einführen. Vereinfacht gesagt handelt es sich bei Universal arguments um ein interaktives Beweissystem, mit dem man die Wahrheit einer Aussage x zeigen kann, und das zusätzlich die folgende Eigenschaft hat: Selbst wenn das Überprüfen der Aussage x durch den Prover sehr aufwendig ist, so ist doch der Aufwand des Verifiers, um den *Beweis* zu überprüfen, gering. Wir werden dies zur Konstruktion von TL-Puzzles ausnutzen, indem wir den Prover $a := f(s,q)$ berechnen lassen, und ihn danach ein Universal argument ausgeben lassen, daß in der Tat $a = f(s,q)$ gilt, um so den Verifier davon zu überzeugen, daß der Prover tatsächlich den verlangten Rechenaufwand betrieben hat. Die Definition eines Universal arguments (nach [BG02]) ist die folgende:

Definition 5.5 (Universal argument)
Es sei $R_{\mathcal{U}}$ die wie folgt definierte Relation: Für eine deterministische Turingmaschine M, ein $t \in \mathbb{N}$ und $x, w \in \Sigma^*$, sei $(M,x,t)R_{\mathcal{U}}\, w$ genau dann, wenn M die Eingabe (x,w) nach höchstens t Schritten akzeptiert. Es sei $L_{\mathcal{U}} := \{(M,x,t) : \exists w \in \Sigma^* \text{ mit } (M,x,t)R_{\mathcal{U}}\, w\}$ die zu $R_{\mathcal{U}}$ gehörige Sprache.[17] Wir betrachten $L_{\mathcal{U}}$ unter Benutzung einer geeigneten Kodierung als Teilmenge von Σ^*. Es bezeichne $T_M(x,w)$ die Anzahl der Schritte, die M bei Eingabe x, w läuft.

Ein *Universal argument* besteht aus zwei interaktiven Turing-Maschinen (ITM), dem Verifier $\mathcal{V}$ und dem Prover $\mathcal{P}$. Diese haben die folgenden Eigenschaften:

- *Effiziente Verifikation.* Es existiert ein Polynom p, so daß $\mathcal{V}$ bei Eingabe (M,x,t) maximal $p(|(M,x,t)|)$ Schritte läuft. (Man beachte, daß $|(M,x,t)| \in \Theta(|M| + |x| + \log t)$.)
- *Relativ-effiziente Vollständigkeit.* Für $(M,x,t)R_{\mathcal{U}}\, w$ ist

$$P\Big(\big\langle \mathcal{P}(M,x,t,w), \mathcal{V}(M,x,t)\big\rangle = 1\Big) = 1.$$

  Außerdem existiert ein Polynom p, so daß die Laufzeit von $\mathcal{P}$ bei Eingabe (M,x,t,w) höchstens $p(T_M(x,w))$ ist.
- *Computational soundness.* Es existiert eine vernachlässigbare Funktion μ, so daß für jeden polynomiell-beschränkten nichtuniformen pro-

(Fortsetzung nächste Seite)

(Fortsetzung)

babilistischen Algorithmus $\tilde{\mathcal{P}}$ und jedes $(M, x, t) \notin L_{\mathcal{U}}$ gilt:

$$P\Big(\big\langle \tilde{\mathcal{P}}(M,x,t), \mathcal{V}(M,x,t)\big\rangle = 1\Big) < \mu(n).$$

Dabei ist $\tilde{\mathcal{P}}$ in $|(M, x, t)|$ polynomiell-beschränkt. (Und nicht etwa in $|M|$ oder t.)

- *Schwache Proof-of-Knowledge-Eigenschaft.* Siehe [BG02].[18]

Detailliertere Erklärungen und Motivationen der verschiedenen Eigenschaften der Universal arguments finden sich in [BG02]. Dort wird auch der folgende Satz gezeigt:

Satz 5.6 (Existenz von Universal arguments)
Wenn eine kollisionsresistente Familie von Hashfunktionen existiert, dann existieren auch Universal arguments.

Wir können nun zum ersten Ergebnis dieses Abschnittes übergehen:

Satz 5.7 (Time-lock puzzles aus härteregulierbaren Funktionen)
Wenn eine kollisionsresistente Familie von Hashfunktionen und deterministische härteregulierbare Funktionen existieren, dann existieren auch Time-lock puzzles.

Beweis: Es sei f eine deterministische härteregulierbare Funktion mit zugehörigem Instanzgenerator G. Weiter seien $\mathcal{P}_{univ}$ und $\mathcal{V}_{univ}$ Prover und Verifier eines Universal arguments (letzteres existiert nach Satz 5.6).

Wir konstruieren nun ein TL-Puzzle wie folgt (wir geben gleichzeitig den Verifier $\mathcal{V}_{TL}$ und den zugehörigen Prover C_{TL} an):

[17] Die Relation $R_{\mathcal{U}}$ und die Sprache $L_{\mathcal{U}}$ sind auf diese Weise formuliert, weil sich alle anderen Sprachen L, die wir im Zusammenhang mit Universal arguments betrachten, leicht auf $L_{\mathcal{U}}$ reduzieren lassen: Es sei M die Turing-Maschine, die entscheidet, ob $x \in L$ ist (evtl. unter Zuhilfenahme einer nichtdeterministischen Eingabe w), und t eine (z. B. exponentielle) obere Schranke für die Laufzeit von M bei Eingabe (x, w). Für eine weitergehende Motivation dieser Sprache siehe [BG02].

[18] Da wir diese Eigenschaft nicht verwenden, ersparen wir dem Leser die Definition dieser recht komplizierten Eigenschaft.

- Es bezeichne k den Sicherheitsparameter und s den Schwierigkeitsgrad des TL-Puzzles. Die ITM $\mathcal{V}_{TL}$ und C_{TL} werden mit den Eingabe $(1^k, s)$ aufgerufen. Wenn $s > 2^k$, so terminieren $\mathcal{V}_{TL}$ und C_{TL} sofort.

- In der ersten Aktivierung wählt der Verifier eine Instanz q für die härteregulierbare Funktion f (d. h. $\mathcal{V}_{TL}$ wählt $q \leftarrow G(1^k, s)$) und sendet q an C_{TL}.

- C_{TL} berechnet $a := f(s, q)$ (dieser Schritt bewirkt, daß das TL-Puzzle nicht zu einfach ist). Dazu verwendet C_{TL} den deterministischen Algorithmus C_f aus Einfachheitsbedingung von Definition 5.4 (dort C genannt) mit Argumenten $(1^k, s, q)$. Dann sendet C_{TL} das Ergebnis a an $\mathcal{V}_{TL}$.

- Wenn a länger als die durch die Längenbeschränkung von f garantierte polynomielle Schranke ist, bricht $\mathcal{V}$ ab.

- C_{TL} beweist $\mathcal{V}_{TL}$ mittels des Universal arguments, daß in der Tat $a = f(s, q)$ gilt. Im Detail geschieht folgendes:

  Es sei M die Turingmaschine, die bei Eingabe $((k, s, a, q), w)$ die Turingmaschine $C_f(1^k, s, q)$ simuliert, und akzeptiert, wenn C_f den Wert a zurückgibt.

  Der Verifier $\mathcal{V}_{TL}$ des TL-Puzzles führt dann den Verifier $\mathcal{V}_{univ}$ des Universal arguments mit den Eingaben $(M, (k, s, a, q), 2^k)$ aus. Der Prover C_{TL} des TL-Puzzles führt den Prover $\mathcal{P}_{univ}$ des Universal arguments mit den Eingaben $(M, (k, s, a, q), 2^k, \lambda)$ aus.

Zunächst wollen wir einsehen, daß $\mathcal{V}_{TL}$ polynomiell-beschränkt ist. Nach Definition 5.4 ist der Instanzgenerator G polynomiell-beschränkt, so daß die Wahl von q nur polynomiell-beschränkte Zeit in Anspruch nimmt. Weiterhin ist auch $|(M, (k, s, a, q), 2^k)| \in O(|M| + |(k, s, a, q)| + k)$ polynomiell-beschränkt, so daß auch das Ausführen von $\mathcal{V}_{univ}$ nur polynomiell-beschränkte Zeit in Anspruch nimmt (wegen der Bedingung der effizienten Verifikation in Definition 5.5). Somit ist $\mathcal{V}_{TL}$ in der Tat polynomiell-beschränkt.

Es bleiben die Einfachheit und die Schwierigkeit des TL-Puzzles zu beweisen. Wir beginnen mit der Einfachheit. Um diese zu zeigen, genügt es zu zeigen, daß (i) für in k polynomiell-beschränktes s die Laufzeit von $C_{TL}(1^k, s)$ ebenfalls in k polynomiell-beschränkt ist, und (ii) daß für alle s mit überwältigender Wahrscheinlichkeit

$$\langle C_{TL}(1^k, s), \mathcal{V}_{TL}(1^k, s) \rangle = 1 \tag{5.1}$$

gilt. Die Laufzeit von C_{TL} ist leicht zu analysieren: Nach der Einfachheitsbedingung der härteregulierbaren Funktionen hat $C_f(1^k, s, q)$ in k und s polynomiell-beschränkte Laufzeit. Nach Konstruktion von M ist dessen Laufzeit $T_M((k, s, a, q), \lambda)$ bei Eingabe $((k, s, a, q), \lambda)$ polynomiell-beschränkt in der

Laufzeit von C_f. Und wegen der relativ-effizienten Vollständigkeit des Universal arguments (Definition 5.5) ist die Laufzeit von $\mathcal{P}_{univ}$ wiederum polynomiell-beschränkt in $T_M((k,s,a,q),\lambda)$. Somit ist die Laufzeit von C_{TL} polynomiell in k und s beschränkt, und für damit in k, wenn s selbst in k polynomiell-beschränkt ist.

Für $a := C_f(1^k,s,q)$ akzeptiert nach Definition $M((k,s,a,q),w)$ immer. Da M nach einer polynomiell-beschränkten Anzahl von Schritten akzeptiert (s. o.), und somit in weniger als $t := 2^k$ Schritten, folgt wegen der relativ-effizienten Vollständigkeit, daß (5.1) für hinreichend großes k immer erfüllt ist. Es folgt die Einfachheitsbedingung des TL-Puzzles.

Wir wenden uns nun der Schwierigkeitsbedingung des TL-Puzzles zu. Hierzu nehmen wir an, das TL-Puzzle $\mathcal{V}_{TL}$ erfülle die Schwierigkeitsbedingung nicht. Damit existieren also eine PITM B_{TL}, so daß für jedes Polynom p

$$\gamma(k) := \sup_{\substack{s\geq p(k)\\ z\in\Sigma^*}} P\big(ACC\big) \tag{5.2}$$

nicht vernachlässigbar ist, wobei wir kurz ACC für $\langle B_{TL}(1^k,s,z),\mathcal{V}_{TL}(1^k,s)\rangle = 1$ schreiben.

Wir definieren nun B_f als den probabilistischen Algorithmus, der bei Eingabe $(1^k,s,q,z)$ die Nachricht a ausgibt, die die ITM $B_{TL}(1^k,s,z)$ bei Empfang der Nachricht q sendet. (Intuitiv ist q der von B_{TL} behauptete Wert von $f(q,s)$.) Offensichtlich ist B_f polynomiell-beschränkt. Wir werden nun zeigen, daß B_f mit nicht vernachlässigbarer Wahrscheinlichkeit $f(q,s)$ ausgibt und somit die Schwierigkeit der härteregulierbaren Funktion f widerlegt.

Es sei p ein festes Polynom. Zunächst gilt aufgrund der Computational soundness des Universal arguments $(\mathcal{P}_{univ},\mathcal{V}_{univ})$, daß der von $\mathcal{V}_{TL}$ ausgeführte Verifier des Universal arguments $\mathcal{V}_{univ}(M,(k,s,a,q),2^k)$ nur mit vernachlässigbarer Wahrscheinlichkeit akzeptiert, wenn $a \neq C_f(1^k,s,q)$. Da $\mathcal{V}_{TL}$ nur akzeptiert, wenn $\mathcal{V}_{univ}$ akzeptiert, ist also

$$\mu(k) := \sup_{\substack{s\geq p(k)\\ z\in\Sigma^*}} P\big(ACC \text{ und } a \neq C_f(1^k,s,q)\big) \tag{5.3}$$

vernachlässigbar (hierbei bezeichnen die Zufallsvariablen a und q die jeweiligen

von $\mathcal{V}_{TL}$ und B_{TL} gesandten Nachrichten). Wir rechnen weiter

$$\begin{aligned}
\delta(k) &:= \sup_{\substack{s\geq p(k)\\ z\in\Sigma^*}} P(a = C(1^k, s, q)) \\
&\geq \sup_{\substack{s\geq p(k)\\ z\in\Sigma^*}} P(ACC \text{ und } a = C_f(1^k, s, q)) \\
&= \sup_{\substack{s\geq p(k)\\ z\in\Sigma^*}} P(ACC) - P(ACC \text{ und } a \neq C_f(1^k, s, q)) \\
&\geq \gamma(k) - \mu(k).
\end{aligned}$$

Da γ nach (5.2) nicht vernachlässigbar und μ nach (5.3) vernachlässigbar ist, ist $\delta = \gamma - \mu$ nicht vernachlässigbar.

Nach Definition von $\mathcal{V}_{TL}$ (der q unter Benutzung von $G(1^k, s)$ wählt) und Definition von B_f (welcher die Nachricht a generiert) können wir δ umschreiben als

$$\delta(k) = \sup_{\substack{s\geq p(k)\\ z\in\Sigma^*}} P\big(q \leftarrow G(1^k, s) : B_f(1^k, s, q, z) = C_f(1^k, s, q)\big) \tag{5.4}$$

Nun gilt aber nach Definition von C_f und der Einfachheit von f (Definition 5.4), daß für mittels $q \leftarrow G(1^k, s)$ gewähltes q mit überwältigender Wahrscheinlichkeit $C_f(1^k, s, q) = f(s, q)$ gilt. Somit folgt aus (5.4) und der Tatsache, daß δ nicht vernachlässigbar ist, daß auch die folgende Wahrscheinlichkeit nicht vernachlässigbar ist:

$$\sup_{\substack{s\geq p(k)\\ z\in\Sigma^*}} P\big(q \leftarrow G(1^k, s) : B_f(1^k, s, q, z) = f(s, q)\big).$$

Da dies für alle Polynome p gilt, liegt ein Wiederspruch zur Schwierigkeit von f vor (vgl. Definition 5.4). Unsere oben getroffene Annahme, daß $\mathcal{V}_{TL}$ die Schwierigkeitsbedingung von Definition 5.3 nicht erfüllt, ist also falsch, und $\mathcal{V}_{TL}$ bildet somit ein TL-Puzzle. □

Wir sehen in Satz 5.7 ein starkes Indiz dafür, daß TL-Puzzles tatsächlich existieren.

Als Indiz für die Existenz von härteregulierbaren Funktionen und dafür, daß diese vielleicht sogar beweisbar sein könnte, kann man die sogenannten Time Hierarchy Theorems nehmen. Das erste Time Hierarchy Theorem wurde in [HS65] gezeigt und besagt (vereinfacht), daß für jedes Polynom p eine Sprache L existiert, die nicht in DTIME(p), aber in DTIME($p^{2+\varepsilon}$) liegt. Somit existiert also *beweisbar* eine feine Hierarchie innerhalb der in polynomieller Zeit entscheidbaren Sprachen. Ähnliche Theoreme existieren z. B. bzgl. der erwarteten Laufzeit [CS99] und für probabilistische Algorithmen mit kurzem Auxiliary

input [Bar02]. Für härteregulierbare Funktionen braucht man jedoch mehr als nur eine Hierarchie der Laufzeit, es ist zusätzlich nötig, daß schwierige Instanzen effizient gezogen werden können. Erste Schritte in diese Richtung finden sich in [FS04]. Die dort vorgestellten Ergebnisse kommen der Existenz härteregulierbarer Funktionen bereits sehr nah.

Es sollte allerdings nicht unerwähnt bleiben, daß die Konstruktion aus Satz 5.7 zwar von theoretischem Interesse ist, aber das resultierende TL-Puzzle im Gegensatz zu denen aus den Abschnitten 5.4.2.1 und 5.4.2.2 nicht praktisch verwendbar ist. Insbesondere kann der Aufwand, der nach der Einfachheitsbedingung durch einen ehrlichen Prover C höchstens getrieben werden muß, wesentlich höher sein als der nach der Schwierigkeitsbedingung durch B mindestens getriebene Aufwand. Darüber hinaus stellen Universal arguments eine sehr aufwendige Konstruktion dar und sind deshalb heutzutage noch nicht in praktischen Protokollen einsetzbar. Glücklicherweise sind diese Nachteile für unsere Anwendung der TL-Puzzles nicht relevant, da uns die Existenz derselben nur interessiert, um trennende Beispiele zu konstruieren.

Wir haben nun die Suche nach TL-Puzzles auf die vermutlich einfachere Suche nach deterministischen härteregulierbaren Funktionen zurückgeführt. Eine natürliche Konstruktion solcher Funktionen ist die bereits zu Beginn dieses Abschnitts angedeutete Iteration einer Funktion. Es scheint eine realistische Annahme zu sein, daß für viele Funktionen g der Ausdruck $g^s(x)$ nicht in weniger als s Schritten zu berechnen ist (eben durch i-fache Anwendung von g). Funktionen g, die diese Eigenschaft haben, wollen wir *schwer iterierbar* nennen. Wir geben nun eine formale Definition dieser Eigenschaft, gleich verallgemeinert auf Familien von Funktionen:

Definition 5.8 (Schwer iterierbare Funktionen)
Eine Familie g_i mit $i \in \Sigma^*$ von Funktionen auf Σ^* zusammen mit probabilistischen Algorithmen G (*Instanzgenerator*) und I (*Indexgenerator*) heißt *schwer iterierbar*, wenn folgende Eigenschaften zutreffen:

(i) Der Wert $g_i(x)$ ist in probabilistisch polynomiell-beschränkter Zeit berechenbar (in der Länge von i und x).

(ii) Für gültiges i (d. h. ein i im Bildbereich von I) ist g_i längenerhaltend, also $|g_i(x)| = |x|$.

(iii) Definieren wir $G_f(1^k) := (G(1^k), I(1^k))$, so ist die folgende Funktion f härteregulierbar mit Instanzgenerator G_f:

$$f(s, (x, i)) := g_i^s(x).$$

(Fortsetzung nächste Seite)

(Fortsetzung)

(Wir lassen also den Index i und den Startwert x von G und I wählen, und wenden g_i s-mal auf x an.)

Wir nennen g_i eine Familie von schwer iterierbaren *deterministischen* Funktionen, wenn in (i) der Wert $g_i(x)$ sogar in deterministisch polynomiell-beschränkter Zeit berechenbar ist.

Die Bedingung (ii) haben wir eingeführt, damit garantiert ist, daß $g_i(x)$ mit steigendem i nicht immer länger wird. In Bedingung (iii) fordern wir, daß die Funktion f härteregulierbar ist. Hier genügt es, die Härtebedingung zu prüfen, da die Einfachheitsbedingung bereits aus (i) und (ii) folgt. Insbesondere ist f im Falle einer Familie von schwer iterierbaren *deterministischen* Funktionen eine deterministische härteregulierbare Funktion. Im folgenden sprechen wir oft der Kürze halber einfach von Funktionen statt von Familien von Funktionen.

Es impliziert also die Existenz schwer iterierbarer Funktionen härteregulierbare Funktionen, und die Existenz schwer iterierbarer deterministischer Funktionen impliziert deterministische härteregulierbare Funktionen. Aus letzteren können wir mit Satz 5.7 TL-Puzzles konstruieren. Wir erhalten also:

Korollar 5.9 (Time-lock puzzles aus schwer iterierbaren Funktionen)
Existieren Familien von schwer iterierbaren deterministischen Funktionen und kollisionsresistente Familien von Hashfunktionen, so existieren auch Time-lock puzzles.

Mit dieser Definition können wir auch das TL-Puzzle von Rivest, Shamir und Wagner (siehe Abschnitt 5.4.2.1) in neuem Lichte betrachten. Die dort verwendete Komplexitätsannahme können wir nämlich nun sehr einfach formulieren: Die Funktion $g_n(x) := x^2 \bmod n$ ist schwer iterierbar.

Weitere realistische Kandidaten für deterministische schwer iterierbare Funktionen sind z. B. kollisionsresistente Hashfunktionen.[19]

5.4.3. Das trennende Beispiel

Nachdem wir die Komplexitätsannahme der Existenz von Time-lock puzzles eingeführt haben und motiviert, warum diese realistisch ist, können wir nun zum eigentlichen Thema dieses Kapitels zurückkehren, und uns dem Beweis

[19] Die Tatsache, daß es sich gerade bei diesen um gute Kandidaten handelt, ist das persönliche Gefühl des Autors und durch keinerlei formale Argumente belegt.

zuwenden, daß spezielle und allgemeine komplexitätstheoretische Sicherheit auseinanderfallen, und zwar selbst dann, wenn man nur polynomiell-beschränkte Protokolle betrachtet (im Gegensatz zu Korollar 5.2).

Wir rufen uns dazu zunächst nochmal das trennende Beispiel aus Abschnitt 5.3 in Erinnerung: Dort senden sowohl Umgebung $\mathcal{Z}$ als auch Simulator $\mathcal{S}$ eine unär kodierte Zahl $s_{\mathcal{Z}}$ bzw. $s_{\mathcal{S}}$ an das ideale Protokoll. Wenn $s_{\mathcal{Z}} > s_{\mathcal{S}}$, dann erfährt die Umgebung, daß sie mit dem idealen Protokoll spricht und kann unterscheiden. Ansonsten verhält sich das ideale Protokoll wie das reale. Wird der Simulator in Abhängigkeit von der Umgebung gewählt, so kann er $s_{\mathcal{S}}$ immer größer wählen als $s_{\mathcal{Z}}$, da die maximale Länge von $s_{\mathcal{Z}}$ durch ein nur von $\mathcal{Z}$ abhängiges Polynom beschränkt ist. Andersherum wird eine vom Simulator abhängige Umgebung diesen immer besiegen.

Es ist nun einfach, diese Idee auf TL-Puzzles zu übertragen: Anstatt eine Zahl der Länge $s_{\mathcal{S}}$ zu senden, löst der Simulator ein TL-Puzzle des Schwierigkeitsgrads $s_{\mathcal{S}}$. Analoges gilt für die Umgebung und ihren Schwierigkeitsgrad $s_{\mathcal{Z}}$. Es gewinnt, wer ein Puzzle größeren Schwierigkeitsgrads gelöst hat. Liegt also beispielsweise die Umgebung fest, so existiert nach der Schwierigkeitseigenschaft der TL-Puzzles (Definition 5.3) ein Polynom p, so daß die polynomiell-beschränkte Umgebung TL-Puzzles des Schwierigkeitsgrads $s_{\mathcal{Z}} \geq p$ nicht mehr lösen kann. Wegen der Einfachheitseigenschaft jedoch gibt es einen Simulator, der TL-Puzzles des Schwierigkeitsgrads $s_{\mathcal{Z}}$ löst. Somit wird immer der Simulator gewinnen. Analoges gilt für eine *nach* dem Simulator gewählte Umgebung.

Obwohl dieser Beweisansatz durchaus funktioniert, wählen wir einen leicht anderen: Nicht Umgebung und Simulator wählen den Schwierigkeitsgrad des Puzzles, sondern das Protokoll selbst wählt einen Schwierigkeitsgrad s, und sowohl Umgebung als auch Simulator müssen ein TL-Puzzle der Schwierigkeit s lösen. Wenn die Umgebung das TL-Puzzle löst, der Simulator aber nicht, so erlauben wir der Umgebung, reales und ideales Protokoll zu unterscheiden, in allen anderen Fällen nicht. Wieder sehen wir, daß ein nach der Umgebung gewählter Simulator alle TL-Puzzles lösen kann, die die Umgebung lösen kann. Somit sind reales und ideales Protokoll im Falle der speziellen Sicherheit ununterscheidbar. Andererseits können wir eine vom Simulator abhängige Umgebung so wählen, daß sie TL-Puzzles löst, die *doppelt* so schwierig sind wie das schwierigste vom Simulator lösbare. Dann wird mit einer gewissen, nicht vernachlässigbaren Wahrscheinlichkeit der vom Protokoll gewählte Schwierigkeitsgrad s im von der Umgebung lösbaren, vom Simulator aber nicht lösbaren Bereich liegen, und die Umgebung gewinnt und unterscheidet. Somit sind reales und ideales Protokoll im Falle der allgemeinen Sicherheit unterscheidbar.

Der Grund dafür, daß wir diesen auf den ersten Blick umständlicheren Ansatz vorziehen, liegt darin, daß in der Definition 5.3 der TL-Puzzles der Schwierigkeitsgrad nicht vom das Puzzle lösenden Prover gewählt wird. Es macht daher den formalen Beweis einfacher, wenn wir den Schwierigkeitsgrad nicht von Simu-

lator und Umgebung (die ja die Rolle des Provers übernehmen) wählen lassen. Andernfalls müßten wir das Vorhandensein des Auxiliary input in der Schwierigkeitsbedingung nutzen, um zu zeigen, daß der Prover keinen Vorteil dadurch erlangt, daß er den Schwierigkeitsgrad selbst wählt.

Wir können nun die eigentliche Konstruktion und den Beweis vorstellen:

Satz 5.10 (Polynomiell-beschränkte Trennung von komplexitätstheoretischer allgemeiner und spezieller Sicherheit)
Wenn Time-lock puzzles (Definition 5.3) existieren, dann gibt es *polynomiell-beschränkte* Protokolle π und ρ, für die folgendes gilt:

- Das Protokoll π ist so sicher wie ρ bezüglich *spezieller* komplexitätstheoretischer Sicherheit sowohl mit als auch ohne Auxiliary input sowohl bzgl. der Sicht als auch bzgl. der Ausgabe der Umgebung.
- Das Protokoll π ist *nicht* so sicher wie ρ bezüglich *allgemeiner* komplexitätstheoretischer Sicherheit weder mit noch ohne Auxiliary input und weder bzgl. der Sicht noch bzgl. der Ausgabe der Umgebung.

Beweis: Es sei $\mathcal{V}$ der Verifier eines TL-Puzzles. O. B. d. A. nehmen wir an, daß die erste Nachricht, die in der Interaktion zwischen dem Verifier $\mathcal{V}$ und einem Prover gesandt wird, vom Prover stammt. Weiterhin nehmen wir an, daß der Verifier $\mathcal{V}$ solange Nachrichten sendet, bis er terminiert, und daß $\mathcal{V}$ nur die Ausgaben 0 und 1 kennt.

Zunächst spezifizieren wir die Protokolle π und ρ. Das reale Protokoll π besteht aus einer einzigen Maschine M_{real}, das ideale Protokoll ρ aus einer einzigen Maschine M_{ideal}.

Die Maschine M_{real} ist wie folgt definiert (vgl. Abbildung 5.3a): M_{real} hat den eingehenden Protokolleingabeport `in_timelock` und die ausgehenden Protokollausgabeports `out_timelock` und `out_identity`. Bei Sicherheitsparameter k verhält sich M_{real} wie folgt:

- Bei der ersten Aktivierung wählt M_{real} ein gleichverteilt zufälliges $s \in \{2^1, 2^2, \ldots, 2^k\}$. Die Zahl s wird binär über den Protokollausgabeport `out_timelock` gesandt.

- Dann simuliert M_{real} eine Instanz $\mathcal{V}_{\mathcal{Z}}$ des Verifiers $\mathcal{V}$ mit Eingaben $(1^k, s)$. Bei Empfang einer Nachricht m über den Protokolleingabeport `in_timelock` wird diese an die ITM $\mathcal{V}_{\mathcal{Z}}$ ausgeliefert. Sendet $\mathcal{V}_{\mathcal{Z}}$ eine Nachricht, so wird diese über den Protokollausgabeport `out_timelock` gesandt. (In anderen Worten, M_{real} stellt der Umgebung ein TL-Puzzle der Schwierigkeit s.)

- Gibt $\mathcal{V}_{\mathcal{Z}}$ eine 1 aus (akzeptiert die Lösung des TL-Puzzles), so sendet M_{real} die Nachricht *real* über den Protokollausgabeport `out_identity`. Terminiert $\mathcal{V}_{\mathcal{Z}}$ mit Ausgabe 0, so wird stattdessen über diesen Port die Nachricht *lost* gesandt.

Die Maschine M_{ideal} (vgl. Abbildung 5.3b) hat einen eingehenden Protokolleingabeport `in_timelock`, einen eingehenden Angreiferport `adv_in_timelock`, ausgehende Protokollausgabeports `out_timelock` und `out_identity` und einen ausgehenden Angreiferport `adv_out_timelock`. Bei Sicherheitsparameter k verhält sich M_{ideal} wie folgt:

- Bei der ersten Aktivierung wählt M_{ideal} ein gleichverteilt zufälliges $s \in \{2^1, 2^2, \ldots, 2^k\}$. Die Zahl s wird binär über den Protokollausgabeport `out_timelock` gesandt. (Dieser Schritt ist identisch zum ersten Schritt von M_{real}.)

- Dann simuliert M_{ideal} eine Instanz $\mathcal{V}_{\mathcal{Z}}$ des Verifiers $\mathcal{V}$ mit den Eingaben $(1^k, s)$. Bei Empfang einer Nachricht m über den Protokolleingabeport `in_timelock` wird diese an die ITM $\mathcal{V}_{\mathcal{Z}}$ ausgeliefert. Sendet $\mathcal{V}_{\mathcal{Z}}$ eine Nachricht, so wird diese über den Protokollausgabeport `out_timelock` gesandt. (In anderen Worten, M_{ideal} stellt der Umgebung ein TL-Puzzle der Schwierigkeit s. Dieser Schritt ist identisch zum zweiten Schritt von M_{real}.)

- Sobald $\mathcal{V}$ terminiert (unabhängig von der Ausgabe) schickt M_{ideal} die Zahl s binär über den Angreiferport `adv_out_timelock`.

- Dann simuliert M_{ideal} eine zweite Instanz $\mathcal{V}_{\mathcal{S}}$ des Verifiers $\mathcal{V}$ mit den Eingaben $(1^k, s)$, leitet dessen Kommunikation aber diesmal über die Angreiferports `adv_in_timelock` und `adv_out_timelock`. (Dem Simulator wird ein TL-Puzzle der gleichen Schwierigkeit s gestellt.)

- Sobald auch $\mathcal{V}_{\mathcal{S}}$ terminiert hat, wird eine Nachricht *id* über den Protokollausgabeport `out_identity` geschickt. Hierbei unterscheiden wir drei Fälle:

  - Der Verifier $\mathcal{V}_{\mathcal{Z}}$ hat 0 ausgegeben (die Umgebung hat das TL-Puzzle nicht korrekt gelöst). Dann ist $id := lost$.
  - Die Verifier $\mathcal{V}_{\mathcal{Z}}$ und $\mathcal{V}_{\mathcal{S}}$ haben beide 1 ausgegeben (sowohl Umgebung als auch Simulator haben das TL-Puzzle korrekt gelöst). Dann ist $id := real$.
  - Der Verifier $\mathcal{V}_{\mathcal{Z}}$ hat 1 ausgegeben, der Verifier $\mathcal{V}_{\mathcal{S}}$ aber 0 (die Umgebung hat das TL-Puzzle korrekt gelöst, der Simulator aber nicht). Dann ist $id := ideal$.

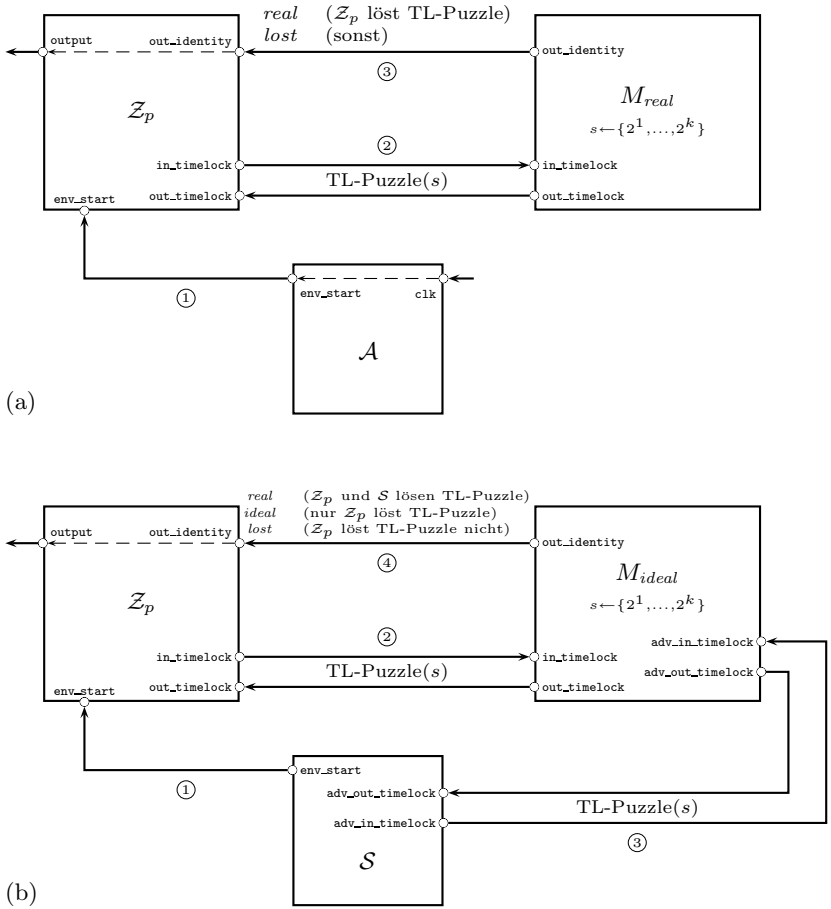


Abbildung 5.3.: Beweis der Unsicherheit bezüglich allgemeiner Sicherheit.

(a) Das reale Protokoll π bestehend aus der Maschine M_{real} zusammen mit Umgebung $\mathcal{Z}_p$ und Angreifer $\mathcal{A}$. ① Zunächst wird $\mathcal{Z}_p$ vom Angreifer aktiviert. ② Dann versucht $\mathcal{Z}_p$ das von M_{real} gestellte TL-Puzzle zu lösen (der Schwierigkeitsgrad s wird von M_{real} zufällig gewählt). ③ Schließlich sendet M_{real} die Nachricht $id \in \{real, lost\}$ an die Umgebung, welche diese ausgibt.

(b) Das ideale Protokoll ρ bestehend aus der Maschine M_{ideal} zusammen mit Umgebung $\mathcal{Z}_p$ und Angreifer $\mathcal{A}$. ① Zunächst wird $\mathcal{Z}_p$ vom Angreifer aktiviert. ② Dann versucht $\mathcal{Z}_p$ das von M_{ideal} gestellte TL-Puzzle zu lösen (der Schwierigkeitsgrad s wird von M_{ideal} zufällig gewählt). ③ Dann muß der Simulator versuchen, ein TL-Puzzle des gleichen Schwierigkeitsgrads s zu lösen. ④ Schließlich sendet M_{ideal} die Nachricht $id \in \{real, ideal, lost\}$ an die Umgebung, welche diese ausgibt.

Die Maschine M_{ideal} unterscheidet sich von der Maschine M_{real} also darin, daß sie auch dem Simulator ein TL-Puzzle der gleichen Schwierigkeit s stellt. Und wenn dieser scheitert, aber die Umgebung ihr TL-Puzzle löst, dann wird der Umgebung die Nachricht *ideal* statt der Nachricht *real* geschickt (was dann natürlich zu einer Unterscheidung führt).

Zunächst zeigen wir, daß π *nicht* so sicher wie ρ bezüglich allgemeiner komplexitätstheoretischer Sicherheit ohne Auxiliary input bzgl. der Ausgabe der Umgebung ist. Daraus folgt dann mit Lemmata A.5 und A.9 auch die Unsicherheit bezüglich der anderen Varianten der allgemeinen komplexitätstheoretischen Sicherheit.

Für ein Polynom p definieren wir die Umgebung $\mathcal{Z}_p$ wie folgt (vgl. auch Abbildung 5.3a): Die Umgebung hat den ausgehenden Protokolleingabeport `in_timelock`, sowie die eingehenden Protokollausgabeports `out_timelock` und `out_identity`, den eingehenden Umgebungsport `env_start`, und den ausgehenden Spezialport `output`. Die Umgebung $\mathcal{Z}_p$ zeigt folgendes Verhalten:

- Bei Empfang der ersten Nachricht über `env_start` schickt $\mathcal{Z}_p$ eine leere Nachricht über den Protokolleingabeport `in_timelock` (dies aktiviert die Maschine M_{real} bzw. M_{ideal}).

- Es sei C_p die polynomiell-beschränkte ITM, die nach der Einfachheitseigenschaft des TL-Puzzles existiert und TL-Puzzles bis zu einem Schwierigkeitsgrad von $p(k)$ löst. Wir nehmen o. B. d. A. an, daß, selbst wenn der Schwierigkeitsgrad s zu groß für C_p ist, C_p solange Nachrichten schickt, bis der Verifier terminiert (evtl. mit einer Ausgabe ungleich 1).

  Bei Empfang einer Zahl s über den Protokollausgabeport `out_timelock` startet $\mathcal{Z}_p$ eine Instanz $C_p(1^k, s)$ von C_p. Die Kommunikation von C_p wird über die Protokollports `in_timelock` und `out_timelock` geleitet. ($\mathcal{Z}_p$ versucht also, das von M_{real} bzw. M_{ideal} gestellte TL-Puzzle bis zu einem Schwierigkeitsgrad $p(k)$ zu lösen.)

- Sobald $\mathcal{Z}_p$ eine Nachricht m über den Protokollausgabeport `out_identity` erhält, gibt $\mathcal{Z}_p$ diese aus (über `output`) und terminiert.

Passend zu $\mathcal{Z}_p$ konstruieren wir noch den Angreifer $\mathcal{A}$ mit dem eingehenden Spezialport `clk` und dem ausgehenden Umgebungsport `env_start`. $\mathcal{A}$ tut nichts anderes, als bei Empfang einer Nachricht auf `clk` diese über `env_start` an die Umgebung weiterzuleiten.

Offensichtlich sind $\mathcal{Z}_p$ und $\mathcal{A}$ zulässig und polynomiell-beschränkt.

Das Zusammenspiel von $\mathcal{Z}_p$, $\mathcal{A}$ und dem realen Protokoll π ist in Abbildung 5.3a dargestellt. Bei einem Protokollauf von $\mathcal{Z}_p$, $\mathcal{A}$ und π geschieht nun folgendes: Zunächst wird der Angreifer in seiner Eigenschaft als Scheduler aktiviert.

Daraufhin aktiviert er die Umgebung $\mathcal{Z}_p$ über den Umgebungsport `env_start`. Diese wiederum aktiviert M_{real}. M_{real} schickt daraufhin einen Schwierigkeitsgrad s an $\mathcal{Z}_p$, dann versucht $\mathcal{Z}_p$ das TL-Puzzle zu lösen, d. h. $\mathcal{Z}_p$ und M_{real} beginnen eine Interaktion, die damit endet, daß der von M_{real} simulierte Verifier $\mathcal{V}_{\mathcal{Z}}$ terminiert. Je nach dessen Ausgabe sendet dann M_{real} eine Nachricht $m \in \{real, lost\}$ an $\mathcal{Z}_p$. $\mathcal{Z}_p$ gibt m aus. Somit ist (unabhängig von p und k)

$$P\Big(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}_p}(k) \in \{real, lost\}\Big) = 1. \tag{5.5}$$

Nun sei ein polynomiell-beschränkter zulässiger Simulator $\mathcal{S}$ gegeben. Um zu zeigen, daß π nicht so sicher wie ρ bezüglich allgemeiner Sicherheit ist, genügt es zu zeigen, daß für jeden $\mathcal{S}$ ein Polynom p existiert, so daß

$$\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}_p}(k) \quad \text{und} \quad \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}_p}(k)$$

komplexitätstheoretisch unterscheidbar sind.

Wir betrachten dazu zunächst eine Interaktion von $\mathcal{Z}_p$, $\mathcal{S}$ und π (vgl. Abbildung 5.3b) im Falle $p = 0$. Auch hier versucht zunächst $\mathcal{Z}_p$ das durch M_{ideal} von $\mathcal{V}_{\mathcal{Z}}$ gestellte TL-Puzzle zu lösen. Daraufhin (unabhängig davon, ob $\mathcal{S}$ das Puzzle korrekt löst) führt $\mathcal{Z}$ eine Instanz $\mathcal{V}_{\mathcal{S}}(1^k, s)$ von $\mathcal{V}$ aus. Da alle Maschinen polynomiell-beschränkt sind, und s zufällig gewählt wird, gibt es nach der Schwierigkeitsbedingung aus Definition 5.3 ein Polynom $p' \geq 1$, so daß

$$P\Big(s \geq p'(k) \text{ und } \mathcal{V}_{\mathcal{S}}(1^k, s) = 1\Big) \tag{5.6}$$

vernachlässigbar ist. Dabei bezeichne die Zufallsvariable s den von M_{ideal} gewählten Schwierigkeitsgrad, und $\mathcal{V}_{\mathcal{S}}(1^k, s)$ die Ausgabe der von M_{ideal} simulierten ITM $\mathcal{V}_{\mathcal{S}}$.

Weiterhin hängt die Wahrscheinlichkeit in (5.6) nicht von p ab, d. h. auch für $p \neq 0$ ist (5.6) vernachlässigbar (mit dem gleichen p'). Dies liegt daran, daß $\mathcal{V}_{\mathcal{S}}(1^k, s)$ nur davon abhängt, ob $\mathcal{V}_{\mathcal{Z}}$ terminiert hat, nicht aber, welche Ausgabe $\mathcal{V}_{\mathcal{Z}}$ getroffen hat (sprich, ob $\mathcal{V}_{\mathcal{Z}}$ die Lösung von $\mathcal{Z}_p$ akzeptiert hat).

Es sei nun $p := 2p'$. Dann ist auch in diesem Fall (5.6) vernachlässigbar. Weiterhin ist in diesem Fall

$$P\Big(s \leq p(k) = 2p'(k) \text{ und } \mathcal{V}_{\mathcal{Z}}(1^k, s) = 0\Big) \tag{5.7}$$

vernachlässigbar, da $\mathcal{Z}_p$ TL-Puzzles bis zu einem Schwierigkeitsgrad s mit überwältigender Wahrscheinlichkeit löst. Dabei meinen wir mit $\mathcal{V}_{\mathcal{Z}}(1^k, s)$ die Ausgabe der von M_{ideal} simulierten ITM $\mathcal{V}_{\mathcal{Z}}$.

Nun gibt die Umgebung höchstens dann ein $m \in \{real, lost\}$ aus, wenn $\mathcal{V}_{\mathcal{S}}(1^k, s) = 1$ oder $\mathcal{V}_{\mathcal{Z}}(1^k, s) = 0$ ist (wenn der Simulator gewinnt oder die Umgebung versagt). Daher folgt mit (5.6) und (5.7), daß

$$\mu(k) := P\Big(p'(k) \leq s \leq 2p'(k) \text{ und } \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}_p}(k) \in \{real, lost\}\Big)$$

vernachlässigbar ist.

Nun liegt aber (für hinreichend großes k) mindestens ein $s \in \{2^1, \ldots, 2^k\}$ im Bereich $p'(k) \leq s \leq 2p'(k)$. Also ist die Wahrscheinlichkeit, daß ein von M_{ideal} gezogenes s in diesem Bereich liegt, mindestens $\frac{1}{k}$. Damit ist

$$\begin{aligned}
&P\Big(\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}_p}(k) \in \{real, lost\}\Big) \\
&\quad = P\Big(p'(k) \leq s \leq 2p'(k) \text{ und } \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}_p}(k) \in \{real, lost\}\Big) \\
&\qquad + P\Big(s \notin \{p'(k), \ldots, 2p'(k)\} \text{ und } \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}_p}(k) \in \{real, lost\}\Big) \\
&\quad \leq \mu + 1 - \frac{1}{k},
\end{aligned}$$

was nicht überwältigend ist.

Zusammen mit (5.5) folgt die komplexitätstheoretische Unterscheidbarkeit von

$$\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}_p}(k) \quad \text{und} \quad \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}_p}(k),$$

also ist π nicht so sicher wie ρ bezüglich allgemeiner komplexitätstheoretischer Sicherheit.

Wir zeigen nun, daß π so sicher wie ρ ist bezüglich spezieller komplexitätstheoretischer Sicherheit mit Auxiliary input bzgl. der Sicht der Umgebung. Mit Lemmata A.5 und A.9 folgt daraus auch die Sicherheit bezüglich der anderen Varianten der speziellen Sicherheit.

Hierzu werden wir eine Familie von Simulatoren $\mathcal{S}_p$ angeben, so daß zu jeder Umgebung ein p existiert, so daß $\mathcal{S}_p$ ein erfolgreicher Simulator ist (d. h. daß die Umgebung nicht unterscheiden kann).

Es sei also $\mathcal{S}_p$ für ein Polynom p der wie folgt definierte Simulator (vgl. Abbildung 5.4): $\mathcal{S}_p$ hat den eingehenden Angreiferport `adv_out_timelock` und den ausgehenden Angreiferport `adv_in_timelock`. $\mathcal{S}_p$ hat das folgende Programm:[20]

- Es sei C_p wieder die ITM, die TL-Puzzles bis zu einem Schwierigkeitsgrad s löst (vgl. die Konstruktion von $\mathcal{Z}_p$). Bei Empfang einer Zahl s über den Angreiferport `adv_out_timelock` startet $\mathcal{S}_p$ eine Instanz $C_p(1^k, s)$ von C_p. Die Kommunikation von C_p wird über die Angreiferports `adv_in_timelock` und `adv_out_timelock` geleitet. ($\mathcal{S}_p$ versucht also, das von M_{ideal} gestellte TL-Puzzle bis zu einem Schwierigkeitsgrad $p(k)$ zu lösen.)

Nun seien eine zulässige polynomiell-beschränkte Umgebung $\mathcal{Z}$ und ein zulässiger polynomiell-beschränkter Angreifer $\mathcal{A}$ gegeben. Um spezielle Sicherheit

[20] In der hier vorgestellten Form ist $\mathcal{S}_p$ *kein* zulässiger Simulator, da $\mathcal{S}_p$ keinen eingehenden Port `clk` hat und somit nicht Scheduler ist. Wir werden $\mathcal{S}_p$ aber weiter unten durch Kombination mit einem Angreifer $\mathcal{A}$ zu einem zulässigen Simulator machen.

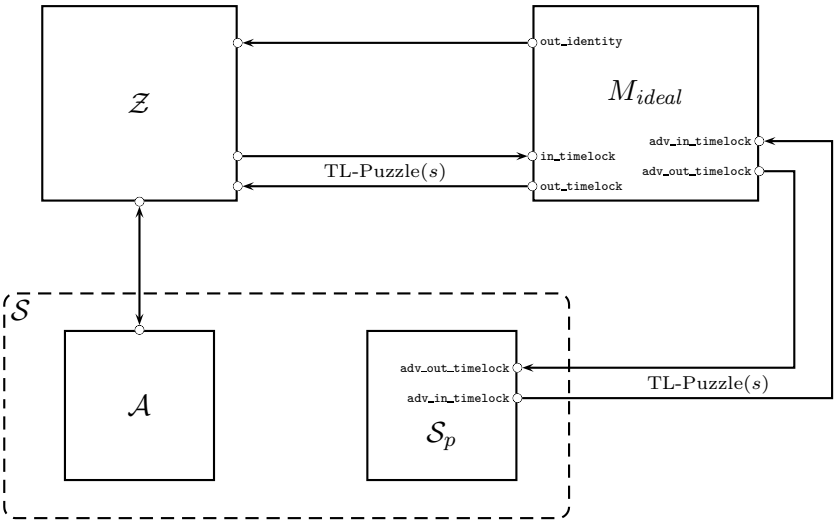


Abbildung 5.4.: Beweis der Sicherheit bezüglich spezieller Sicherheit.
Der Simulator $\mathcal{S}$ besteht aus dem Angreifer $\mathcal{A}$ und dem Simulator $\mathcal{S}_p$. Letzterer löst alle TL-Puzzle, die von der idealen Protokollmaschine M_{ideal} gelöst werden, sofern sie einen Schwierigkeitsgrad $s \leq p(k)$ haben. Wenn p groß genug gewählt ist, bedeutet dies, daß $\mathcal{S}_p$ alle TL-Puzzle löst, die die Umgebung $\mathcal{Z}$ lösen kann.

zu zeigen, müssen wir nun also einen (von $\mathcal{Z}$ und $\mathcal{A}$ abhängigen) polynomiell-beschränkten zulässigen Simulator $\mathcal{S}$ angeben, so daß

$$\text{CVIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) \quad \text{und} \quad \text{CVIEW}_{\rho,\mathcal{S},\mathcal{Z}}(k,z) \tag{5.8}$$

komplexitätstheoretisch ununterscheidbar sind.

O. B. d. A. können wir annehmen, der Angreifer $\mathcal{A}$ habe keine Ports mit den Namen `adv_in_timelock` und `adv_out_timelock`.

Zunächst betrachten wir das aus $\rho = \{M_{ideal}\}$, $\mathcal{Z}$, $\mathcal{A}$ und $\mathcal{S}_p$ bestehende Netzwerk (Abbildung 5.4). In leichter Erweiterung unserer Notation bezeichne $\text{CVIEW}_{\rho,\mathcal{A},\mathcal{S}_p,\mathcal{Z}}(k,z)$ die Sicht der Umgebung in einer Ausführung dieses Netzwerks bei Sicherheitsparameter k und Auxiliary input z.

Es sei $p := 0$. Ganz analog wie oben erkennen wir, daß ein Polynom p' existiert (da $\mathcal{A}$ und $\mathcal{Z}$ polynomiell-beschränkt sind), so daß in einer Ausführung dieses Netzwerks

$$\mu_{\mathcal{Z}} := P\Big(s \geq p'(k) \text{ und } \mathcal{V}_{\mathcal{Z}}(1^k, s) = 1\Big)$$

vernachlässigbar ist. Weiterhin sind p' und diese Wahrscheinlichkeit wieder unabhängig von p (da p erst verwendet wird, nachdem $\mathcal{V}_{\mathcal{Z}}(1^k, s)$ festliegt), so daß dies insbesondere für $p := p'$ gilt.

Weiterhin gilt nach Konstruktion von $\mathcal{S}_p$ und M_{ideal} und Definition von C_p, daß auch

$$\mu_{\mathcal{S}} := P\Big(s \leq p(k) \text{ und } \mathcal{V}_{\mathcal{S}}(1^k, s) = 0\Big)$$

vernachlässigbar ist. Es folgt (mit $p = p'$), daß auch

$$\mu := P\Big(\mathcal{V}_{\mathcal{Z}}(1^k, s) = 1 \text{ und } \mathcal{V}_{\mathcal{S}}(1^k, s) = 0\Big) \leq \mu_{\mathcal{Z}} + \mu_{\mathcal{S}}$$

vernachlässigbar ist. (Denn aus $A \wedge B$ folgt $(s \geq p \wedge A) \vee (s \leq p \wedge B)$.)

Wir können also M_{ideal} dahingehend ändern, daß die Nachricht id, die über den Protokollausgabeport `out_identity` gesandt wird, im Falle $\mathcal{V}_{\mathcal{Z}} = 1$ immer $id = real$ ist (statt, wie zuvor $id = ideal$ für $\mathcal{V}_{\mathcal{S}} = 0$). Die resultierende Maschine nennen wir M'_{ideal}. Da diese Änderung nur den Fall $\mathcal{V}_{\mathcal{Z}} = 1 \wedge \mathcal{V}_{\mathcal{S}} = 0$ betrifft, der höchstens mit der vernachlässigbaren Wahrscheinlichkeit μ eintritt, sind

$$\text{CVIEW}_{\rho,\mathcal{A},\mathcal{S}_p,\mathcal{Z}}(k,z) \quad \text{und} \quad \text{CVIEW}_{M'_{ideal},\mathcal{A},\mathcal{S}_p,\mathcal{Z}}(k,z)$$

komplexitätstheoretisch ununterscheidbar.

Nun wird das Ergebnis $\mathcal{V}_{\mathcal{S}}$ im Programm von M'_{ideal} nie verwendet. Wir können also M'_{ideal} weiter dahingehend verändern, daß das TL-Puzzle mit Prover $\mathcal{S}_p$ gar nicht durchgeführt wird (man beachte, daß nach Definition von C_p und $\mathcal{V}_{\mathcal{S}}$ die Interaktion mit $\mathcal{S}_p$ immer terminiert). Die resultierende Maschine nennen wir M''_{ideal}. Es ist

$$\text{CVIEW}_{M'_{ideal},\mathcal{A},\mathcal{S}_p,\mathcal{Z}}(k,z) = \text{CVIEW}_{M''_{ideal},\mathcal{A},\mathcal{S}_p,\mathcal{Z}}(k,z).$$

Da M''_{ideal} nun die Angreiferports `adv_in_timelock` und `adv_out_timelock` nicht mehr benutzt, können wir diese Ports entfernen. Die resultierende Maschine ist aber gerade M_{real}. Weiterhin können wir die Maschine $\mathcal{S}_p$, die dann mit keiner anderen Maschine mehr verbunden ist, aus dem Netzwerk entfernen. Wir erhalten damit:

$$\mathrm{CVIEW}_{M''_{ideal},\mathcal{A},\mathcal{S}_p,\mathcal{Z}}(k,z) = \mathrm{CVIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k,z).$$

Kombinieren wir die letzten drei Gleichungen, so erhalten wir, daß

$$\mathrm{CVIEW}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) \quad \text{und} \quad \mathrm{CVIEW}_{\rho,\mathcal{A},\mathcal{S}_p,\mathcal{Z}}(k,z) \tag{5.9}$$

komplexitätstheoretisch ununterscheidbar sind.

Schließlich definieren wir $\mathcal{S}$ als den aus $\mathcal{S}_p$ und $\mathcal{A}$ bestehenden Simulator. In anderen Worten, $\mathcal{S}$ hat die Ports von $\mathcal{S}_p$ und $\mathcal{A}$, simuliert $\mathcal{S}_p$ und $\mathcal{A}$ und leitet alle Kommunikation über die Ports zu der jeweiligen Submaschine weiter. Da $\mathcal{S}_p$ und $\mathcal{A}$ polynomiell-beschränkt sind, und $\mathcal{A}$ ein zulässiger Angreifer ist, der keine Ports `adv_out_timelock` oder `adv_in_timelock` hat, sieht man leicht, daß auch $\mathcal{S}$ polynomiell-beschränkt und zulässig ist.

Ersetzen wir $\mathcal{S}_p$ und $\mathcal{A}$ durch $\mathcal{S}$, ändert sich die Sicht von $\mathcal{Z}$ offensichtlich nicht, und aus (5.9) ergibt sich (5.8). Damit ist π so sicher wie ρ bezüglich allgemeiner komplexitätstheoretischer Sicherheit, und der Satz ist bewiesen. □

6. Verdeckte Time-lock puzzles und Komposition

Im vorangegangenen Kapitel haben wir gelernt, daß allgemeine und spezielle Sicherheit tatsächlich verschiedene Begriffe sind. Doch dies allein impliziert noch keine negative Antwort auf die Frage, ob spezielle Sicherheit hinreichend für allgemeine Komposition ist. Im Gegenteil, wir haben in Kapitel 4 gesehen, daß bereits *statistische* spezielle Sicherheit hinreichend für allgemeine Komposition ist. In diesem Kapitel wenden wir uns der Frage zu, ob *im komplexitätstheoretischen Fall* spezielle Sicherheit hinreichend für allgemeine Komposition ist. Diese werden wir – unter gewissen Komplexitätsannahmen – negativ beantworten.

6.1. Allgemeine Komponierbarkeit impliziert nicht allgemeine Sicherheit

Zunächst wollen wir untersuchen, ob nicht vielleicht die trennenden Beispiele aus dem vorangegangenen Kapitel, insbesondere aus Abschnitt 5.4.3, bereits auch ein trennendes Beispiel für allgemeine Komponierbarkeit und spezielle Sicherheit darstellen. Diese Frage können wir darauf reduzieren, ob die im Beweis von Satz 5.10 konstruierten Protokolle π und ρ nebenläufig komponieren, sprich, ob für jedes polynomiell-beschränkte f gilt, daß $f \cdot \pi$ so sicher wie $f \cdot \rho$ ist ($f \cdot \pi$ und $f \cdot \rho$ bezeichnen die f-fache nebenläufige Komposition, vgl. Definition 2.24). Leider kann jedoch der Beweis aus Abschnitt 5.4.3 für die spezielle Sicherheit dieser Protokolle relativ einfach auf den nebenläufig komponierten Fall erweitert werden:

Lemma 6.1 (Nebenläufige Komposition der Protokolle π und ρ) Für die Protokolle π und ρ aus dem Beweis von Satz 5.10 und jede polynomiell-beschränkte effizient berechenbare Funktion f gilt:

- Das nebenläufig komponierte Protokoll $f \cdot \pi$ ist so sicher wie $f \cdot \rho$ bezüglich *spezieller* komplexitätstheoretischer Sicherheit sowohl mit als auch ohne Auxiliary input sowohl bzgl. der Sicht als auch bzgl. der Ausgabe der Umgebung.

Beweis: Wegen Lemmata A.5 und A.9 ist es hinreichend, den Fall der Sicherheit mit Auxiliary input bzgl. der Sicht der Umgebung zu betrachten. Im folgenden sei Sicherheit deshalb immer Sicherheit mit Auxiliary input bzgl. der Sicht der Umgebung.

Es seien also $\pi = \{M_{real}\}$ und $\rho = \{M_{ideal}\}$ wie im Beweis von Satz 5.10.

Wir wollen nun zeigen, daß $f \cdot \pi$ so sicher wie $f \cdot \rho$ ist. Da der Beweis der zweiten Hälfte des Beweises von Satz 5.10 sehr ähnlich ist, werden wir im folgenden oft auf letztere zurückgreifen. (Dort wurde gezeigt, daß π so sicher wie ρ ist.)

Zunächst sei $\mathcal{S}_p$ definiert wie im Beweis von Satz 5.10 (d. h. $\mathcal{S}_p$ löst von M_{ideal} gelöste TL-Puzzles bis zu einem Schwierigkeitsgrad von $p(k)$). Dann bezeichne $\mathcal{S}_{p,f} := f \cdot \mathcal{S}_p$ die f-fache nebenläufige Komposition von $\mathcal{S}_p$ (gemäß Definition 2.24). $\mathcal{S}_{p,f}$ löst also für jede von $f \cdot M_{ideal}$ simulierte Instanz von M_{ideal} das von dieser Instanz gestellte TL-Puzzle, sofern der Schwierigkeitsgrad nicht größer als $p(k)$ ist.

Nun seien eine zulässige polynomiell-beschränkte Umgebung $\mathcal{Z}$ und ein zulässiger polynomiell-beschränkter Angreifer $\mathcal{A}$ gegeben. Um spezielle Sicherheit zu zeigen, müssen wir nun also einen (von $\mathcal{Z}$ und $\mathcal{A}$ abhängigen) polynomiell-beschränkten zulässigen Simulator $\mathcal{S}$ angeben, so daß

$$\mathrm{CVIEW}_{f\cdot\pi,\mathcal{A},\mathcal{Z}}(k,z) \quad \text{und} \quad \mathrm{CVIEW}_{f\cdot\rho,\mathcal{S},\mathcal{Z}}(k,z) \tag{6.1}$$

komplexitätstheoretisch ununterscheidbar sind.

O. B. d. A. können wir annehmen, der Angreifer $\mathcal{A}$ habe keine Ports mit den Namen `adv_in_timelock` und `adv_out_timelock`.

Zunächst betrachten wir das aus $f \cdot \rho = \{f \cdot M_{ideal}\}$, $\mathcal{Z}$, $\mathcal{A}$ und $\mathcal{S}_{p,f}$ bestehende Netzwerk (ähnlich wie in Abbildung 5.4). In leichter Erweiterung unserer Notation bezeichne $\mathrm{CVIEW}_{f\cdot\rho,\mathcal{A},\mathcal{S}_{p,f},\mathcal{Z}}(k,z)$ die Sicht der Umgebung in einer Ausführung dieses Netzwerks bei Sicherheitsparameter k und Auxiliary input z.

Wir wollen nun zeigen, daß ein Polynom p' existiert, so daß für alle Polynome p die folgende Wahrscheinlichkeit vernachlässigbar ist:

$$\mu^p_{\mathcal{Z}} := P\Big(\exists i \in \{1,\ldots,f(k)\} : s^{(i)} \geq p'(k) \text{ und } \mathcal{V}^{(i)}_{\mathcal{Z}}(1^k, s^{(i)}) = 1 \\ \text{in einer Ausführung des Netzwerks } \{f \cdot M_{ideal}, \mathcal{Z}, \mathcal{A}, \mathcal{S}_{p,f}\}\Big). \tag{6.2}$$

Hierbei bezeichne $s^{(i)}$ den von der i-ten Instanz von M_{ideal} gewählten Schwierigkeitsgrad s, und $\mathcal{V}^{(i)}_{\mathcal{Z}}(1^k, s^{(i)})$ die Ausgabe der von der i-ten Instanz von M_{ideal} simulierten ITM $\mathcal{V}_{\mathcal{Z}}$ (also $\mathcal{V}^{(i)}_{\mathcal{Z}}(1^k, s^{(i)}) = 1$ wenn $\mathcal{Z}$ das i-te TL-Puzzle löst).

Im Beweis von Satz 5.10 haben wir dies nachgewiesen, indem wir zunächst die Existenz eines p' gezeigt haben, so daß $\mu^p_{\mathcal{Z}}$ im Spezialfall $p = 0$ vernachlässigbar ist. Und dann haben wir eingesehen, daß diese Wahrscheinlichkeit unabhängig von p ist, da $\mathcal{S}_p$ das Polynom p erst verwendet, wenn die Ausgabe von $\mathcal{V}_{\mathcal{Z}}$

bereits feststeht. Dieser Ansatz funktioniert hier jedoch nicht, da hier, nachdem die Umgebung $\mathcal{Z}$ das erste TL-Puzzle gelöst hat, der Simulator $\mathcal{S}$ versucht, ein TL-Puzzle der gleichen Schwierigkeit zu lösen. Dabei fließt p ein. Wenn also $\mathcal{Z}$ das zweite TL-Puzzle löst, können wir nicht mehr davon ausgehen, daß $\mathcal{Z}$'s Strategie unabhängig von p ist.

Daher müssen wir einen anderen, leicht komplexeren Weg einschlagen. Wir nehmen o. B. d. A. an, daß die Maschine C_p, die von $\mathcal{S}_p$ simuliert wird, TL-Puzzles mit einem Schwierigkeitsgrad $s > p(k)$ nie, also mit Wahrscheinlichkeit 0 löst. Weiterhin löst C_p und damit $\mathcal{S}_p$ Puzzles des Schwierigkeitsgrads $s \leq p(k)$ mit überwältigender Wahrscheinlichkeit. Wir definieren nun M^p_{ideal} als die Maschine, die, anstatt ein i-tes TL-Puzzles mit $\mathcal{S}_p$ durchzuführen, annimmt, daß $\mathcal{V}^{(i)}_{\mathcal{S}}(1^k, s^{(i)}) = 1$ genau dann, wenn $s \leq p(k)$. Weiterhin habe M^p_{ideal} die Ports `adv_in_timelock` und `adv_out_timelock` nicht mehr. Die folgenden Zufallsvariablen sind dann komplexitätstheoretisch ununterscheidbar:

$$\mathrm{CVIEW}_{f\cdot\rho,\mathcal{A},\mathcal{S}_{p,f},\mathcal{Z}}(k,z) \quad \text{und} \quad \mathrm{CVIEW}_{f\cdot M^p_{ideal},\mathcal{A},\mathcal{Z}}(k,z). \tag{6.3}$$

Anstatt (6.2) zu zeigen, genügt es uns also zu beweisen, daß ein Polynom p' existiert, so daß für jedes Polynom p die folgende Wahrscheinlichkeit vernachlässigbar ist:

$$P\Big(\exists i \in \{1,\ldots,f(k)\} : s^{(i)} > p'(k) \text{ und } \mathcal{V}^{(i)}_{\mathcal{Z}}(1^k, s^{(i)}) = 1 \\ \text{in einer Ausführung des Netzwerks } \{f\cdot M^p_{ideal}, \mathcal{Z}, \mathcal{A}\}\Big). \tag{6.4}$$

Der wesentliche Unterschied zur Situation in (6.2) ist, daß es ein die Komplexität der Maschinen $\{f \cdot M^p_{ideal}, \mathcal{Z}, \mathcal{A}\}$ beschränkendes Polynom gibt, das nicht von p abhängt. Damit können wir das Polynom p auch als Auxiliary input auffassen, wir hätten damit eine polynomiell-beschränkte Maschine, die für jedes Polynom p' mit nicht vernachlässigbarer Wahrscheinlichkeit ein TL-Puzzle mit Schwierigkeitsgrad größer p' löst. Man sieht leicht, daß dies im Widerspruch zur Schwierigkeitsbedingung des TL-Puzzles steht (vgl. Definition 5.3).

Es sei nun $p := p'$. Die i-te Instanz von M^p_{ideal} schickt nach Konstruktion nur dann die Nachricht $id = ideal$ an die Umgebung, wenn die simulierte ITM $\mathcal{V}^{(i)}_{\mathcal{Z}}$ die Ausgabe 1 hat und $s^{(i)} > p(k)$. Also ist die Wahrscheinlichkeit für diese Nachricht nach (6.4) vernachlässigbar. Wir verändern nun M^p_{ideal} dahingehend, daß statt $id = ideal$ in diesem Fall $id = real$ gesandt wird. Die resultierende Maschine ist gerade M_{real}. Wir haben also, daß

$$\mathrm{CVIEW}_{f\cdot M^p_{ideal},\mathcal{A},\mathcal{Z}}(k,z) \quad \text{und} \quad \mathrm{CVIEW}_{f\cdot M_{real},\mathcal{A},\mathcal{Z}}(k,z)$$

komplexitätstheoretisch ununterscheidbar sind. Zusammen mit (6.3) sind also auch

$$\mathrm{CVIEW}_{f\cdot\rho,\mathcal{A},\mathcal{S}_{p,f},\mathcal{Z}}(k,z) \quad \text{und} \quad \mathrm{CVIEW}_{f\cdot\pi,\mathcal{A},\mathcal{Z}}(k,z)$$

komplexitätstheoretisch ununterscheidbar. Und wenn wir $\mathcal{S}$ als Kombination von $\mathcal{S}_{p,f}$ und $\mathcal{A}$ definieren, ergibt sich (6.1) und das Lemma ist bewiesen. □

Wir sehen also, daß das trennende Beispiel aus Abschnitt 5.4.3 nebenläufig komponiert und somit nicht geeignet ist, um allgemeine Komponierbarkeit und spezielle Sicherheit zu trennen. Doch auch dieses negative Ergebnis hat einen Wert, sagt es uns doch, daß allgemeine Sicherheit zwar hinreichend (Lemma 2.29), aber nicht notwendig für allgemeine Komponierbarkeit ist:

Korollar 6.2 (Allgemeine Sicherheit und allgemeine Komponierbarkeit sind verschieden)
Wenn Time-lock puzzles (Definition 5.3) existieren, dann gibt es *polynomiell-beschränkte* Protokolle π und ρ, so daß folgendes gilt:

- Das Protokoll π ist so sicher wie ρ bezüglich komplexitätstheoretischer polynomiell-beschränkter allgemeiner Komponierbarkeit sowohl mit als auch ohne Auxiliary input.
- Das Protokoll π ist *nicht* so sicher wie ρ bezüglich allgemeiner komplexitätstheoretischer Sicherheit weder mit noch ohne Auxiliary input und weder bzgl. der Sicht noch bzgl. der Ausgabe der Umgebung.

Beweisskizze: Es seien π und ρ wie in Satz 5.10 und Lemma 6.1. Wie in der Beweisskizze des allgemeinen Kompositionstheorems 2.27 folgt aus Lemma 6.1 die polynomiell-beschränkte allgemeine Komponierbarkeit mit und ohne Auxiliary input. Die Unsicherheit bezüglich der verschiedenen Varianten allgemeiner komplexitätstheoretischer Sicherheit liefert Satz 5.10. □

6.2. Die Beweisidee

Wir wollen nun also folgenden Satz beweisen:

Satz 6.3 (Spezielle Sicherheit genügt nicht für nebenläufige Komposition)
Wenn Time-lock puzzles (Definition 5.3) und Enhanced trapdoor permutations (Definition 1.9) existieren, dann gibt es *polynomiell-beschränkte* Protokolle π und ρ, so daß folgendes gilt:

- Das Protokoll π ist so sicher wie ρ bezüglich *spezieller* komplexitätstheoretischer Sicherheit sowohl mit als auch ohne Auxiliary input sowohl bzgl. der Sicht als auch bzgl. der Ausgabe der Umgebung.

(Fortsetzung nächste Seite)

(Fortsetzung)

- Für jede unbeschränkte effizient berechenbare Funktion f ist $f \cdot \pi$ *nicht* so sicher wie $f \cdot \rho$ bezüglich spezieller komplexitätstheoretischer Sicherheit weder mit noch ohne Auxiliary input und weder bzgl. der Sicht noch bzgl. der Ausgabe der Umgebung.

Da der Beweis dieses Satzes (der erst in Abschnitt 6.8 vollendet wird) relativ komplex ist, und selbst die Beweisidee nicht unmittelbar durchsichtig ist, wollen wir letztere nun zunächst in groben Zügen herleiten, bevor wir in den nächsten Abschnitten die Details des Beweises vorführen.

- Die Grundidee des Beweises ist die folgende: Jede Instanz des idealen Protokolls ρ stellt dem Simulator ein TL-Puzzle. Wir wollen dabei erreichen, daß der Simulator um so schwierigere TL-Puzzles lösen muß, je mehr Instanzen des idealen Protokolls ρ existieren. Genauer soll die Schwierigkeit der zu lösenden TL-Puzzles superpolynomiell mit der Anzahl der Instanzen des idealen Protokolls ρ steigen.

  Im Detail sieht dies wie folgt aus: Gegenüber jeder Instanz ρ_i von ρ darf der Simulator $\mathcal{S}$ einen Schwierigkeitsgrad s_i wählen. Die ρ_i tauschen dann die folgenden Informationen aus: Welche Werte haben die s_i, und welche TL-Puzzles hat der Simulator gelöst. Der Simulator gilt als erfolgreich, wenn (i) er alle TL-Puzzles gelöst hat und (ii) für die Schwierigkeitsgrade gilt: $s_i \geq k^i$ für alle i (für irgendeine Sortierung der s_i). Wenn der Simulator nicht erfolgreich ist, geben die Instanzen ρ_i eine spezielle Ausgabe, die der Umgebung erlaubt, zu unterscheiden (ansonsten verhält sich ρ_i aus Sicht der Umgebung wie π_i).

  Es ist nun π so sicher wie ρ, da – wenn nur eine Instanz von ρ vorliegt – der Simulator $\mathcal{S}$ nur ein TL-Puzzle der Schwierigkeit $s_1 = 2^1$ lösen muß.

  Betrachten wir aber $f \cdot \rho$, also f Instanzen ρ_i, so muß der Simulator, um erfolgreich zu sein, ein TL-Puzzle der Schwierigkeit k^f lösen, was superpolynomiell und damit zu aufwendig ist. Somit ist $f \cdot \pi$ nicht so sicher wie $f \cdot \rho$.

- Mit diesem Ansatz stoßen wir aber auf ein großes Problem: Bei der nebenläufigen Komposition erlauben wir es mehreren Protokollinstanzen nämlich gar nicht, direkt miteinander zu kommunizieren. Vielmehr darf jede Instanz nur mit der Umgebung und mit dem Simulator/Angreifer kommunizieren. Bevor wir dieses Problem weiter unten lösen, treffen wir zunächst provisorisch folgende (falsche) Annahme: Es steht eine weitere Maschine

F zur Verfügung, mit der die Protokollinstanzen kommunizieren können. Dann senden alle Protokollinstanzen die Schwierigkeitsgrade s_i, und ob die TL-Puzzles gelöst wurden an F, und F schickt dann an die Umgebung die entsprechende Ausgabe: Wenn alle TL-Puzzles gelöst wurden und die Schwierigkeitsgrade hoch genug sind (s. o.), dann wird *success* ausgegeben, ansonsten *failed*.

Reale Protokollinstanzen π_i kommunizieren auch mit F, da diese aber keine TL-Puzzles stellen, haben diese keinen Einfluß auf die Ausgabe von F.

Wir nehmen dabei an, daß die Umgebung die Kommunikation zwischen F und den Instanzen nicht abhören kann, da sonst diese Kommunikation bereits zur Unterscheidung hinreicht.

- Aus technischen Gründen, die wir erst weiter unten kennenlernen werden, müssen wir davon ausgehen, daß die Umgebung bestimmen kann, welche der Protokollinstanzen mit F kommunizieren können. Insbesondere kann die Umgebung einige Instanzen von der Kommunikation ausschließen (was aber nur zum Nachteil der Umgebung ist, da der Simulator dann weniger schwierige TL-Puzzles lösen muß), und die Umgebung $\mathcal{Z}$ kann neue Instanzen von ρ_i simulieren und mit F kommunizieren lassen. Dadurch steigt natürlich der Schwierigkeitsgrad der TL-Puzzles und der Simulator kann diese nun nicht mehr unbedingt lösen und damit nicht die Ausgabe *success* erzwingen.

  Wir können aber einsehen, daß dies nicht weiter schlimm ist: Denn um zwischen unkomponiertem realem Protokoll π und idealem Protokoll ρ unterscheiden zu können, genügt es der Umgebung nicht, daß F im idealen *failed* ausgibt. Es ist zusätzlich nötig, daß im realen *success* ausgegeben wird. Simuliert $\mathcal{Z}$ nun also n Protokollinstanzen ρ_i, so muß $\mathcal{Z}$ auch die zugehörigen TL-Puzzles bis zu einem Schwierigkeitsgrad $s_n = k^n$ lösen (selbst im realen). Es existiert aber eine polynomielle Schranke q, so daß die Umgebung TL-Puzzles größer als q nicht mehr lösen kann (nach der Schwierigkeitsbedingung der TL-Puzzles). Der Simulator kann also davon ausgehen, daß $k^n \leq q$. Damit ist $k^{n+1} \leq kq$ auch polynomiell-beschränkt, und es genügt also, wenn der Simulator ein TL-Puzzle der Schwierigkeit kq löst, um die Ausgabe *success* zu erhalten (außer wenn bereits real *failed* auftritt).

  Hier nutzen wir übrigens die Tatsache, daß wir spezielle Sicherheit betrachten: Der Simulator muß abhängig von der Umgebung gewählt werden, da er TL-Puzzles der Schwierigkeit kq lösen muß, wobei q von der Umgebung abhängt. Da allgemeine Sicherheit hinreichend für nebenläufige Komposition ist, steht auch zu erwarten, daß wir diese Tatsache irgendwo im Beweis ausnutzen.

- Die Tatsache, daß die Umgebung $\mathcal{Z}$ zusätzliche ideale Protokollinstanzen ρ_i simulieren kann, wirft noch ein weiteres Problem auf, das die Überlegungen des vorangegangenen Punktes noch nicht lösen. Die Umgebung kann nicht nur ideale Instanzen ρ_i hinzufügen, sondern ist auch nicht gezwungen, diese ehrlich zu simulieren. So kann die Umgebung ideale Instanzen ρ_i simulieren, die F gegenüber behaupten, ein TL-Puzzle sehr hoher Schwierigkeit sei gelöst worden. Diesen Schwierigkeitsgrad muß der Simulator dann überbieten, was ihm nicht möglich sein wird.

  Dies zu verhindern ist jedoch nicht weiter schwierig: Die TL-Puzzles werden von F gestellt und überprüft, und die Protokollinstanzen ρ_i leiten die TL-Puzzles lediglich durch. Nun ist es der Umgebung nicht mehr möglich, durch unehrlich simulierte ideale Protokollinstanzen das Ergebnis zu beeinflussen. (Eine simulierte unehrliche ideale Instanz ρ_i kann zwar F gegenüber behaupten, sie sei eine reale Instanz π_i, aber diese haben sowieso keinen Einfluß auf das Ergebnis.)

- Unter der Annahme, daß es eine vermittelnde Maschine F gibt, haben wir also ein trennendes Beispiel konstruiert. Leider steht uns eine solche Maschine F nicht zur Verfügung. Dieses Problem lösen wir, indem wir anstelle der Maschine F eine sichere Funktionsauswertung verwenden, die die Ausgaben der Maschine F berechnet. Eine solche sichere Funktionsauswertung hat die Eigenschaft, daß, selbst wenn man die Kommunikation zwischen den Parteien abhören und modifizieren kann, man keine Möglichkeit hat, (i) die Eingaben der Parteien zu erfahren, oder (ii) das Ergebnis zu modifizieren. Es ist den Protokollinstanzen daher möglich, die für die sichere Funktionsauswertung notwendigen Nachrichten über die Umgebung zu schicken, d. h. eine Protokollinstanz A, die an eine andere Instanz B im Rahmen der Funktionsauswertung eine Nachricht schicken möchte, liefert diese an die Umgebung $\mathcal{Z}$ mit der Bitte um Auslieferung an B.

  Die Umgebung hat nun natürlich die Möglichkeit, die Kommunikation beliebig zu unterdrücken oder zu modifizieren. Die Eigenschaften der Funktionsauswertung garantieren jedoch, daß die Umgebung dadurch nichts weiter erreichen kann, als die Funktionsauswertung abzubrechen, was nicht in ihrem Interesse ist (denn zum Unterscheiden braucht die Umgebung die Ausgabe von F, also das Ergebnis der Funktionsauswertung). Darüber hinaus kann die Umgebung natürlich bestimmen, welche Protokollinstanzen an der Funktionsauswertung teilnehmen dürfen, und sogar eigene simulierte Instanzen hinzufügen, aber diese Angriffspunkte haben wir bereits oben diskutiert und als harmlos erkannt.

Wir fassen also kurz zusammen, wie unser trennendes Beispiel aussieht: Jede Protokollinstanz π_i oder ρ_i (bestehend aus nur einer Maschine) nimmt an einer

sicheren Funktionsauswertung teil. Die für diese Funktionsauswertung notwendigen Nachrichten werden über $\mathcal{Z}$ versandt.

Die ausgewertete (reaktive) Funktion F erwartet zunächst von jeder Protokollinstanz als Eingaben: Ein Flag b_i, welches angibt, ob die Protokollinstanz eine reale oder eine ideale ist, sowie, im Falle einer idealen Instanz, einen Schwierigkeitsgrad s_i. Dann stellt F jeder idealen Protokollinstanz ρ_i ein TL-Puzzle mit Schwierigkeitsgrad s_i und überprüft die Lösungen. Nur wenn alle TL-Puzzles gelöst wurden, und $s_i \geq k^i$ für alle i (für irgendeine Reihenfolge der s_i), dann gibt F *success* aus, sonst *failed*.

Die idealen Protokollinstanzen leiten dabei alle Ein-/Ausgaben an den Simulator weiter *mit Ausnahme des Flags, welches angibt, daß eine ideale Instanz vorliegt.* Reale Protokollinstanzen geben F gegenüber an, daß eine reale Instanz vorliegt und F erwartet dann keine weiteren Eingaben. Sobald das Endergebnis der Funktionsauswertung festliegt (*success* oder *failed*), leiten die Protokollinstanzen diese an die Umgebung weiter, welche dies benutzen kann, um zu versuchen, reales und ideales Protokoll zu unterscheiden.

In den folgenden Abschnitten werden wir die Details dieser Konstruktion ausarbeiten und beweisen.

6.3. Die reaktive Funktion

Wir beschreiben nun im Detail die reaktive Funktion F, die im vorangegangenen Abschnitt bereits grob skizziert wurde.

Unter einer reaktiven Funktion verstehen wir folgendes: F besteht aus mehreren Runden, deren Anzahl vom Sicherheitsparameter abhängen kann, aber für gegebenen Sicherheitsparameter fest ist. Jede Runde r ist durch eine probabilistische Funktion F_r gegeben. Diese Funktion F_r nimmt als Argument den Sicherheitsparameter k, den internen Zustand $w_r \in \Sigma^*$ der reaktiven Funktion (in der ersten Runde ist $w_1 = \lambda$), und für jede Partei i eine Eingabe $I_r^i \in \Sigma^*$ (die r-te Eingabe von Partei i). Die Funktion F_r liefert dann folgendes zurück: Einen neuen internen Zustand w_{i+1}, und für jede Partei eine Ausgabe O_r^i (die r-te Ausgabe für Partei i). Dabei darf die Anzahl der Parteien vom Sicherheitsparameter k abhängen. Wir werden im folgenden immer von genau k Parteien ausgehen.

Wir erinnern uns, welche Aufgaben F hat:

- In der ersten Aktivierung erwartet F von jeder Partei j eine Eingabe der Form (b_i, s_i). Dabei ist $b_i \in \{real, ideal\}$ ein Flag, das angibt, ob Partei i eine reale oder eine ideale Protokollinstanz darstellt. Weiterhin ist $s_i \in \mathbb{N}$ eine natürliche Zahl, die den Schwierigkeitsgrad des TL-Puzzles für die Partei i festlegt. Im Falle $b_i = real$ ist s_i irrelevant.

- In den folgenden Aktivierungen wird jeder Partei i mit $b_i = \mathit{ideal}$ ein TL-Puzzle des Schwierigkeitsgrads s_i gestellt. Die Ausgaben von F für die einzelnen Parteien sind die vom Verifier $\mathcal{V}$ des TL-Puzzles gesandten Nachrichten, die Eingaben in den jeweiligen Runden die Antworten an den Verifier.

- In der letzten Runde (also wenn das TL-Puzzle fertig ist) wird an alle Parteien die gleiche Ausgabe *out* gegeben. Dabei ist $\mathit{out} = \mathit{success}$, wenn *alle* TL-Puzzles gelöst wurden, und außerdem $s_{i_\mu} \geq 2^\mu$ für alle $\mu = 1, \ldots, n$, wobei $s_{i_1}, \ldots, s_{i_n}$ die Schwierigkeitsgrade der Parteien mit $b_i = \mathit{ideal}$ nach aufsteigender Größe sortiert seien.

Um dies genauer zu spezifizieren, sei zunächst $\mathcal{V}$ der Verifier eines beliebigen, aber im folgenden festen TL-Puzzles mit den folgenden Eigenschaften:

- Die erste Nachricht in einer Interaktion zwischen $\mathcal{V}$ und einem Prover wird immer von $\mathcal{V}$ geschickt.

- Das TL-Puzzle läuft genau $\mathit{rounds}_\mathcal{V}(k)$ Runden, wobei $\mathit{rounds}_\mathcal{V}(k)$ nur vom Sicherheitsparameter k abhängt. Das heißt der Verifier sendet genau $\mathit{rounds}_\mathcal{V}(k)$ Nachrichten und empfängt $\mathit{rounds}_\mathcal{V}(k)$ Nachrichten und terminiert dann. Dabei ist $\mathit{rounds}_\mathcal{V}$ polynomiell-beschränkt und effizient berechenbar.

- Der Verifier kennt nur die Ausgaben 0 und 1 (1 falls das TL-Puzzle gelöst wurde, 0 sonst).

Da man offensichtlich jedes TL-Puzzle leicht in diese Form bringen kann, existiert $\mathcal{V}$ unter den Annahmen von Satz 6.3. Hier und im Rest dieses Kapitels bezeichne $\mathcal{V}$ immer den Verifier dieses speziellen TL-Puzzles. Der wegen der Einfachheitsbedingung (Definition 5.3) für jedes Polynom p existierende zugehörige Prover, der TL-Puzzles bis zu einem Schwierigkeitsgrad von $p(k)$ löst, heiße im folgenden $C_{TL,p}$.

Definition 6.4 (Die reaktive Funktion F)
F ist die reaktive Funktion in $\mathit{rounds}_\mathcal{V} + 1$ Runden, die wie folgt definiert ist (k bezeichne den Sicherheitsparameter):

- In der ersten Runde erwartet F von jeder Partei $i \in \{1, \ldots, k\}$ Eingaben $(b_i, s_i) \in \{\mathit{real}, \mathit{ideal}\} \times \{1, \ldots, 2^k\}$. Es sei $I := \{i : b_i = \mathit{ideal}\}$. Für jedes $i \in I$ instantiiert F eine Instanz $\mathcal{V}_i$ von $\mathcal{V}$ mit Eingaben $(1^k, s)$. Die Ausgabe von F an Partei $i \in I$ ist dann die erste von $\mathcal{V}_i$

(Fortsetzung nächste Seite)

(Fortsetzung)

gesandte Nachricht. Die Ausgabe von F an Partei $i \notin I$ ist λ (das leere Wort).
Der interne Zustand besteht (hier und in den folgenden Runden) aus den b_i, den s_i und den internen Zuständen der $\mathcal{V}_i$.

- In Runden $r = 2, \ldots, \mathit{rounds}_{\mathcal{V}}$ wird von jeder Partei $i \in I$ eine Eingabe m_i erwartet (die Eingaben von Parteien $i \notin I$ werden ignoriert). Die Eingabe m_i wird als $((r-1)$-te) eingehende Nachricht an $\mathcal{V}_i$ geleitet. Es sei m_i' die Antwort von $\mathcal{V}_i$ (die r-te ausgehende Nachricht). Die Ausgabe von F an Partei $i \in I$ ist m_i'. Die Ausgabe an Partei $i \notin I$ ist λ.
- In der letzten Runde $r = \mathit{rounds}_{\mathcal{V}} + 1$ wird von jeder Partei $i \in I$ eine Eingabe m_i erwartet (die Eingaben von Parteien $i \notin I$ werden ignoriert). Die Eingabe m_i wird als ($\mathit{rounds}_{\mathcal{V}}$-te, d. h. letzte) eingehende Nachricht an $\mathcal{V}_i$ geleitet. Es bezeichne $V_i \in \{0,1\}$ die Ausgabe von $\mathcal{V}_i$ (da $\mathcal{V}_i$ immer genau $\mathit{rounds}_{\mathcal{V}}$ Runden läuft, wird jetzt eine Ausgabe gegeben). Die Ausgabe an jede Partei $i \in \{1, \ldots, k\}$ ist jetzt *out*, wobei *out* wie folgt definiert ist:

  Es sei $s_{i_1} \leq \cdots \leq s_{i_{\#I}}$ eine aufsteigende Sortierung der s_i mit $i \in I$. Dann ist $\mathit{out} = \mathit{success}$, wenn folgende Bedingungen beide erfüllt sind:
  - Für jedes $i \in I$ ist $V_i = 1$ (d. h. alle TL-Puzzles wurden gelöst).
  - Für jedes $\mu = 1, \ldots, \#I$ ist $s_{i_\mu} \geq k^\mu$.

  Andernfalls ist $\mathit{out} = \mathit{failed}$.

6.4. Die sichere Funktionsauswertung

Wie wir bereits in Abschnitt 6.2 gesehen haben, ist es nötig, die Funktion F aus Definition 6.4 in sicherer Weise verteilt von mehreren Parteien ausrechnen zu lassen. Wir müssen daher zunächst spezifizieren, was wir unter einer verteilten Berechnung einer reaktiven Funktion verstehen, und welche Sicherheitsanforderungen wir an diese Berechnung stellen.[1]

Ein Protokoll zur sicheren Berechnung einer reaktiven Funktion F besteht aus einer ITM für jede Partei $i = 1, \ldots, k$. Zu Beginn jeder Runde der Funktionsauswertung erwartet jede ITM i eine Eingabe I_i, darauf kommunizieren die ITM, und schließlich liefert jede ITM i eine Ausgabe O_i. Dies wiederholt sich für jede

[1] Eine natürliche Anforderung, insbesondere im Kontext dieser Arbeit, wäre allgemeine oder spezielle komplexitätstheoretische Sicherheit. Leider ist diese Sicherheitsanforderung jedoch zu strikt, denn es gibt beweisbar keine allgemeinen Konstruktionen für sichere Funktionsauswertungen bezüglich dieser Sicherheitsbegriffe [CF01, CKL03].

Runde. Intuitiv verlangen wir, daß die Ausgaben O_i gerade die Ausgaben sind, die die reaktive Funktion F bei Eingaben I_i auch liefern würde.

Um unsere Sicherheitsanforderungen genauer spezifizieren zu können, wollen wir zunächst etwas Notation einführen. Das Protokoll der Funktionsauswertung ist durch eine PITM P gegeben, welche die Eingaben $(1^k, i)$ erwartet. Dabei ist k der Sicherheitsparameter, und i die Nummer der Partei.[2]

Die PITM $P(1^k, i)$ kennt zwei Typen von Nachrichten: Kommunikationsnachrichten und Ein-/Ausgabenachrichten. Erstere sind für die Kommunikation zwischen den Parteien gedacht, zweitere für die Ein- und Ausgaben I_i und O_i. Eine Kommunikationsnachricht ist immer mit der Identität einer Partei versehen (dem Empfänger bei ausgehenden, und dem Absender bei eingehenden Nachrichten).

Eine *reale Protokollausführung von* P sieht wie folgt aus: Gegeben seien eine PITM A (der Angreifer) und eine PITM Z (die Umgebung), die Auxiliary inputs z_A und z_Z erhalten, und eine Menge $C \subseteq \{1, \ldots, k\}$ (die Menge der korrumpierten Parteien). Dann besteht $\langle P_C, A(1^k, z_A, C), Z(1^k, z_Z, C)\rangle := (o_A, o_Z)$ aus der Ausgabe von A und Z nach folgender Ausführung:

- Zunächst werden A und Z mit den Eingaben $(1^k, z_A, C)$ und $(1^k, z_Z, C)$ instantiiert.
- Für jedes $i \notin C$ wird eine Instanz P_i von P mit Eingaben $(1^k, i)$ gestartet (eine unkorrumpierte Partei i).
- Wenn Z eine Nachricht an die (unkorrumpierte) Partei i mit $i \notin C$ schickt, wird P_i mit dieser Nachricht als Eingabenachricht aktiviert.
- Wenn die (unkorrumpierte) Partei P_i $(i \notin C)$ eine Ausgabenachricht m schickt, wird Z mit Nachricht (m, i) aktiviert.
- Wenn A eine Nachricht m an eine unkorrumpierte Partei i $(i \notin C)$ im Namen einer korrumpierten Partei j schickt, wird P_i mit Kommunikationsnachricht m mit Absender j aktiviert.
- Wenn die (unkorrumpierte) Partei P_i $(i \notin C)$ eine Kommunikationsnachricht m mit Empfänger j schickt, so unterscheiden wir zwei Fälle: Ist Partei j unkorrumpiert (d. h. $j \notin C$) dann wird P_j mit Kommunikationsnachricht m mit Absender i aktiviert. Ist Partei j korrumpiert (d. h. $j \in C$) dann wird A mit der Nachricht (m, i, j) aktiviert.

[2] Wir verwenden eine PITM P für alle Parteien, da die Anzahl der Parteien im Sicherheitsparameter steigen kann und wir deshalb nicht für jede Partei eine eigene PITM spezifizieren können. Das Verhalten der Parteien kann natürlich trotzdem verschieden sein, da P die Identität i der Partei als Eingabe erhält.

- Am Ende geben A und Z die Ausgaben o_A und o_Z.

Die Situation ist also die folgende: Die Umgebung Z kann Ein-/Ausgabenachrichten an unkorrumpierte Parteien senden und von diesen empfangen. Die Kommunikation zwischen den unkorrumpierten Parteien ist unmittelbar (der Angreifer A kann nichts verzögern) und geheim (der Angreifer kann Nachrichten weder lesen noch verändern). Die unkorrumpierten Parteien führen das von P vorgeschriebene Protokoll aus, während die korrumpierten Parteien komplett vom Angreifer A gesteuert werden: Der Angreifer liest und sendet im Namen der korrumpierten Parteien. Angreifer und Umgebung können nicht direkt miteinander kommunizieren.[3]

Um vergleichen zu können, ob ein Protokoll P tatsächlich eine ideale reaktive Funktion F implementiert, müssen wir noch die *ideale Funktionsauswertung* definieren. Gegeben eine PITM S (den Simulator) und eine PITM Z (die Umgebung), die Auxiliary inputs z_S und z_Z erhalten, und eine Menge $C \subseteq \{1, \ldots, k\}$ (die Menge der korrumpierten Parteien), besteht das Resultat $\langle F_C, S(1^k, z_S, C), Z(1^k, z_Z, C)\rangle := (o_S, o_Z)$ der idealen Funktionsauswertung aus der Ausgabe von S und Z nach folgender Ausführung:

- Zunächst werden S und Z mit den Eingaben $(1^k, z_S, C)$ und $(1^k, z_Z, C)$ instantiiert. Die reaktive Funktion F befindet sich Anfangs in der ersten Runde $r = 1$ und erwartet die Eingaben der ersten Runde.

- Wenn Z eine Nachricht m an die (unkorrumpierte) Partei i mit $i \notin C$ schickt, erhält F die Nachricht m als r-te Eingabe von Partei i.

- Wenn S eine Nachricht m an die (korrumpierte) Partei i mit $i \in C$ schickt, erhält F die Nachricht m als r-te Eingabe von Partei i.

- Sobald F alle r-ten Eingaben erhalten hat, werden die r-ten Ausgaben $(o_1, \ldots, o_k)$ berechnet. Dann geht F in die nächste Runde $r := r + 1$ über (es sei denn, wir waren bereits in der letzten Runde). Für jedes $i \notin C$ wird (o_i, i) an Z geschickt (d. h. aus Sicht von Z hat Partei i die Nachricht o_i geschickt). Für jedes $i \in C$ wird (o_i, i) an S geschickt.[4]

- Am Ende geben S und Z die Ausgaben o_S und o_Z.

[3] Dies ist sehr wichtig, da sonst die in [CF01, CKL03] gezeigten Unmöglichkeitsergebnisse auch hier zutreffen würden.

[4] Wir sind an dieser Stelle etwas unsauber und spezifizieren nicht, in welcher Reihenfolge diese Nachrichten ausgeliefert werden und wie das Scheduling im Detail aussieht. Theoretisch wäre dies notwendig, da Z ja auch anhand von Schedulingunterschieden zwischen realer und idealer Protokollausführung unterscheiden könnte. Wir haben uns jedoch dafür entschieden, diese Details zu übergehen, da sie die Präsentation unnötig komplizieren würden und für den Beweis nicht wesentlich sind.

Im Idealen ist die Situation also die folgende: Die Eingaben, die die Umgebung Z an die unkorrumpierten Parteien schickt, werden statt dessen an die Funktion F ausgeliefert. Die Eingaben der korrumpierten Parteien darf der Simulator S bestimmen. Entsprechend werden auch die Ausgaben von F verteilt. Wieder können S und Z nicht direkt miteinander kommunizieren.

Man beachte, daß im Falle $C = \varnothing$ (keine Partei ist korrumpiert) der Angreifer bzw. der Simulator mit keiner anderen Maschine kommunizieren können. Diese haben daher auch keinen Einfluß auf die Ausgabe der Umgebung Z. Wir schreiben deshalb im Falle $C = \varnothing$ einfach $\langle P_\varnothing, Z(1^k, z_Z, \varnothing)\rangle$ und $\langle F_\varnothing, Z(1^k, z_Z, \varnothing)\rangle$ statt $\langle P_\varnothing, A(1^k, z_A, \varnothing), Z(1^k, z_Z, \varnothing)\rangle$ und $\langle F_\varnothing, S(1^k, z_S, \varnothing), Z(1^k, z_Z, \varnothing)\rangle$, wobei wir davon ausgehen, daß die nicht angegebenen Angreifer und Simulator leere Ausgabe haben.

Wir können nun die gewünschten Sicherheitseigenschaften definieren. Für unsere Zwecke ist es nicht notwendig, daß die Funktionsauswertung in jeder Situation sicher ist, vielmehr genügt es, daß sie sicher ist, wenn niemand korrumpiert ist, und wenn alle bis auf eine Partei korrumpiert sind (andere Szenarien kommen in unserem Beweis nicht vor).

Definition 6.5 (Korrektheit einer Funktionsauswertung)
Eine Funktionsauswertung P von F ist *korrekt* (d. h. sicher im unkorrumpierten Fall), wenn für jede PITM Z (die Umgebung) die folgenden Familien von Zufallsvariablen komplexitätstheoretisch ununterscheidbar sind:

$$\Big\{\big\langle P_\varnothing, Z(1^k, z_Z, \varnothing)\big\rangle\Big\}_{k, z_Z} \quad \text{und} \quad \Big\{\big\langle F_\varnothing, Z(1^k, z_Z, \varnothing)\big\rangle\Big\}_{k, z_Z},$$

wobei $z_Z \in \Sigma^*$.

Definition 6.6 (Sicherheit einer Funktionsauswertung bei einer unkorrumpierten Partei)
Eine Funktionsauswertung P von F ist *sicher bei einer unkorrumpierten Partei*, wenn für jede PITM A (den Angreifer) eine PITM S (der Simulator) existiert, so daß für jede PITM Z (die Umgebung) gilt:

$$\Big\{\big\langle P_C, A(1^k, z_A, C), Z(1^k, z_Z, C)\big\rangle\Big\}_{k, z_A, z_Z, C}$$
$$\text{und} \quad \Big\{\big\langle F_C, S(1^k, z_A, C), Z(1^k, z_Z, C)\big\rangle\Big\}_{k, z_A, z_Z, C}$$

(Fortsetzung nächste Seite)

(Fortsetzung)

sind ununterscheidbar, wobei $z_Z, z_A \in \Sigma^*$ und $C \subseteq \{1, \ldots, k\}$ mit $\#C = k - 1$.

Es stellt sich nun die Frage, ob eine sichere Funktionsauswertung für allgemeine F überhaupt existiert. Diese Frage beantwortet uns z. B. [Gol04, Kapitel 7] (dort insbesondere Theorem 7.5.15) positiv.[5] Dort wird zwar nur der Fall nichtreaktiver Funktionen (in anderen Worten reaktive Funktionen in einer Runde) behandelt, aber mit den Bemerkungen aus [Gol04, Abschnitt 7.7.1.3 (Reactive Systems)] lassen sich die dortigen Ergebnisse auch auf den reaktiven Fall erweitern, und wir erhalten als Spezialfall die folgende Aussage:[6]

Satz 6.7 (Sichere Funktionsauswertung)
Es sei F eine reaktive Funktion im Sinne von Abschnitt 6.3. Wenn Enhanced trapdoor permutations (Definition 1.9) existieren, und F durch eine polynomiell-beschränkte ITM realisierbar ist, dann existiert eine Funktionsauswertung P für F im obigen Sinne (d. h. eine PITM P), die die Sicherheitsanforderungen aus Definitionen 6.5 und 6.6 erfüllt.

Unter den Voraussetzungen von Satz 6.3 existiert also eine sichere Funktionsauswertung P für die reaktive Funktion F aus Definition 6.4 im Sinne der Definitionen 6.5 und 6.6. Im folgenden bezeichne P immer diese Funktionsauswertung.

6.5. Die Protokolle

Wir können nun die Protokolle π und ρ angeben, aus denen unser trennendes Beispiel besteht. Wir erinnern uns (vgl. Abschnitt 6.2), daß sowohl π als auch ρ aus Sicht der Umgebung eine Partei der sicheren Funktionsauswertung P aus dem vorherigen Abschnitt darstellen, und nach Beendigung der Funktionsauswertung das Ergebnis an die Umgebung liefern. Die Eingaben für die Funktionsauswertung jedoch werden im ersten Falle von π selbst, im Falle von ρ aber vom Simulator gewählt. Dies garantiert, daß – sofern nicht das Endergebnis der

[5] Dort wird zusätzlich verlangt, daß den Parteien ein Broadcast-Kanal zur Verfügung steht. Ein solcher ist in unserer Modellierung einer Funktionsauswertung P nicht vorhanden. Theorem 7.5.15 aus [Gol04] ist aber trotzdem anwendbar, da wir uns nur auf die Fälle beschränken, in denen alle oder genau eine Partei unkorrumpiert sind. In diesen Fällen kann man einen Broadcast einer Nachricht m dadurch ersetzen, daß die Nachricht m an alle Parteien einzeln gesandt wird.

[6] Uns ist leider keine explizite Behandlung des reaktiven Falls in der Literatur bekannt.

Funktionsauswertung verschieden ist – die Umgebung nicht zwischen π und ρ unterscheiden kann.

Wir überlassen es dabei der Umgebung, die Kommunikation zwischen den Parteien der sicheren Funktionsauswertung (d. h. zwischen den Instanzen von π bzw. ρ) zu vermitteln. Insbesondere kann die Umgebung Parteien simulieren und mit den Parteien π bzw. ρ verbinden (was sie im unkomponierten Falle auch tun muß, da sonst nur eine Partei an der k-Parteien-Funktionsauswertung teilnehmen würde). Damit das Protokoll π bzw. ρ auch weiß, welche Partei es implementieren muß, übergibt die Umgebung jeder Protokollinstanz zu Beginn eine Parteiidentität $i \in \{1, \ldots, k\}$.

Die Eingaben der sicheren Funktionsauswertung P (und damit für die ausgewertete Funktion F) werden von den Protokollinstanzen wie folgt gewählt:

- Das reale Protokoll π setzt $b_i := \mathit{real}$ und verwendet dieses b_i als Eingabe für die erste Runde von P bzw. F. Da für $b_i = \mathit{real}$ die Eingaben s_i und die Eingaben der folgenden Runden von F ignoriert werden, können wir diese willkürlich festlegen.

  Die Ausgabe der letzten Runde wird an die Umgebung geliefert, die Ausgaben der anderen Runden werden ignoriert.

- Das ideale Protokoll ρ setzt $b_i := \mathit{ideal}$ und fragt den Simulator nach dem Schwierigkeitsgrad s_i. Das resultierende Paar (b_i, s_i) ist dann die Eingabe für die erste Runde von P bzw. F.

  Die Ausgaben aller Runden außer der letzten werden an den Simulator geliefert (es handelt sich dabei um die Fragen des TL-Puzzle-Verifiers $\mathcal{V}$), die Eingaben der zweiten bis letzten Runde (die Antworten auf diese Fragen) werden vom Simulator erwartet. Die Ausgabe der letzten Runde wird an die Umgebung geliefert.

Wir geben zunächst die Details des realen Protokolls π:

Definition 6.8 (Das reale Protokoll π)
Das Protokoll π besteht aus einer Maschine M_{real} (vgl. Abbildung 6.1a).

Die Maschine M_{real} hat den eingehenden Protokolleingabeport `in_func` und die ausgehenden Protokollausgabeports `out_func` und `out_result`. Die Maschine M_{real} verhält sich wie folgt:

- Bei der ersten Aktivierung über `in_func` mit einer Nachricht $i \in \{1, \ldots, k\}$ wird eine Instanz $P(1^k, i)$ von P gestartet. (D. h. die i-te Partei der sicheren Funktionsauswertung P wird simuliert.) Als Eingabe für die erste Runde erhält P die Nachricht $(b_i, s_i) := (\mathit{real}, 0)$.

(Fortsetzung nächste Seite)

(Fortsetzung)

- Wenn P eine Kommunikationsnachricht sendet, so wird diese über den Protokollausgabeport `out_func` ausgegeben (zusammen mit dem Empfänger).
- Wenn über `in_func` eine Nachricht empfangen wird (zusammen mit einem Absender), so wird diese als Kommunikationsnachricht an P geliefert.
- Wenn P eine Ausgabenachricht liefert (außer in der letzten Runde), so wird diese verworfen und P erhält als Eingabe für die nächste Runde die leere Eingabe λ.
- Wenn P in der letzten Runde eine Ausgabenachricht liefert, so wird diese über `out_result` ausgegeben, und M_{real} terminiert.

Das ideale Protokoll ρ sieht im Detail wie folgt aus:

Definition 6.9 (Das ideale Protokoll ρ)
Das Protokoll ρ besteht aus einer Maschine M_{ideal} (vgl. Abbildung 6.1b).

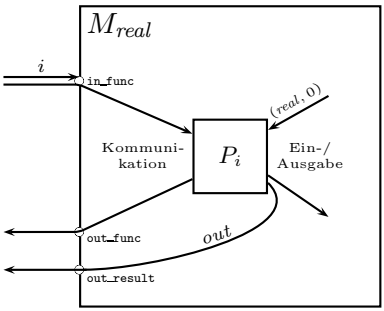
Die Maschine M_{ideal} hat den eingehenden Protokolleingabeport `in_func` und die ausgehenden Protokollausgabeports `out_func` und `out_result`. Weiterhin hat M_{ideal} den eingehenden Angreiferport `adv_input` und den ausgehenden Angreiferport `adv_output`.

Die Maschine M_{ideal} verhält sich wie folgt:

- Bei der ersten Aktivierung über `in_func` mit einer Nachricht $i \in \{1, \ldots, k\}$ wird eine Instanz $P(1^k, i)$ von P gestartet. (D. h. die i-te Partei der sicheren Funktionsauswertung P wird simuliert.) Dann wird eine Nachricht *start* über `adv_output` geschickt (um den Simulator zu informieren, daß die Funktionsauswertung begonnen hat und eine Eingabe erwartet wird).
- Wenn P eine Kommunikationsnachricht sendet, so wird diese über den Protokollausgabeport `out_func` ausgegeben (zusammen mit dem Empfänger).
- Wenn über `in_func` eine Nachricht empfangen wird (zusammen mit einem Absender), so wird diese als Kommunikationsnachricht an P geliefert.

(Fortsetzung nächste Seite)

(a)



(b)

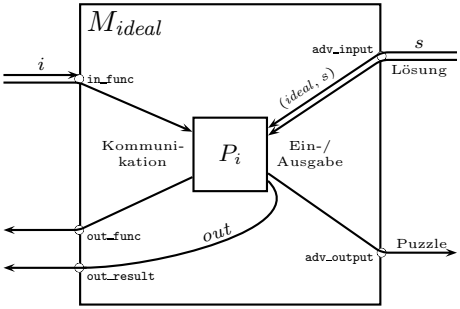


Abbildung 6.1.: Die Protokollmaschinen M_{real} und M_{ideal}.

(a) M_{real} simuliert die Partei i der Funktionsauswertung P. Der Index i wird von der Umgebung festgelegt. Die Kommunikationsnachrichten von P_i werden über die Umgebung geleitet. Die Eingaben für P_i sind fest: $(b_i, s_i) := (real, 0)$. Die Ausgaben von P_i werden verworfen, mit Ausnahme der letzten Ausgabe out, die an die Umgebung geleitet wird.

(b) M_{ideal} funktioniert wie M_{real}, mit den folgenden Änderungen: Die Eingaben für P_i werden vom Simulator gewählt, lediglich für die Eingabe der ersten Runde wird $b_i = ideal$ erzwungen, s_i darf der Simulator jedoch selbst wählen. Die Ausgaben (bis auf die der letzten Runde) werden dem Simulator übermittelt.

(Fortsetzung)

- Wenn über den Angreiferport `adv_input` eine Nachricht m empfangen wird, so wird diese als Eingabenachricht an P gegeben (also als Eingabe für die aktuelle Runde verwendet). Ist m jedoch die erste solche Nachricht, so wird stattdessen $(b_i, s_i) := (\mathit{ideal}, m)$ an P geliefert.
- Wenn P eine Ausgabenachricht liefert (außer in der letzten Runde), so wird diese über den Angreiferport `adv_output` weitergeleitet.
- Wenn P in der letzten Runde eine Ausgabenachricht liefert, so wird diese über `out_result` ausgegeben, und M_{ideal} terminiert.

Nachdem wir die Protokolle π und ρ aus Satz 6.3 explizit angegeben haben, können wir endlich zeigen, daß π zwar so sicher wie ρ bezüglich spezieller Sicherheit ist, aber das komponierte Protokoll $f \cdot \pi$ nicht so sicher wie $f \cdot \rho$ bezüglich spezieller Sicherheit ist. Dies soll in den nächsten zwei Abschnitten geschehen.

6.6. Unsicherheit des komponierten Protokolls

Wir werden nun zunächst den einfacheren Fall behandeln und zeigen, daß $f \cdot \pi$ nicht so sicher wie $f \cdot \rho$ ist. Die Beweisidee ist die folgende: Um festzustellen, ob f Instanzen von π oder von ρ vorliegen, simuliert die Umgebung noch $k - f$ weitere Instanzen des realen Protokolls π. Im Falle des realen Modells wird also eine sichere Funktionsauswertung von F durchgeführt, bei der alle Parteien die Eingabe $b_i = \mathit{real}$ liefern. Nach Definition von F ist die Ausgabe der Funktionsauswertung dann $\mathit{out} = \mathit{success}$ (wir benutzen hier die Korrektheit von P gemäß Definition 6.5).

Im idealen Modell jedoch nehmen an der Funktionsauswertung nun die $k - f$ von der Umgebung simulierten Parteien mit Eingabe $b_i = \mathit{real}$, sowie die f von den Instanzen von ρ simulierte Parteien mit Eingabe $b_i = \mathit{ideal}$. Nach Definition von F kann die Funktionsauswertung höchstens dann $\mathit{out} = \mathit{success}$ ausgeben, wenn ein TL-Puzzle des Schwierigkeitsgrads $s \geq k^{\#I}$ gelöst wird (auch hier nutzen wir wieder die Korrektheit von P). Da aber hier $\#I = f$, muß also ein TL-Puzzle vom Schwierigkeitsgrad $s \geq k^f$ gelöst werden; und da f nicht beschränkt ist, ist k^f nicht polynomiell-beschränkt. Somit kann kein solches TL-Puzzle gelöst werden (zumindest nicht von einem polynomiell-beschränkten Simulator), und die Ausgabe der Funktionsauswertung ist nur mit vernachlässigbarer Wahrscheinlichkeit $\mathit{out} = \mathit{success}$. Hieran kann die Umgebung das reale und das ideale Modell unterscheiden, und wir erkennen, daß $f \cdot \pi$ nicht so sicher wie $f \cdot \rho$ ist.

Diese Beweisführung gliedern wir in drei Lemmata, die im wesentlichen die gleiche Aussage in verschiedenen Modellen zeigen. Im ersten Lemma 6.10 betrachten wir eine ideale Auswertung der reaktiven Funktion F und untersuchen, in welchen Fällen die Ausgabe $out = success$ auftritt. Im zweiten Lemma 6.11 ersetzen wir F durch die sichere Funktionsauswertung P. Und im dritten Lemma 6.12 schließlich betrachten wir die Protokolle π und ρ. Wir möchten den Leser an dieser Stelle vor möglichen Verwirrungen warnen: Der Simulator $\mathcal{S}$ aus der Definition der speziellen Sicherheit, der im dritten Lemma vorkommt, entspricht der *Umgebung* Z_{ideal} der sicheren Funktionsauswertung im ersten und zweiten Lemma. Die Umgebung $\mathcal{Z}$ aus dem dritten Lemma hingegen übernimmt im zweiten Lemma die Rolle des Netzwerkes (kommt also nur implizit vor) und kommt im ersten Lemma gar nicht vor. Es ist für das Verständnis des Beweises wichtig, diese Tatsache im Auge zu behalten und sich nicht von ähnlichen Bezeichnungen in verschiedenen Kontexten irreführen zu lassen.

Lemma 6.10 (Unsicherheit im F-Modell)
Es sei F die reaktive Funktion aus Definition 6.4 und f unbeschränkt und effizient berechenbar mit $f \leq k$.

Die PITM Z_{real} sei eine Umgebung für die reaktive Funktionsauswertung F, die jeder Partei für die erste Runde die Eingabe $(b_i, s_i) = (real, 0)$ gibt, und in allen weiteren Runden die leere Eingabe.

Die PITM Z_{ideal} sei eine Umgebung für F, die höchstens $k - f$ Parteien in der ersten Runde eine Eingabe (b_i, s_i) mit $b_i = real$ gibt.

Die Ausgabe von Z_{ideal} und von Z_{real} sei die Ausgabe der letzten Runde einer beliebigen Partei.

Dann sind

$$P\Big(\big\langle F_\varnothing, Z_{real}(1^k, z, \varnothing)\big\rangle = success\Big)$$

$$\text{und} \quad P\Big(\big\langle F_\varnothing, Z_{ideal}(1^k, z, \varnothing)\big\rangle \neq success\Big)$$

überwältigend.

Beweis: Wir zeigen zunächst, daß immer $P\big(\langle F_\varnothing, Z_{real}(1^k, z, \varnothing)\rangle = success\big) = 1$. Da Z_{real} jeder Partei in jeder Runde Eingaben liefert, wird die letzte Runde der idealen Auswertung von F erreicht. Dann liefert F an alle Parteien die gleiche Ausgabe *out*. Es sei I die Menge der i mit $b_i = ideal$, also $I = \varnothing$. Diese Ausgabe *out* ist nach Definition (vgl. Definition 6.4) trivialerweise $out = success$ wenn $I = \varnothing$. Mit Wahrscheinlichkeit 1 gilt also $\langle F_\varnothing, Z_{real}(1^k, z, \varnothing)\rangle = success$.

Nun wenden wir uns der Aussage zu, daß $P\big(\langle F_\varnothing, Z_{ideal}(1^k, z, \varnothing)\rangle \neq success\big)$

überwältigend ist. Wir nehmen dazu an, dem sei nicht so, also daß mit *nicht* vernachlässigbarer Wahrscheinlichkeit

$$\langle F_\varnothing, Z_{ideal}(1^k, z, \varnothing)\rangle = success.$$

Es bezeichne im folgenden b_i, s_i die entsprechend benannten Eingaben in der ersten Runde von F, und $\mathcal{V}_i$ die Ausgabe der zu Partei i gehörigen Instanz des von F simulierten TL-Puzzle-Verifiers $\mathcal{V}$ (vgl. Definition 6.4). I sei die Menge der i mit $b_i = ideal$. Weiter bezeichne out die Ausgabe von F (die für alle Parteien gleich ist). Es ist also $out = success$ mit nicht vernachlässigbarer Wahrscheinlichkeit μ.

Da out nur dann definiert ist, wenn jede der k Parteien ein $b_i \in \{real, ideal\}$ eingibt, und da Z_{ideal} höchstens $(k-f)$ Eingaben $b_i = real$ eingibt, folgt $\#I \geq f$.

Nach Definition von F (genauer: von out) ist höchstens dann $out = success$, wenn ein i existiert, so daß $\mathcal{V}_i = 1$ (das i-te TL-Puzzle wurde erfolgreich gelöst) und $s_i \geq k^{\#I} \geq k^f$. Also löst Z_{ideal} mit nicht vernachlässigbarer Wahrscheinlichkeit ein TL-Puzzle des Schwierigkeitsgrads $s \geq k^f$. Genauer: Wenn wir ein $i \in \{1, \ldots, k\}$ zufällig wählen, so gilt mit Wahrscheinlichkeit mindestens μ/k (was nicht vernachlässigbar ist), daß $s_i \geq k^f$ und $\mathcal{V}_i = 1$. Dies steht im Widerspruch zur Schwierigkeitsbedingung der TL-Puzzles (Definition 5.3). Somit ist unsere Annahme falsch und $P\big(\langle F_\varnothing, Z_{ideal}(1^k, z, \varnothing)\rangle \neq success\big)$ überwältigend. □

Lemma 6.11 (Unsicherheit im *P*-Modell)
Es sei P die von Satz 6.7 garantierte sichere Funktionsauswertung für F und f, Z_{real} und Z_{ideal} wie in Lemma 6.10. Dann sind

$$P\Big(\langle P_\varnothing, Z_{real}(1^k, z, \varnothing)\rangle = success\Big)$$

$$\text{und} \quad P\Big(\langle P_\varnothing, Z_{ideal}(1^k, z, \varnothing)\rangle \neq success\Big)$$

überwältigend.

Beweis: Unmittelbar aus Lemma 6.10 und der Korrektheit von P (Satz 6.7). □

Lemma 6.12 (Unsicherheit des komponierten Protokolls)

(Fortsetzung nächste Seite)

(Fortsetzung)

Es seien π und ρ wie in Definitionen 6.8 und 6.9. Weiter sei f unbeschränkt und effizient berechenbar. Dann ist $f \cdot \pi$ *nicht* so sicher wie $f \cdot \rho$ bezüglich spezieller komplexitätstheoretischer Sicherheit weder mit noch ohne Auxiliary input und weder bzgl. der Sicht noch bzgl. der Ausgabe der Umgebung.

Beweis: Wegen Lemmata A.5 und A.9 ist es hinreichend, den Fall der Sicherheit ohne Auxiliary input bzgl. der Ausgabe der Umgebung zu betrachten.

Wenn $\min\{k, f\} \cdot \pi$ nicht so sicher wie $\min\{k, f\} \cdot \rho$ ist, dann ist erst recht nicht $f \cdot \pi$ so sicher wie $f \cdot \rho$. Wir können deshalb o. B. d. A. annehmen, daß $f \leq k$.

Wir definieren zunächst eine Umgebung, die die Aufgabe hat, $k - f$ Instanzen von M_{real} zu simulieren, und die Funktionsauswertungs-Kommunikation zwischen diesen und den f Instanzen von M_{real} bzw. M_{ideal} durchzuleiten.

Die Umgebung $\mathcal{Z}$ sieht wie folgt aus: Sie hat einen eingehenden Umgebungsport `env_start` (über den sie zu Beginn des Protokollaufs durch den Angreifer aktiviert werden kann), sowie einen ausgehenden Protokolleingabeport `in_func` und eingehende Protokollausgabeports `out_func` und `out_result`, sowie den ausgehenden Spezialport `output` für die Ausgabe am Ende des Protokollaufs. Die Umgebung $\mathcal{Z}$ hat folgendes Programm:

- Es werden $k - f$ Instanzen von M_{real} simuliert, diese bezeichnen wir im folgenden mit $M_{f+1}, \ldots, M_k$. Die f Instanzen von M_{real} oder M_{ideal}, mit denen $\mathcal{Z}$ über die Protokollports verbunden ist, bezeichnen wir mit $M_1, \ldots, M_f$. Wenn wir z. B. sagen, daß $\mathcal{Z}$ eine Nachricht m an M_i über `in_func` sendet, so ist damit gemeint, daß für $i \leq f$ die Nachricht (i, m) über `in_func` nach außen gesandt wird (wo die Nachricht m die i-te Instanz von M_{real} bzw. M_{ideal} in $f \cdot M_{real}$ bzw. $f \cdot M_{ideal}$ erreicht). Ist $i > f$, so wird die simulierte Instanz M_i von M_{real} mit der Nachricht m auf `in_func` aktiviert.

- Für $i = 1, \ldots, k$ wird über `in_func` an M_i die Nachricht i geschickt.[7]

- Wann immer eine (simulierte oder nicht simulierte) Maschine M_i über den Port `out_func` eine Kommunikationsnachricht an eine Partei j schickt (wir

[7] Hier und an weiteren Stellen der Programmbeschreibung kommen Anweisungen vor, daß $\mathcal{Z}$ mehreren Maschinen eine Nachricht schicken soll. Natürlich kann $\mathcal{Z}$ dies nicht innerhalb einer Aktivierung tun, da eine Aktivierung mit dem Senden *einer* Nachricht endet. Wir gehen davon aus, daß $\mathcal{Z}$ eine Warteschlange für zu sendende Nachrichten hat, und bei jeder Aktivierung eine Nachricht aus dieser Warteschlange abarbeitet. Wir werden den Angreifer $\mathcal{A}$ weiter unten so wählen, daß dieser $\mathcal{Z}$ immer wieder aktiviert, wenn das Token verloren geht.

erinnern uns: sowohl M_{real} und M_{ideal} nutzen die Protokollports `out_func` und `in_func` um die Kommunikationsnachrichten der Funktionsauswertung zu übermitteln, vgl. Definitionen 6.8 und 6.9), so schickt $\mathcal{Z}$ an M_i über `in_func` diese Nachricht an M_j mit Absender i.

- Wenn eine Maschine M_i über den Protokollausgabeport `out_result` eine Nachricht *out* ausgibt, so wird diese Nachricht über `output` ausgegeben, und $\mathcal{Z}$ terminiert.

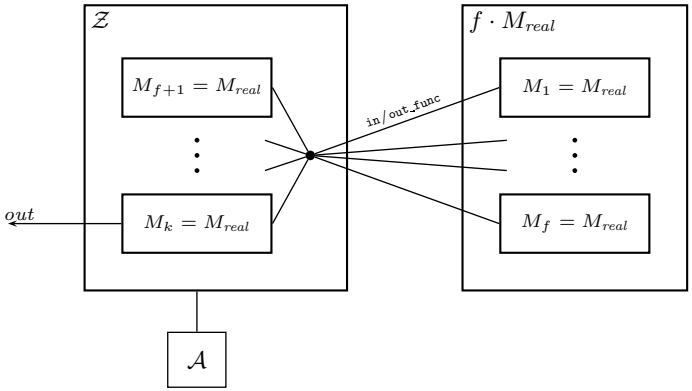
Passend zu $\mathcal{Z}$ konstruieren wir noch den Angreifer $\mathcal{A}$ mit dem eingehenden Spezialport `clk` und dem ausgehenden Umgebungsport `env_start`. $\mathcal{A}$ tut nichts anderes, als bei Empfang einer Nachricht auf `clk` diese über `env_start` an die Umgebung weiterzuleiten.

Zunächst betrachten wir einen Protokollauf von $f \cdot \pi$, $\mathcal{Z}$ und $\mathcal{A}$ (vgl. Abbildung 6.2a). Das komponierte Protokoll $f \cdot \pi$ besteht aus f Instanzen von M_{real}. Somit sind alle Maschinen M_i (die von $\mathcal{Z}$ simulierten für $i > f$, und die real existierenden für $i \leq f$) Instanzen von M_{real}. Jede dieser Maschinen simuliert eine Partei $P_i := P(1^k, i)$ der sicheren Funktionsauswertung P. Die Umgebung $\mathcal{Z}$ leitet die Kommunikationsnachrichten zwischen den Parteien P_i unverändert durch. Jede der Parteien P_i erhält (nach Definition von M_{real}, vgl. Definition 6.8) in der ersten Runde die Eingabe $(b_i, s_i) = (real, 0)$. In allen weiteren Runden erhält P_i die leere Eingabe. Insgesamt simuliert also der Protokollauf von $f \cdot \pi$, $\mathcal{Z}$, $\mathcal{A}$ eine sichere Funktionsauswertung P im unkorrumpierten Fall mit einer Umgebung Z_{real} wie in Lemmata 6.10 und 6.11. Die Ausgabe der Umgebung $\mathcal{Z}$ ist gerade die Ausgabe dieser Umgebung Z_{real}. Nach Lemma 6.11 ist letztere Ausgabe mit überwältigender Wahrscheinlichkeit $out = success$. Somit gibt auch $\mathcal{Z}$ mit überwältigender Wahrscheinlichkeit *success* aus, es ist also

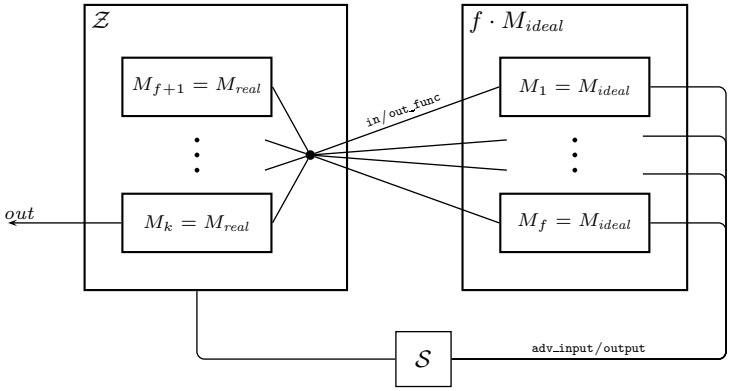
$$P\Big(\mathrm{OUTPUT}_{f\cdot\pi,\mathcal{A},\mathcal{Z}}(k) = success\Big) \tag{6.5}$$

überwältigend.

Nun untersuchen wir einen Protokollauf von $f \cdot \rho$, $\mathcal{Z}$ und $\mathcal{S}$, wobei $\mathcal{S}$ ein beliebiger polynomiell-beschränkter Simulator sei (vgl. Abbildung 6.2b). Das komponierte Protokoll besteht nun aus f Instanzen von M_{ideal}, und $\mathcal{Z}$ simuliert zusätzlich $k - f$ Kopien von M_{real}. Wieder simuliert jede dieser Maschinen M_i eine Partei $P_i := P(1^k, i)$ der sicheren Funktionsauswertung P. Die Umgebung leitet die Kommunikationsnachrichten zwischen den Parteien P_i unverändert durch. Nach Definition von M_{ideal} (Definition 6.9) erhält P_i mit $i > f$ in der ersten Runde eine Eingabe $(b_i, s_i) = (real, 0)$ und in allen weiteren Runden leere Eingabe. Nach Definition M_{real} erhält P_i mit $i \leq f$ in der ersten Runde eine Eingabe (b_i, s_i) mit $b_i = ideal$. Bis auf diese Einschränkungen werden die Eingaben vom Simulator



(a)



(b)

Abbildung 6.2.: Die Ausführung der komponierten Protokolle.

(a) Das reale Protokoll $f \cdot \pi$ bestehend aus der Protokollmaschine $f \cdot M_{real}$ führt f Instanzen von M_{real} aus. Die Umgebung simuliert weitere $k - f$ Instanzen, insgesamt werden also k Instanzen von M_{real} simuliert. Die Kommunikation zwischen diesen Instanzen wird von der Umgebung unverändert durchgeleitet. Das Ergebnis *out* der Funktionsauswertung wird von der Umgebung am Ende ausgegeben.

(b) Das ideale Protokoll $f \cdot \rho$ hingegen besteht aus der Protokollmaschine $f \cdot M_{ideal}$ und führt f Instanzen von M_{ideal} aus. Die Umgebung simuliert $k - f$ Instanzen von M_{real}. Die Kommunikation zwischen diesen Instanzen wird wieder von der Umgebung übernommen. Wieder wird *out* von der Umgebung am Ende ausgegeben. Die Eingaben für die simulierten Instanzen von P_i in den Kopien von M_{ideal} werden vom Simulator $\mathcal{S}$ geliefert.

$\mathcal{S}$ gewählt. Wir können die Wahl der Eingaben also durch eine polynomiell-beschränkte PITM Z_{ideal} beschreiben (welche im wesentlichen den Simulator $\mathcal{S}$ simuliert). Die Maschinen M_{ideal} und M_{real} geben schließlich das Ergebnis *out* der Funktionsauswertung über den Protokollausgabeport `out_result` aus. Die Ausgabe von Z_{ideal} sei die erste dieser Ausgaben *out*.

Diese PITM Z_{ideal} hat die in Lemmata 6.10 und 6.11 geforderten Eigenschaften. Also ist nach Lemma 6.11

$$P\Big(\big\langle P_{\varnothing}, Z_{ideal}(1^k, z, \varnothing)\big\rangle = success\Big)$$

vernachlässigbar. Damit ist aber auch die Wahrscheinlichkeit vernachlässigbar, daß die Ausgabe von $\mathcal{Z}$ (die ja wiederum der ersten Ausgabe einer Maschine M_i über `out_result` entspricht) *success* ist.

Mit (6.5) folgt (für beliebigen polynomiell-beschränkten Simulator), daß die folgenden Zufallsvariablen komplexitätstheoretisch unterscheidbar sind:

$$\mathrm{OUTPUT}_{f\cdot\pi,\mathcal{A},\mathcal{Z}}(k) \quad \text{und} \quad \mathrm{OUTPUT}_{f\cdot\rho,\mathcal{S},\mathcal{Z}}(k).$$

Damit aber ist $f \cdot \pi$ nicht so sicher wie $f \cdot \rho$ bezüglich spezieller komplexitätstheoretischer Sicherheit ohne Auxiliary input bzgl. der Ausgabe der Umgebung. □

6.7. Sicherheit des unkomponierten Protokolls

Um den Beweis von Satz 6.3 zu beenden, müssen wir nur noch zeigen, daß π nicht so sicher wie ρ ist. Die Idee ist die folgende: Das Protokoll π bzw. ρ simuliert *eine* Partei P_i der sicheren Funktionsauswertung P. Alle anderen Parteien der Funktionsauswertung kann die Umgebung $\mathcal{Z}$ simulieren. Darüber hinaus hat die Umgebung volle Kontrolle über die Kommunikation der Funktionsauswertung P. Das reale Protokoll π und das ideale Protokoll ρ unterscheiden sich nur darin, welche Eingaben die Partei P_i bekommt: Im ersten Fall erhält P_i immer die Eingabe $b_i = real$, im zweiten Fall erhält sie die Eingabe $b_i = ideal$ und alle weiteren Eingaben werden vom Simulator gewählt. Da die Umgebung diese Eingaben nicht direkt beobachten kann, muß sie anhand der Kommunikation von P_i oder anhand des Endergebnisses der Funktionsauswertung unterscheiden. Wegen der Sicherheit der Funktionsauswertung kann man anhand der Kommunikation nicht mehr lernen als aus dem Endergebnis. Die einzige Möglichkeit für die Umgebung zwischen realem und idealem Protokoll zu unterscheiden ist also das Endergebnis *out* der Funktionsauswertung. Und dieses – wieder wegen der Sicherheit der Funktionsauswertung – kann die Umgebung nur beeinflussen, indem sie Eingaben für die reaktive Funktion F findet, die die Ausgabe entsprechend beeinflussen.

Um zu zeigen, daß π so sicher wie ρ ist (bezüglich *spezieller* Sicherheit) genügt es also einzusehen, daß – für gegebene Umgebung – ein Simulator existiert, der die Eingaben für die Funktionsauswertung so wählen kann, daß das Ergebnis *out* der Funktionsauswertung in den folgenden zwei Fällen das gleiche ist: (i) die Partei P_i erhält die Eingabe $b_i = \mathit{real}$ oder (ii) die Partei P_i erhält die Eingabe $b_i = \mathit{ideal}$ und die vom Simulator gewählten Eingaben. Wir erinnern uns dazu an die Definition der ausgewerteten Funktion F (vgl. Definition 6.4): Es sei I die Menge der Parteien, die $b_i = \mathit{ideal}$ eingegeben haben. Wenn wir die Schwierigkeitsgrade der von diesen Parteien gelösten TL-Puzzles aufsteigend als $s_1 \leq \cdots \leq s_{\#I}$ anordnen, muß $s_\mu \geq k^\mu$ für jedes μ sein, damit $\mathit{out} = \mathit{success}$ ist.[8] Es sei q der höchste Schwierigkeitsgrad, den die Umgebung noch meistern kann. Wenn der Simulator dann seinen Schwierigkeitsgrad $s^* \geq kq$ wählt, so gilt zunächst einmal, daß $s_1 \leq \cdots \leq s_{\#I-1} \leq s^*$, wobei $s_1, \ldots, s_{n-1}$ die von der Umgebung gelösten TL-Puzzles sind.

Damit die Umgebung zwischen realem und idealem Modell unterscheiden kann, muß für die von ihr gewählten Schwierigkeitsgrade $s_i \geq k^i$ gelten. Andernfalls wird sowohl in einem realen als auch in einem idealen Protokollauf die Ausgabe $\mathit{out} = \mathit{failed}$ sein. Insbesondere muß also $s_{\#I-1} \geq k^{\#I-1}$ sein. Da aber nach Definition von q auch $s_{\#I-1} \leq q$, folgt $s^* = kq \geq k \cdot k^{\#I-1} \geq k^{\#I}$. Damit ist ideal wie real die Funktionsausgabe $\mathit{out} = \mathit{success}$, und auch in diesem Fall kann die Umgebung nicht unterscheiden.[9]

Wie im vorangegangenen Abschnitt gliedern wir den Beweis in drei Lemmata: In Lemma 6.13 betrachten wir das Modell, in dem unsere Parteien mit der idealen Funktion F kommunizieren. In Lemma 6.14 übertragen wir unsere Ergebnisse auf das Modell, in dem die Parteien stattdessen die sichere Funktionsauswertung P durchführen, und in Lemma 6.15 folgern wir schließlich, daß π so sicher wie ρ ist.

Wie bereits im vorangegangenen Abschnitt ist an dieser Stelle eine Warnung vor möglichen Verwirrungen angebracht. Bei unseren Reduktionen von Lemma zu Lemma wechseln die verschiedenen Entitäten wieder ihre Rollen: Der Simulator $\mathcal{S}$ aus der Definition der speziellen Sicherheit, der im dritten Lemma 6.15 vorkommt, spielt in den ersten beiden Lemmata 6.13 und 6.14 die Rolle der Umgebung Z_p der Funktionsauswertung. Die Umgebung $\mathcal{Z}$ der speziellen Sicherheit (zusammen mit dem Angreifer $\mathcal{A}$) aus dem dritten Lemma 6.15 hingegen wird im zweiten Lemma 6.14 zum Angreifer A, und taucht im ersten Lemma nur noch indirekt auf: Der Simulator S aus dem ersten Lemma 6.13 wird in Abhängigkeit vom Angreifer A aus dem zweiten Lemma 6.14 gewählt, er simuliert damit indirekt die Umgebung $\mathcal{Z}$ aus dem dritten Lemma 6.15. Der Leser ist also gehalten,

[8] Außerdem müssen natürlich auch alle TL-Puzzles gelöst werden.

[9] Aus technischen Gründen wird der tatsächlich konstruierte Simulator unter Umständen ein TL-Puzzle von einem noch höheren Schwierigkeitsgrad als kq lösen müssen. Siehe hierzu den Beweis von Lemma 6.14.

acht zu geben, nicht von ähnlichen Begriffen in den verschiedenen Szenarien darauf zu schließen, daß es sich um einander entsprechende Maschinen handelt.

Lemma 6.13 (Sicherheit im F-Modell)
Es sei F die reaktive Funktion aus Definition 6.4.

Die PITM Z_{real} sei eine Umgebung für die reaktive Funktion F, die bei Eingabe $(1^k, z, C)$ mit $C = \{1, \ldots, k\} \setminus \{i\}$ der Partei i für die erste Runde die Eingabe $(b_i, s_i) = (real, 0)$ gibt, und in allen weiteren Runden die leere Eingabe.[10] Zum Schluß gibt Z_{real} die Ausgabe der letzten Runde von Partei i aus.

Die PITM Z_p sei eine Umgebung für die reaktive Funktionsauswertung F mit dem folgenden Programm: Bei Eingabe $(1^k, z, C)$ mit $C = \{1, \ldots, k\} \setminus \{i\}$ gibt Z_p der Partei i für die erste Runde die Eingabe (b_i, s_i) mit $b_i = ideal$ und $s_i = p(k)$. Dann simuliert Z_p den Prover $C_{TL,p}$ (s. Seite 185) mit Eingabe $(1^k, s_i)$ und leitet die Ausgaben von Partei i an $C_{TL,p}$ (mit Ausnahme der Ausgabe der letzten Runde) und die Antworten von $C_{TL,p}$ wieder an Partei i. (Z_p löst also ein TL-Puzzle der Schwierigkeit $s_i = p(k)$.) Zum Schluß gibt Z_p die Ausgabe der letzten Runde von Partei i aus.

Dann gibt es für jede PITM S ein Polynom p, so daß die folgenden Familien von Zufallsvariablen komplexitätstheoretisch ununterscheidbar sind:

$$\begin{aligned} & \big\langle F_C, S(1^k, z_A, C), Z_{real}(1^k, z_Z, C) \big\rangle \\ \text{und} \quad & \big\langle F_C, S(1^k, z_A, C), Z_p(1^k, z_Z, C) \big\rangle. \end{aligned}$$

(Für $C \subseteq \{1, \ldots, k\}$ mit $\#C = k - 1$ und $z_A, z_Z \in \Sigma^*$.)

Beweis: Im folgenden sei C immer $C = \{1, \ldots, k\} \setminus \{i\}$, also i der Index der unkorrumpierten Partei.

Für eine beliebige Umgebung Z betrachten wir zunächst die Auswertung von F durch Umgebung Z und Angreifer S, also

$$\big\langle F_C, S(1^k, z_A, C), Z(1^k, z_Z, C) \big\rangle.$$

Es sei q ein Polynom. Wir ändern nun das Programm von F (vgl. Definition 6.4) wie folgt: Wenn für ein $j \in C$ (also $j \neq i$) für die Eingabe (b_j, s_j) gilt: $b_j = ideal$ und $s_j > q(k)$, wird für die Berechnung der Funktionsausgabe *out* angenommen, daß die j-te Instanz $\mathcal{V}_j$ des TL-Puzzle-Verifiers $\mathcal{V}$ eine 0 ausgegeben hat (unabhängig davon, was $\mathcal{V}_j$ tatsächlich ausgibt). Die resultierende reaktive Funktion nennen wir F^q.

[10] Diese Umgebung Z_{real} verhält sich also wie die Umgebung Z_{real} aus Lemma 6.10, nur daß sie für den Fall konstruiert ist, in dem nur eine Partei unkorrumpiert ist ($\#C = k - 1$).

Wegen der Schwierigkeitsbedingung der TL-Puzzles (Definition 5.3) gibt es nun ein (von S abhängiges, aber von Z unabhängiges) Polynom q, so daß die Wahrscheinlichkeit, daß S TL-Puzzles der Schwierigkeit $s > q(k)$ löst, vernachlässigbar ist. Es sind also für $Z \in \{Z_{real}, Z_p\}$

$$\left\langle F_C, S(1^k, z_A, C), Z(1^k, z_Z, C)\right\rangle \quad \text{und} \quad \left\langle F_C^q, S(1^k, z_A, C), Z(1^k, z_Z, C)\right\rangle \quad (6.6)$$

komplexitätstheoretisch ununterscheidbar.

Es sei $p := k \cdot q$.

Nun verändern wir F^q dahingehend, daß bei der Berechnung der Funktionsausgabe *out* immer davon ausgegangen wird, daß die i-te Instanz $\mathcal{V}_i$ des TL-Puzzle-Verifiers $\mathcal{V}$ die Ausgabe 1 hat (also das TL-Puzzle der von Z gesteuerten Partei i gelöst wurde). Die resultierende reaktive Funktion nennen wir $\tilde{F}_q$. Im Falle $b_i = real$ hat diese Änderung keinen Effekt, da in diesem Falle die Instanz $\mathcal{V}_i$ gar nicht verwendet wird. Im Zusammenspiel mit der Umgebung Z_p (für beliebiges p) löst Z_p das TL-Puzzle mit überwältigender Wahrscheinlichkeit (da sie den entsprechenden Prover $C_{TL,p}$ benutzt und immer $s_i := p(k)$ wählt). Somit sind für $Z \in \{Z_{real}, Z_p\}$

$$\left\langle F_C^q, S(1^k, z_A, C), Z(1^k, z_Z, C)\right\rangle \quad \text{und} \quad \left\langle \tilde{F}_C^q, S(1^k, z_A, C), Z(1^k, z_Z, C)\right\rangle \quad (6.7)$$

komplexitätstheoretisch ununterscheidbar.

Wir vergleichen nun die Auswertung von $\tilde{F}^q$ durch Umgebung Z_{real} bzw. Z_p. Die Kommunikation mit S vor der letzten Runde ist in beiden Fällen die gleiche (da bis zu dieser Runde die Ausgaben der verschiedenen Parteien unabhängig voneinander verarbeitet werden). Es genügt also einzusehen, daß die Ausgabe *out* der letzten Runde in beiden Fällen gleich ist. Im Falle der Auswertung durch Umgebung Z_{real} und Angreifer S wird die Ausgabe out_{real} wie folgt bestimmt:

- Es sei $I \cap C$ die Menge der Parteiindizes $j \in C$ mit $b_j = ideal$. (Man beachte hier, daß $b_i \neq ideal$.)
- Es seien $s_{j_1} \leq \cdots \leq s_{j_{\#(I \cap C)}}$ die Eingaben der Parteien $i_\nu \in I \cap C$, aufsteigend sortiert.
- Es ist $out_{real} = success$ genau dann, wenn (i) $s_j \leq q$ für alle $j \in I \cap C$ (diese Bedingung haben wir oben bei der Konstruktion von F^q hinzugefügt), (ii) alle $s_{j_\mu} \geq k^\mu$, (iii) alle $\mathcal{V}_j$ mit $j \in I \cap C$ die Ausgabe 1 liefern.

Dahingegen wird die Ausgabe out_p einer Auswertung von $\tilde{F}^q$ durch Umgebung $Z_p = Z_{kq}$ und Angreifer S wie folgt bestimmt:

- Es sei I die Menge der Parteiindizes $j \in \{1, \ldots, k\}$ mit $b_j = ideal$.
- Es seien $s_{j_1} \leq \cdots \leq s_{j_{\#I}}$ die Eingaben der Parteien $i_\nu \in I$, aufsteigend sortiert.

- Es ist $out_p = success$ genau dann, wenn (i) $s_j \leq q$ für alle $j \in I \cap C$, (ii) alle $s_{j_\mu} \geq k^\mu$, (iii) alle $\mathcal{V}_j$ mit $j \in I \cap C$ die Ausgabe 1 liefern (wir haben bei der Konstruktion von $\tilde{F}_q$ die Partei $i \notin C$ von dieser Bedingung ausgeschlossen).

Aus $out_p = success$ folgt $out_{real} = success$. Ist hingegen $out_{real} = success$, so gilt auch für eine Auswertung mit Z_p folgendes: Es existiert ein $j \in I \cap C$ mit $s_j \geq k^{\#(I\cap C)} = k^{\#I-1}$. Andererseits gilt $s_j \leq q$, somit ist $k^{\#I} \leq kq = p$. Damit ist $s_i = p \geq k^{\#I}$ (wobei i die unkorrumpierte Partei ist), und es folgt $out_p = success$. Damit haben out_{real} und out_p die gleiche Verteilung.

Somit ist aber auch

$$\langle \tilde{F}_C^q, S(1^k, z_A, C), Z_{real}(1^k, z_Z, C)\rangle \;=\; \langle \tilde{F}_C^q, S(1^k, z_A, C), Z_p(1^k, z_Z, C)\rangle.$$

Mit (6.6) und (6.7) folgt daraus, daß

$$\langle F_C, S(1^k, z_A, C), Z_{real}(1^k, z_Z, C)\rangle \quad \text{und} \quad \langle F_C, S(1^k, z_A, C), Z_p(1^k, z_Z, C)\rangle$$

komplexitätstheoretisch ununterscheidbar sind, und das Lemma ist bewiesen. □

Lemma 6.14 (Sicherheit im *P*-Modell)
Es sei P die von Satz 6.7 garantierte sichere Funktionsauswertung für F. Weiter seien Z_{real} und Z_p wie in Lemma 6.10.

Dann gibt es für jede PITM A (den Angreifer) ein Polynom p, so daß die folgenden Zufallsvariablen komplexitätstheoretisch ununterscheidbar sind:

$$\langle P_C, A(1^k, z_A, C), Z_{real}(1^k, z_Z, C)\rangle$$
$$\text{und} \quad \langle P_C, A(1^k, z_A, C), Z_p(1^k, z_Z, C)\rangle.$$

(Für $C \subseteq \{1, \ldots, k\}$ mit $\#C = k - 1$ und $z_A, z_Z \in \Sigma^*$.)

Beweis: Es sei A im folgenden fest. Da P eine sichere Funktionsauswertung von F ist (vgl. Satz 6.7) und damit insbesondere sicher ist bei einer unkorrumpierten Partei (vgl. Definition 6.6), existiert eine PITM S (der Simulator), so daß für jede PITM Z (die Umgebung) die folgenden Zufallsvariablen komplexitätstheoretisch ununterscheidbar sind:

$$\langle P_C, A(1^k, z_A, C), Z(1^k, z_Z, C)\rangle \quad \text{und} \quad \langle F_C, S(1^k, z_A, C), Z(1^k, z_Z, C)\rangle. \tag{6.8}$$

(Für $C \subseteq \{1, \ldots, k\}$ mit $\#C = k - 1$ und $z_A, z_Z \in \Sigma^*$.)

Nach Lemma 6.13 gibt es nun ein von S abhängiges Polynom p, so daß

$$\langle F_C, S(1^k, z_A, C), Z_{real}(1^k, z_Z, C)\rangle \quad \text{und} \quad \langle F_C, S(1^k, z_A, C), Z_p(1^k, z_Z, C)\rangle.$$

Durch zweimalige Anwendung von (6.8) (mit $Z := Z_{real}$ und mit $Z := Z_p$) erhalten wir daraus, daß

$$\left\langle P_C, A(1^k, z_A, C), Z_{real}(1^k, z_Z, C)\right\rangle \quad \text{und} \quad \left\langle P_C, A(1^k, z_A, C), Z_p(1^k, z_Z, C)\right\rangle$$

komplexitätstheoretisch ununterscheidbar sind. □

Bei der Reduktion von Lemma 6.14 auf Lemma 6.13 tritt der folgende interessante Effekt auf: Das Polynom p ist so gewählt, daß es das k-fache des höchsten TL-Puzzle-Schwierigkeitsgrads q, den S noch bewältigen kann, ist. Andererseits ist es denkbar, daß S wesentlich höhere Schwierigkeitsgrade bewältigen kann als A (denn es wird lediglich gefordert, daß S polynomiell-beschränkt ist, wir wissen aber nicht, in welchem konkreten Verhältnis die Laufzeit von S zu der von A steht). Somit genügt es nicht unbedingt, daß p das k-fache des höchsten von A bewältigbaren Schwierigkeitsgrads ist. Dies steht im Gegensatz zu unser Intuition in der Beweisskizze (Abschnitt 6.2 und Beginn dieses Abschnittes), bei der wir p direkt von den von der Umgebung lösbaren Schwierigkeitsgraden abhängig gemacht haben. Glücklicherweise ist die hier gezeigte schwächere Aussage für unsere Zwecke noch hinreichend, denn wir sind ja nur an der Existenz irgendeines Polynoms p interessiert. Der Grund für diesen Effekt liegt darin, daß eine sichere Funktionsauswertung nicht notwendigerweise „komplexitätserhaltend“ ist, es ist also nicht garantiert, daß Probleme, die gegenüber der idealen Funktion schwierig (aber noch in polynomiell-beschränkter Zeit lösbar) gegenüber der Implementierung P nicht deutlich einfacher werden.[11]

Lemma 6.15 (Sicherheit des unkomponierten Protokolls)
Es seien π und ρ wie in Definitionen 6.8 und 6.9. Dann ist π so sicher wie ρ bezüglich spezieller komplexitätstheoretischer Sicherheit sowohl mit als auch ohne Auxiliary input sowohl bzgl. der Sicht als auch bzgl. der Ausgabe der Umgebung.

[11] Es ist sogar einfach, ein konkretes Beispiel hierfür zu konstruieren: Um die Funktion F aus Definition 6.4 zu implementieren, verändern wir diese zunächst wie folgt: F' verhält sich wie F, erlaubt es Parteien aber, in der ersten Runde ein Flag *easy* zu setzen. Ist dieses gesetzt, so wird das i-te TL-Puzzle nicht mit Schwierigkeitsgrad s_i gestellt, sondern mit Schwierigkeitsgrad $\sqrt{s_i}$. Dann gibt es eine sichere Funktionsauswertung P' für F'. Daraus machen wir eine sichere Funktionsauswertung P für F: Die Parteien von P verhalten sich wie die Parteien von P', nur daß sie es nicht zulassen, daß das Flag *easy* gesetzt wird (durch unkorrumpierte Parteien).

Man überzeugt sich, daß (i) dies eine sichere Funktionsauswertung für F im Sinne der Definitionen 6.5 und 6.6 ist (der Simulator muß u. U. TL-Puzzles lösen, deren Schwierigkeit quadratisch gegenüber den vom Angreifer gelösten ist, aber auch das ist noch polynomiell-beschränkt), aber (ii) es für Lemma 6.14 nicht genügt, daß p das k-fache des höchsten Schwierigkeitsgrads ist, den A lösen könnte (denn A kann die korrumpierten Parteien das *easy*-Flag setzen lassen und somit wesentlich größere s_j eingeben.)

Beweis: Wegen Lemmata A.5 und A.10 ist es hinreichend, den Fall der Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung zu betrachten.

Es seien ein zulässiger polynomiell-beschränkter Angreifer $\mathcal{A}$ und eine zulässige polynomiell-beschränkte Umgebung $\mathcal{Z}$ gegeben. Um das Lemma zu beweisen, müssen wir zeigen, daß ein Simulator $\mathcal{S}$ existiert, so daß

$$\text{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) \quad \text{und} \quad \text{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z) \tag{6.9}$$

komplexitätstheoretisch ununterscheidbar sind.

Für ein Polynom p konstruieren wir hierzu zunächst einen Simulator $\mathcal{S}_p$ (vgl. Abbildung 6.3): Dieser hat den eingehenden Angreiferport `adv_output` und den ausgehenden Angreiferport `adv_input`. $\mathcal{S}_p$ zeige das folgende Verhalten:

- Bei der ersten Aktivierung (M_{ideal} aktiviert $\mathcal{S}_p$ mittels einer Nachricht *start*) schickt $\mathcal{S}_p$ über `adv_input` die Nachricht $s := p(k)$ an M_{ideal}. (M_{ideal} wandelt diese in die Eingabe $(ideal, s)$ für die simulierte Partei der Funktionsauswertung P um).
- Dann startet $\mathcal{S}_p$ eine simulierte Instanz von $C_{TL,p}$ mit der Eingabe $(1^k, s)$. (Diese löst TL-Puzzles mit einem Schwierigkeitsgrad $s = p(k)$.)
- Wann immer $\mathcal{S}_p$ danach über `adv_output` mit einer Nachricht m aktiviert wird, liefert $\mathcal{S}_p$ die Nachricht m an $C_{TL,p}$ und sendet die Antwort über `adv_input` an M_{ideal}.

Wir untersuchen nun einen Protokollauf von $\rho = \{M_{ideal}\}$ zusammen mit $\mathcal{Z}$, $\mathcal{A}$ und $\mathcal{S}_p$. (Vgl. Abbildung 6.3.)

Nun simuliert M_{ideal} eine Partei P_i der sicheren Funktionsauswertung P_i. Die Identität i der Partei wird von $\mathcal{Z}$ gewählt. Die Eingabenachrichten werden von $\mathcal{S}_p$ gewählt, die Eingabe der ersten Runde ist dabei $(ideal, s)$ mit $s = p(k)$. Die Ausgaben werden (mit Ausnahme der letzten Runde) an $\mathcal{S}_p$ geschickt. Die Ausgabe *out* der letzten Runde wird an $\mathcal{Z}$ geliefert. Die Kommunikationsnachrichten von P_i unterliegen der Kontrolle von $\mathcal{Z}$. O. B. d. A. können wir annehmen, daß die Umgebung $\mathcal{Z}$ eine Ausgabe der Form (x, out) hat, wobei *out* die Ausgabe der letzten Runde ist und x nicht von *out* abhängt.

Wir können also einen Angreifer A für die Funktionsauswertung P definieren, welcher $\mathcal{Z}$ und $\mathcal{A}$ simuliert, und die Kommunikationsnachrichten so beantwortet, wie $\mathcal{Z}$ es getan hätte. Wenn $\mathcal{Z}$ mit Ausgabe (x, out) terminiert, liefert A die Ausgabe x.

Weiterhin können wir aus dem Simulator $\mathcal{S}_p$ eine Umgebung Z_p für die Funktionsauswertung P konstruieren. Dieser liefert genau die Eingaben an P, die $\mathcal{S}_p$ liefern würde. Die Ausgabe der Umgebung Z_p sei die Ausgabe der letzten Runde der Funktionsauswertung. Diese Umgebung Z_p ist gerade die in Lemma 6.13 definierte.

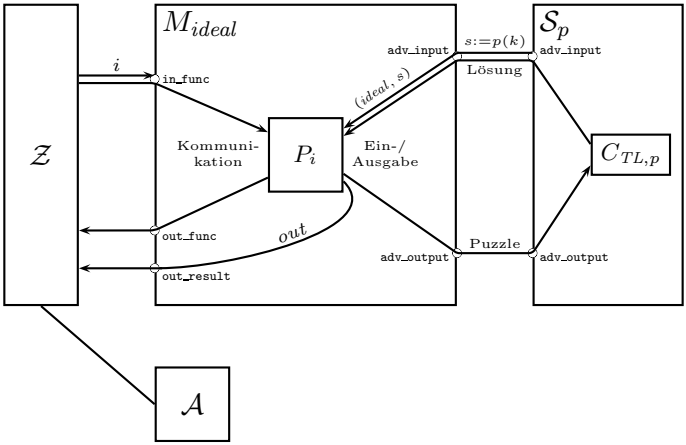


Abbildung 6.3.: Die ideale Protokollausführung. Der Simulator $\mathcal{S}_p$ wählt den Schwierigkeitsgrad s für das von P_i gestellte TL-Puzzle. Dann leitet er das TL-Puzzle zwecks Lösung an den TL-Puzzle-Prover $C_{TL,p}$. Die Kommunikation der Partei P_i sowie das Endresultat der Funktionsauswertung werden an die Umgebung $\mathcal{Z}$ weitergeleitet. Der Angreifer $\mathcal{A}$ ist nur mittelbar involviert, indem er mit der Umgebung kommuniziert.

Wenn i die Partei ist, die M_{ideal} simuliert, und $C = \{1, \ldots, k\} \setminus \{i\}$, dann ist also

$$\mathrm{OUTPUT}_{\rho,\mathcal{A},\mathcal{S}_p,\mathcal{Z}}(k,z) \;=\; \big\langle P_C, A(1^k, z, C), Z_p(1^k, \lambda, C)\big\rangle. \tag{6.10}$$

Wie im Beweis von Satz 5.10 bezeichne dabei in leichter Erweiterung unserer Notation $\mathrm{OUTPUT}_{\rho,\mathcal{A},\mathcal{S}_p,\mathcal{Z}}(k,z)$ die Ausgabe dieses Netzwerks bei Sicherheitsparameter k und Auxiliary input z. (Genaugenommen wird i von $\mathcal{Z}$ gewählt, aber wir können o. B. d. A. i als fest durch den Auxiliary input gegeben annehmen, und daher C in obiger Gleichung als Teil des Auxiliary inputs auffassen.)

Nun betrachten wir einen Protokollauf von $\pi = \{M_{real}\}$ zusammen mit $\mathcal{Z}$ und $\mathcal{A}$. Wie oben können wir aus $\mathcal{Z}$ und $\mathcal{A}$ den Angreifer A konstruieren. (Es handelt sich dabei um die gleiche PITM A wie oben.) Weiterhin wählt M_{real} die Eingabenachrichten für P_i fest: Die erste Nachricht ist $(real, 0)$, alle weiteren sind leer. Konstruieren wir also analog wie oben die entsprechende Umgebung, so erhalten wir gerade die Umgebung Z_{real} aus Lemma 6.13. Es ist also

$$\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) \;=\; \big\langle P_C, A(1^k, z, C), Z_{real}(1^k, \lambda, C)\big\rangle. \tag{6.11}$$

Nach Lemma 6.14 gibt es ein Polynom p, so daß die rechten Seiten von (6.10) und (6.11) komplexitätstheoretisch ununterscheidbar sind. Damit sind auch die linken Seiten

$$\mathrm{OUTPUT}_{\rho,\mathcal{A},\mathcal{S}_p,\mathcal{Z}}(k,z) \quad \text{und} \quad \mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$$

komplexitätstheoretisch ununterscheidbar.

Konstruieren wir nun aus $\mathcal{A}$ und $\mathcal{S}_p$ den zulässigen Simulator $\mathcal{S}$, welcher beide Maschinen simuliert, so erhalten wir

$$\mathrm{OUTPUT}_{\rho,\mathcal{A},\mathcal{S}_p,\mathcal{Z}}(k,z) = \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)$$

und es folgt (6.9). □

6.8. Zusammenfassung

Wir haben nun alle Tatsachen bewiesen, die wir für Satz 6.3 benötigen. Der Übersicht halber wiederholen wir den Satz an dieser Stelle:

Satz 6.3 (Spezielle Sicherheit genügt nicht für nebenläufige Komposition)

(Fortsetzung nächste Seite)

(Fortsetzung)

Wenn Time-lock puzzles (Definition 5.3) und Enhanced trapdoor permutations (Definition 1.9) existieren, dann gibt es *polynomiell-beschränkte* Protokolle π und ρ, so daß folgendes gilt:

- Das Protokoll π ist so sicher wie ρ bezüglich *spezieller* komplexitätstheoretischer Sicherheit sowohl mit als auch ohne Auxiliary input sowohl bzgl. der Sicht als auch bzgl. der Ausgabe der Umgebung.
- Für jede unbeschränkte effizient berechenbare Funktion f ist $f \cdot \pi$ *nicht* so sicher wie $f \cdot \rho$ bezüglich spezieller komplexitätstheoretischer Sicherheit weder mit noch ohne Auxiliary input und weder bzgl. der Sicht noch bzgl. der Ausgabe der Umgebung.

Beweis: Die Protokolle sind die aus Definitionen 6.8 und 6.9. Diese existieren unter den Annahmen des Theorems (wir benutzen die Existenz der TL-Puzzles für die Existenz von $\mathcal{V}$, und die der Enhanced trapdoor permutations für die Existenz von P, die beide in die Konstruktion von π und ρ einfließen). Nach Lemma 6.15 ist die erste Aussage des Satzes erfüllt, nach Lemma 6.12 die zweite. □

Wir erhalten damit auch unmittelbar:

Korollar 6.16 (Spezielle Sicherheit impliziert nicht allgemeine Komponierbarkeit)
Wenn Time-lock puzzles (Definition 5.3) und Enhanced trapdoor permutations (Definition 1.9) existieren, dann gibt es *polynomiell-beschränkte* Protokolle π und ρ, so daß folgendes gilt:

- Das Protokoll π ist so sicher wie ρ bezüglich *spezieller* komplexitätstheoretischer Sicherheit sowohl mit als auch ohne Auxiliary input sowohl bzgl. der Sicht als auch bzgl. der Ausgabe der Umgebung.
- Das Protokoll π ist *nicht* so sicher wie ρ bezüglich polynomiell-beschränkter *allgemeiner* Komponierbarkeit sowohl mit als auch ohne Auxiliary input.

7. Spieltheorie und exponentielle Angreifer

In diesem Kapitel werden wir den Begriff der statistischen Sicherheit (in der allgemeinen und der speziellen Variante) als Spiel im Sinne der Spieltheorie zwischen Umgebung und Simulator auffassen. Diese Sichtweise ermöglicht es uns, Ergebnisse der Spieltheorie, insbesondere die Existenz des Nash-Equilibriums auf unser Szenario anzuwenden. Dem Nash-Equilibrium entspricht in unserem Modell der Begriff der universellen Umgebung und des universellen Simulators. Diese haben die Eigenschaft, daß wir o. B. d. A. den Sicherheitsbegriff auf diese Umgebung und diesen Angreifer einschränken können. Hieraus erhalten wir zwei wichtige Korollare: Zum einen fallen im Falle von Protokollen mit beschränkter Kommunikationskomplexität spezielle und allgemeine Sicherheit zusammen. Darüber hinaus zeigen wir, daß im Falle polynomiell-beschränkter Protokolle die universelle Umgebung und der universelle Simulator höchstens exponentielle Laufzeit haben. Damit ergibt sich, daß wir o. B. d. A. alle Angreifer, Umgebungen und Simulatoren als exponentiell-beschränkt annehmen dürfen.

Um die Ergebnisse dieses Kapitels im Detail untersuchen zu können, brauchen wir zunächst einige Grundlagen der Spieltheorie, die wir im nächsten Abschnitt vorstellen werden.

7.1. Grundlagen der Spieltheorie

Wir geben in diesem Abschnitt einen kurzen Überblick über die wichtigsten Konstrukte der Spieltheorie und die für dieses Kapitel benötigten Ergebnisse. Diese Einführung erhebt keinerlei Anspruch auf Vollständigkeit oder darauf, eine repräsentative Auswahl zu treffen. Für eine ausführlichere Einführung in die Problemstellungen und Methoden der Spieltheorie sei der Leser auf entsprechende Lehrbücher verwiesen, z. B. [Ras89].

7.1.1. Spiele in Normalform

Der grundlegende Begriff der Spieltheorie ist der des *Spiels*. In seiner allgemeinsten Form wurde ein n-Spieler-Spiel G von [vNM44] als durch seine sog. *Normalform*[1] beschrieben definiert: Jedem Spieler P_i wird eine Menge $\mathfrak{S}_i$ von sog. *reinen Strategien* zugeordnet. Intuitiv beschreibt eine Strategie, in welcher Situati-

[1] In [vNM44] hatte die Normalform diesen Namen jedoch noch nicht, [vNM44] spricht einfach von der allgemeinen formalen Beschreibung strategischer Spiele.

(a)

	Stein	*Schere*	*Papier*
Stein	$0, 0$	$1, -1$	$-1, 1$
Schere	$-1, 1$	$0, 0$	$1, -1$
Papier	$1, -1$	$-1, 1$	$0, 0$

(b)

	Schweigen	*Aussage*
Schweigen	$-1, -1$	$-11, 0$
Aussage	$0, -11$	$-10, -10$

Abbildung 7.1.: Normalform des Spiels Stein-Schere-Papier (a) und des Gefangenendilemmas (b).
Jede Normalform ist durch eine Tabelle dargestellt, die einer Kombination von Strategien s_1, s_2 ein Paar h_1, h_2 zuordnet, wobei h_i der Gewinn $H_i(s_1, s_2)$ von Spieler i ist. Die Zeilen entsprechen Spieler 1, die Spalten Spieler 2.

on der Spieler P_i welche Entscheidung trifft. Spielen nun die Spieler $P_1, \ldots, P_n$ das Spiel G, so verfolgt jeder eine Strategie s_i. Wir nehmen an, daß für jede solche Kombination von Strategien festliegt, welchen Gewinn jeder Spieler am Ende des Spiels davonträgt (falls das Spiel probabilistisch ist, kann man hier z. B. den Erwartungswert des Gewinns verwenden). Die *Normalform* eines Spiels besteht also aus einem Tupel von reellwertigen *Gewinnfunktionen* H_i für die Spieler $i = 1, \ldots, n$, wobei $H_i(s_1, \ldots, s_n)$ den Gewinn des Spielers i beschreibt, wenn die Spieler die Strategien $s_1, \ldots, s_n$ verfolgen.

Um dieses Konzept zu illustrieren, stellen wir zwei klassische Beispiele der Spieltheorie in Normalform vor. Das erste ist das bekannte Spiel *Stein-Schere-Papier*. Es handelt sich um ein Zwei-Spieler-Spiel, bei dem jeder Spieler (unabhängig vom anderen) ein von drei Möglichkeiten wählt: Stein, Schere oder Papier. Es gelten die Regeln, daß der Stein immer die Schere besiegt (die Schere wird stumpf, wenn sie den Stein zu schneiden versucht), die Schere immer das Papier (da die Schere das Papier zerschneiden kann), und das Papier den Stein (das Papier kann den Stein einwickeln). Wählen beide Spieler das gleiche Objekt, so endet das Spiel unentschieden. Die Menge der Strategien für Spieler 1 und 2 ist also $\mathfrak{S}_1 = \mathfrak{S}_2 = \{\mathit{Stein}, \mathit{Schere}, \mathit{Papier}\}$, und wenn wir annehmen, daß ein gewonnenes Spiel den Wert 1, ein verlorenes den Wert -1, und ein unentschiedenes den Wert 0 hat, so erhalten wir die in Abbildung 7.1a dargestellte Normalform.

Bei dem zweiten Beispiel handelt es sich um das *Gefangenendilemma*. Wir stellen uns vor, zwei Verbrecher P_1 und P_2 seien von der Polizei festgenommen worden. Beiden wurde ein Verbrechen nachgewiesen, welches eine zwölfmonatige Inhaftierung ermöglicht. Beide haben noch ein weiteres Verbrechen begangen,

welches eine zehnjährige Inhaftierung nach sich ziehen würde. Der Polizei fehlen jedoch die Beweise, deshalb wird dem Verbrecher P_1 der folgende Handel vorgeschlagen: Wenn er gegen den anderen Verbrecher P_2 aussagt, wird ihm (P_1) die zwölfmonatige Strafe erlassen. Dem Verbrecher P_2 wird parallel in einem anderen Raum (so daß die Verbrecher sich nicht koordinieren können) der gleiche Handel vorgeschlagen. Die Verbrecher haben nun zwei Handlungsalternativen $\mathfrak{S}_1 = \mathfrak{S}_2 = \{Schweigen, Aussage\}$, und wenn wir jedem Jahr Gefängnis einen Verlust von 1 zuordnen, so erhalten wir die in Abbildung 7.1b dargestellte Normalform dieses Spiels. Schweigen beide, so erhält jeder zwölf Monate Gefängnis. Sagt einer aus, so kommt er straffrei davon, der andere aber erhält sowohl die zehnjährige als auch die zwölfmonatige Strafe. Sagen beide aus, so werden beide zu zehn Jahren verurteilt, da ihnen die zwölfmonatige Strafe erlassen wird.

Die wesentliche Problemstellung der Spieltheorie ist, neben der Modellierung eines Spiels, die Bestimmung empfehlenswerter Spielstrategien bzw. die Vorhersage des Verhaltens der Spieler bzw. des Ausgangs des Spiels. Diese beiden Aufgabenstellungen sind eng verwandt: Wenn wir, wie in der Spieltheorie üblich, alle Spieler als rational annehmen, d. h. davon ausgehen, daß jeder Spieler das tut, was für ihn am besten ist, so ist die Vorhersage gerade die, daß jeder Spieler das tut, was wir empfehlen würden. Andersherum brauchen wir, um eine Empfehlung auszusprechen, eine Vorhersage darüber, was die anderen Spieler tun werden. Aufgrund dieser gegenseitigen Abhängigkeit ist es nur in wenigen Spezialfällen möglich zu definieren, was eine optimale Strategie ist. Stattdessen behilft man sich üblicherweise mit dem Konzept eines *Gleichgewichts*. Man geht also davon aus, daß die Wahl der Strategien der einzelnen Spieler zueinander passen, so daß in einer vom Typ des Gleichgewichts abhängigen Weise jede der Strategien in ebendieser Situation die beste Wahl ist. Das bekannteste (aber bei weitem nicht das einzige) Gleichgewicht ist das *Nash-Equilibrium*, bei dem ein Tupel von Strategien $s_1, \ldots, s_n$ im Gleichgewicht stehen, wenn keiner der Spieler einen Vorteil davon hat, seine Strategie zu ändern (vorausgesetzt, die anderen Spieler bleiben bei ihrer Strategie). Aus der Sicht der Kryptologie könnte man das Nash-Equilibrium vielleicht wie folgt motivieren: Das Nash-Equilibrium ist eine Spielanweisung für alle Spieler (ein Protokoll) mit der Eigenschaft, sicher gegen eine korrumpierte Partei zu sein. Dabei betrachten wir nur „egoistische“ Angreifer, die nur dann einen Angriff fahren, wenn dieser ihnen auch einen Vorteil bringt. Wir definieren das Nash-Equilibrium nun formal:

Definition 7.1 (Nash-Equilibrium)
Es sei ein n-Spieler-Spiel G mit Mengen $\mathfrak{S}_1, \ldots, \mathfrak{S}_n$ von Strategien und

(Fortsetzung nächste Seite)

(Fortsetzung)

mit Gewinnfunktionen H_i gegeben. Ein Tupel von Strategien $(s_1, \ldots, s_n)$ ist ein *Nash-Equilibrium*, wenn für jedes $i \in \{1, \ldots, n\}$ und jedes $s^* \in \mathfrak{S}_i$ gilt:

$$H_i(s_1, \ldots, s_n) \geq H_i(s_1, \ldots, s_{i-1}, s^*, s_{i+1}, \ldots, s_n).$$

Je nachdem, welche Typen von Strategien wir zulassen, unterscheidet man verschiedene Typen von Nash-Equilibria. Bislang kennen wir nur reine Strategien, man spricht dann auch von einem Nash-Equilibrium in reinen Strategien. Weiter unten werden wir noch Nash-Equilibria in gemischten und in Verhaltensstrategien kennenlernen.

Wir wollen nun die zwei obigen Beispiele daraufhin untersuchen, welche Nash-Equilibria sie haben. Wir beginnen mit dem Gefangenendilemma. Wir betrachten zunächst den Spieler P_1. Egal welche Strategie P_2 fährt, für P_1 ist es in beiden Fällen besser, auszusagen (es erspart ihm in jedem Fall ein Jahr Gefängnis). Somit kann es kein Nash-Equilibrium geben, in dem P_1 schweigt. Analog sehen wir, daß auch P_2 nicht schweigen wird. Die Situation, in der beide Spieler aussagen jedoch, stellt ein Nash-Equilibrium dar: Wenn einer der Spieler stattdessen schweigt, so führt das zu einem zusätzlichen Jahr Gefängnis. Somit ist das einzige Nash-Equilibrium des Gefangenendilemmas (*Aussage*, *Aussage*). Das Gefangenendilemma ist insofern interessant, daß es ein Beispiel dafür liefert, daß das Nash-Equilibrium nicht unbedingt die Situation darstellt, die für alle Spieler am besten ist. So würden beide Spieler die Situation (*Schweigen*, *Schweigen*) bevorzugen. Das Nash-Equilibrium aber sagt voraus, daß sich diese Situation aufgrund einer Art Eskalation nicht einstellen wird.[2]

Nun betrachten wir das Spiel Stein-Schere-Papier. Hier stellen wir fest, daß es kein Nash-Equilibrium in reinen Strategien gibt: Für jedes Paar von Strategien (s_1, s_2) wird einer der Spieler nicht gewinnen, sagen wir P_2. Es gibt dann aber für gegebenes s_1 eine Strategie s^* für P_2, mit der P_2 gewinnt. Daher wird Spieler P_2 die Situation (s_1, s^*) bevorzugen, und (s_1, s_2) ist kein Nash-Equilibrium (in reinen Strategien).

Man wird nun einwenden, daß es doch eine optimale Strategie für Stein-Schere-Papier gibt: Man wählt einfach zufällig, für welche der drei Aktionen man sich entscheidet, der Gegner kann dann seine Strategie nicht darauf abstimmen. Um diese Beobachtung fassen zu können, führen wir nun den Begriff einer *gemischten Strategie* ein. Eine gemischte Strategie ist eine Strategie, die den Spieler anweist, zufällig gemäß einer gegebenen Verteilung eine reine Strategie zu wählen und

[2] Es werden hier natürlich Effekte wie z. B. ein Ehrenkodex oder die Möglichkeit späterer Repressalien vernachlässigt.

diese anzuwenden. Formal ist also eine gemischte Strategie eine Wahrscheinlichkeitsverteilung μ_i auf der Menge der reinen Strategien $\mathfrak{S}_i$. Wenden die Spieler die gemischten Strategien $\mu_1, \ldots, \mu_n$ an, so ist der Gewinn $H_i(\mu_1, \ldots, \mu_n)$ für Spieler P_i der Erwartungswert des Gewinns $H_i(s_1, \ldots, s_n)$, wenn die reinen Strategien s_j unabhängig jeweils entsprechend der Verteilung μ_i gezogen werden. Die Menge der gemischten Strategien für Spieler P_i wollen wir mit $\mathfrak{M}_i$ bezeichnen. Die Definition des Nash-Equilibriums ist nun direkt auf gemischte Strategien anwendbar: Ein Nash-Equilibrium in gemischten Strategien ist ein Tupel $(\mu_1, \ldots, \mu_n)$ von gemischten Strategien μ_i, so daß kein Spieler seinen (erwarteten) Gewinn steigern kann, indem er eine andere gemischte Strategie wählt.

Wir wenden uns wieder dem Spiel Stein-Schere-Papier zu. Ein Beispiel für eine gemischte Strategie ist $\mu \in \mathfrak{M}_1 = \mathfrak{M}_2$, welche jeder reinen Strategie $s \in \{\mathit{Stein}, \mathit{Schere}, \mathit{Papier}\}$ die Wahrscheinlichkeit $\frac{1}{3}$ zuordnet. Man überzeugt sich leicht davon, daß für jede gemischte Strategie $\mu' \in \mathfrak{M}_1 = \mathfrak{M}_2$ gilt: $H_1(\mu', \mu) = 0$ und $H_2(\mu, \mu') = 0$. In anderen Worten, wenn ein Spieler die Strategie μ wählt, so ist der Gewinn des anderen immer 0, unabhängig von der Wahl der Strategie des anderen. Insbesondere hat der andere dann keine Veranlassung, seine Strategie zu ändern. Damit ist (μ, μ) ein Nash-Equilibrium in gemischten Strategien, was unserer Intuition entspricht, daß es bei Stein-Schere-Papier am sinnvollsten ist, seine Entscheidung zufällig zu wählen, um dem Gegner die Möglichkeit zu nehmen, seine Entscheidung anzupassen. Man kann leicht nachrechnen, daß (μ, μ) in der Tat das einzige Nash-Equilibrium ist. (Es gibt aber durchaus Spiele, in denen mehrere Nash-Equilibria existieren.) Ein großer Vorteil des Konzepts des Nash-Equilibriums ist die folgende in [Nas50] gezeigte Tatsache:

Satz 7.2 (Existenz von Nash-Equilibria [Nas50])
Jedes endliche[3] Spiel hat ein Nash-Equilibrium in gemischten Strategien.

Eine wesentliche Eigenschaft, die man einem Spiel zuordnen kann, ist die folgende: Ein Spiel ist ein *Nullsummenspiel*, wenn für jede Kombination $(s_1, \ldots, s_n)$ gilt, daß $\sum_i H_i(s_1, \ldots, s_n) = 0$. In anderen Worten, die verschiedenen Parteien sind echte Gegner in dem Sinne, daß wann immer ein Spieler einen Vorteil erringt, ein anderer dafür büßen muß. Offensichtlich ist diese Definition unabhängig davon, ob man nur reine Strategien betrachtet, oder auch gemischte Strategien zuläßt. Von den oben untersuchten Spielen ist nur Stein-Schere-Papier ein Nullsummenspiel. Zwei-Spieler-Nullsummenspiele haben den Vorteil, sich besonders einfach untersuchen zu lassen. So war Satz 7.2 für Zwei-Spieler-Nullsummenspiele bereits von [vNM44] gezeigt worden. Eine wichtige Eigenschaft eines Nash-Equilibriums für Zwei-Spieler-Nullsummenspiele ist, daß wenn

[3] Ein Spiel heißt endlich, wenn jeder Spieler nur endlich viele reine Strategien hat.

(s_1, s_2) und (s'_1, s'_2) Nash-Equilibria bilden, dann sind auch (s'_1, s_2) und (s_1, s'_2) welche. Darüber hinaus haben alle Nash-Equilibria für beide Parteien den gleichen Gewinn. Dies ist im allgemeinen nicht selbstverständlich. Ein Spiel z. B., bei dem beide Spieler Gewinn 1 haben, wenn sie die gleiche (reine) Strategie $s \in \mathfrak{S}_1 = \mathfrak{S}_2$ wählen, und Gewinn 0 sonst, hat offensichtlich für jede Strategie $s \in \mathfrak{S}_1$ ein Nash-Equilibrium (s, s). Aber wählt ein Spieler eine Strategie s_1 aus einem Nash-Equilibrium, der andere aber eine Strategie s_2 aus einem anderen, so führen diese Strategien zusammen zu dem geringeren Gewinn 0. Die Tatsache, daß dies bei Zwei-Spieler-Nullsummenspielen nicht passieren kann, impliziert also die Existenz optimaler Strategien (nämlich die, die Teil eines Nash-Equilibriums sind), währen im allgemeinen Optimalität nur relativ zu einem bestimmten Nash-Equilibrium definierbar ist. Darüber hinaus kann bei einem Zwei-Spieler-Nullsummenspiel nicht der Effekt eintreten, den wir beim Gefangenendilemma beobachtet haben, daß es eine Kombination von Strategien gibt, die für alle Spieler besser als das Nash-Equilibrium ist. Für die Ergebnisse in diesem Kapitel werden wir nur Zwei-Spieler-Nullsummenspiele benötigen. Bei einem Zwei-Spieler-Nullsummenspiel genügt es, die Gewinnfunktion $H := H_1$ des Spielers P_1 anzugeben, da $H_2 = -H_1$. Dementsprechend werden wir im folgenden bei der Untersuchung solcher Spiele nur die Funktion H angeben.

7.1.2. Spiele in Extensivform

Die Normalform allein gibt zwar an, bei welcher Kombination von Strategien welcher Ausgang zu erwarten ist, aber sie versteckt vor uns die innere Struktur des Spiels. Wir wollen dies an einem weiteren Beispiel illustrieren: Spieler P_1 will Spieler P_2 eine Ware verkaufen. Spieler P_2 kann zunächst die Entscheidung treffen, ob er bezahlt. Danach entscheidet P_1 (in Abhängigkeit davon, ob P_2 bezahlt), ob er P_2 die Ware überreicht. Versuchen wir nun die Normalform dieses Spiels darzustellen, müssen wir zunächst alle reinen Strategien der Spieler aufzählen: P_2 hat die Strategien *zahlen* und *nicht zahlen* und P_1 die Strategien *Ware immer geben*, *Ware nie geben*, *Ware geben falls gezahlt* und *Ware geben falls nicht gezahlt*. Da eine reine Strategie jedem möglichen Spielverlauf eine Aktion zuordnen muß, ist die Anzahl der reinen Strategien exponentiell in der Anzahl der Spielverläufe und damit doppelt exponentiell in der Länge des Spiels. Ein komplexes Spiel über seine Normalform anzugeben ist also vergleichbar damit, eine Formel über ihre Wertetabelle zu repräsentieren. Um diesem Problem beizukommen, verwendet man die sogenannte *Extensivform*, die nicht nur den Strategien einen Gewinn zuordnet, sondern die möglichen Verläufe des Spiels darstellt.

In der Extensivform besteht ein n-Spieler-Spiel aus einem *Spielbaum*, einer *Gewinnfunktion* und einer Menge von *Informationsmengen*. Wir betrachten zu-

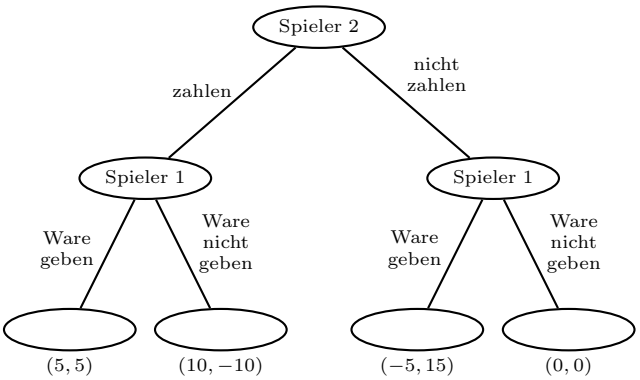
nächst den Spielbaum: Jeder Spielsituation[4] ordnet man in diesem Baum einen Knoten zu. Die Wurzel ist der Spielbeginn. Jedem inneren Knoten v ist dabei ein Typ zugeordnet: v kann ein *Spieler-i-Knoten* sein für $i \in \{1, \ldots, n\}$ oder ein *Zufallsknoten*. Ein Spieler-i-Knoten stellt eine Situation dar, in der Spieler i am Zug ist. Bei einem Zufallsknoten ist die nächste Spielsituation zufällig (gemäß einer festen Verteilung). Jeder ausgehenden Kante eines Spieler-i-Knotens v ist eine *Aktion* zugeordnet. Befindet sich das Spiel in der Situation v, und Spieler i führt die Aktion a aus, so ist die nächste Spielsituation der über die mit Aktion a markierte Kante von v aus erreichbare Knoten. Ein Spieler kann in einer Spielsituation keine Aktion treffen, die nicht an einer der ausgehenden Kanten steht, und die Aktionen an den von einem Knoten ausgehenden Kanten müssen alle verschieden sein. Einem Zufallsknoten hingegen ist eine Wahrscheinlichkeitsverteilung auf den ausgehenden Kanten zugeordnet, die angibt, mit welcher Wahrscheinlichkeit welcher Kante gefolgt wird. Der Spielbaum unseres einfachen Verkaufsspiels ist in Abbildung 7.2a dargestellt.

Die *Gewinnfunktion* H_i für Spieler i ordnet nun jedem Blatt des Spielbaums (ein Blatt stellt das Ende des Spiels dar) den Gewinn zu, den Spieler i erzielt, wenn das Spiel in dieser Situation endet. Wir haben in Abbildung 7.2a die Gewinnfunktion für beide Spieler unter den Blättern des Spielbaums aufgetragen. Wir nahmen dabei an, daß die Ware für den Verkäufer (Spieler 2) den Wert 5 habe, für den Käufer den Wert 15, und daß der Käufer den Betrag 10 zahle.

Eine reine Strategie für Spieler i ist nun eine Funktion, die jedem Spieler-i-Knoten eine der dort verfügbaren Aktionen zuordnet. Für Strategien $s_1, \ldots, s_n$ ist die Gewinnfunktion $H_i(s_1, \ldots, s_n)$ der Erwartungswert der Gewinnfunktion H_i angewandt auf das Blatt, das erreicht wird, wenn jedem Spieler-j-Knoten der nächste Knoten gemäß Strategie s_j gewählt wird (wir bilden den Erwartungswert, da selbst bei reinen, also deterministischen Strategien wegen der Zufallsknoten das erreichte Blatt zufällig sein kann). Angewandt auf unser Beispiel erhält man die bereits oben aufgezählten reinen Strategien und die in Abbildung 7.2b dargestellte Normalform. Man erkennt übrigens leicht, daß das Nash-Equilibrium in diesem Spiel daraus besteht, daß der Verkäufer nie die Ware gibt und der Käufer nicht zahlt.

Bislang haben wir nur den Spielbaum und die Gewinnfunktion der Extensivform erläutert. Um die volle Allgemeinheit der Extensivform nutzen zu können, müssen wir aber noch das Konzept der *Informationsmengen* kennenlernen. Dazu betrachten wir nochmals das Gefangenendilemma. Hier waren wir davon ausgegangen, daß beide Spieler ihre Entscheidung unabhängig voneinander treffen. Versuchen wir aber den Spielbaum des Gefangenendilemmas zu entwerfen, so

[4] Dabei beinhaltet eine Spielsituation nicht nur den aktuellen Zustand des Spiels, sondern auch dessen bisherigen Verlauf, andernfalls würden wir bei Spielen, die Zyklen erlauben, keinen Spiel*baum* erhalten.



(a)

	zahlen	*nicht zahlen*
Ware immer geben	$5, 5$	$-5, 15$
Ware nie geben	$10, -10$	$0, 0$
Ware geben falls gezahlt	$5, 5$	$0, 0$
Ware geben falls nicht gezahlt	$10, -10$	$-5, 15$

(b)

Abbildung 7.2.: Die Extensivform.

(a) Der Spielbaum mit der Gewinnfunktion. Das Spiel beginnt mit einem Spieler-2-Knoten, in dem Spieler 2 die Wahl zwischen den Aktionen *zahlen* und *nicht zahlen* hat. Abhängig von dieser Entscheidung hat Spieler 1 dann die Wahl zwischen den Aktionen *Ware geben* und *Ware nicht geben*. Die Gewinnfunktion ist durch die Paare (*Gewinn von Spieler 1*, *Gewinn von Spieler 2*) unter den Blättern spezifiziert.

(b) Die zu dem in (a) in Extensivform dargestellten Spiel gehörige Normalform. Spieler 2 hat seine Aktionen *zahlen* und *nicht zahlen* als reine Strategien. Spieler 1 aber hat vier reine Strategien, da er seine Aktion abhängig von der Aktion von Spieler 2 wählen kann.

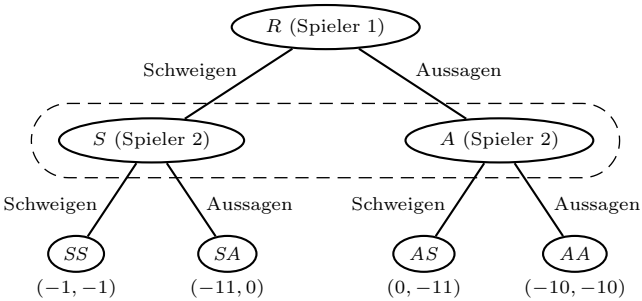


Abbildung 7.3.: Die Extensivform des Gefangenendilemmas.
In diesem Spiel führt zunächst Spieler 1 eine Aktion *Schweigen* oder *Aussagen* aus. Danach kann Spieler 2 zwischen *Schweigen* oder *Aussagen* wählen. Damit diese Entscheidung unabhängig davon ist, ob wir uns in Knoten S oder A befinden, also davon, welche Entscheidung Spieler 1 getroffen hat, befinden sich die Knoten S und A in einer Informationsmenge, sind also für Spieler 2 ununterscheidbar. Wir kennzeichnen diese Informationsmenge durch eine gestrichelte Linie um S und A. (Alle anderen Informationsmengen sind einelementig und daher nicht eingezeichnet.)

stellen wir fest, daß zunächst ein Spieler seine Entscheidung trifft, und erst dann der andere (wir können also keine Gleichzeitigkeit ausdrücken). Der Spielbaum des Gefangenendilemmas wird also wie in Abbildung 7.3 aussehen (die gestrichelte Umrandung möge der Leser noch ignorieren). Wir haben willkürlich Spieler 1 den ersten Zug gegeben. In dieser Darstellung hat nun aber Spieler 2 mehr Möglichkeiten: Da er die Entscheidung von Spieler 1 kennt, kann er seine Entscheidung von der von Spieler 1 abhängig machen und hat somit zusätzlich die Strategien: *Schweigen, wenn der andere schweigt* und *Aussage, wenn der andere schweigt*. Dies ist offensichtlich nicht das Originalspiel, wir brauchen also eine Möglichkeit auszudrücken, daß Spieler 2 zum Zeitpunkt seiner Entscheidung nicht weiß, wie Spieler 1 entschieden hat (daß also ein Spiel mit *imperfekter Information* und nicht eines mit *perfekter Information* vorliegt). In anderen Worten, wir brauchen eine Möglichkeit, um festzulegen, daß Spieler 2 die Knoten S und A nicht unterscheiden kann. Hierzu dienen die Informationsmengen. Diese sind eine Partition der Menge der Spieler-Knoten des Spielbaums. Zwei Knoten sind in der gleichen Informationsmenge, wenn sie für den Spieler am Zug nicht unterscheidbar sind. Im Falle des Gefangenendilemmas befinden sich also S und A in der gleichen Informationsmenge (dies drücken wir durch die gestrichelte Umrandung in Abbildung 7.3 aus). Die Menge der Informationsmengen ist also $\{\{S, A\}, \{R\}\}$. (Die Blätter befinden sich nicht in Informationsmengen, da in den Blättern keine Entscheidung getroffen wird.) Unter Berücksichtigung der

Informationsmengen müssen wir genauer spezifizieren, welche reinen Strategien zulässig sind. So darf z. B. die Strategie *Schweigen, wenn der andere schweigt* keine zulässige Strategie sein. Eine reine Strategie muß auf verschiedenen Knoten in der *gleichen* Informationsmenge die *gleiche* Entscheidung treffen. Somit sind in Abbildung 7.3 die Strategien von Spieler 2 nur *Schweigen* und *Aussagen*, es ergibt sich die Normalform aus Abbildung 7.1b.

Zusammenfassend ist die Extensivform also wie folgt definiert:

Definition 7.3 (Extensivform)
Ein n-Spieler-Spiel G in *Extensivform* besteht aus *Spielbaum* G, *Gewinnfunktionen* $H_1, \ldots, H_n$ und der Menge $\mathfrak{I}$ der *Informationsmengen* (die *Informationspartition*).

Der Spielbaum G ist ein Baum mit ausgezeichneter Wurzel. Jedem inneren Knoten von G ist ein Typ zugeordnet, der entweder *Zufallsknoten* oder *Spieler-i-Knoten* für ein $i \in \{1, \ldots, n\}$ sein kann.

Jeder von einem Spieler-i-Knoten ausgehenden Kante ist eine Aktion a zugeordnet. Keinen zwei vom gleichen Knoten ausgehenden Kanten ist die gleiche Aktion zugeordnet. Die Menge der den von einem Knoten v ausgehenden Kanten zugeordneten Aktionen nennen wir die *im Knoten v möglichen Aktionen.*

Jedem Zufallsknoten ist eine Wahrscheinlichkeitsverteilung auf seinen direkten Nachfolgern zugeordnet.

Die Gewinnfunktion H_i von Spieler i ordnet jedem Blatt des Spielbaumes eine reelle Zahl zu.

Die Menge $\mathfrak{I}$ der Informationsmengen ist eine Partition der Menge der Spieler-i-Knoten von G (für beliebige i, d. h. alle Knoten außer Blättern und Zufallsknoten), die die folgenden Eigenschaften erfüllt:

- Alle Knoten $v \in I$ in einer Informationsmenge $I \in \mathfrak{I}$ haben die gleiche Menge an in v möglichen Aktionen.
- Alle Knoten $v \in I$ sind Spieler-i-Knoten mit dem gleichen i.
- Kein Pfad in G enthält zwei Knoten v_1, v_2, die in der gleichen Informationsmenge $I \in \mathfrak{I}$ liegen.[5]

Eine *reine Strategie s für Spieler i* ist eine Abbildung, die jedem Spieler-i-Knoten v in G eine im Knoten v mögliche Aktion $s(v)$ zuordnet. Dabei muß für je zwei Spieler-i-Knoten v_1, v_2, die in der gleichen Informationsmenge $I \in \mathfrak{I}$ liegen, $s(v_1) = s(v_2)$ gelten.

Es sei s_i für $i = 1, \ldots, n$ eine reine Strategie für Spieler i. Der *Ausgang* V

(Fortsetzung nächste Seite)

(Fortsetzung)

ist eine Zufallsvariable, die als Werte Blätter von G annimmt. Dabei ist $P(V = v)$ wie folgt definiert: Es sei p der (eindeutige) Pfad von der Wurzel von G zu v. Gilt für einen Spieler-i-Knoten v auf p, daß $s_i(v)$ nicht die Aktion auf der v in p folgenden Kante ist, so ist $P(V = v) := 0$. Ansonsten ist $P(V = v) := \prod_\mu p_\mu(\mu')$, wobei μ über alle Zufallsknoten auf p geht, μ' der Nachfolger von μ auf p ist, und p_μ die dem Zufallsknoten μ zugeordnete Verteilung auf seinen Nachfolgern ist.[6]

Die Gewinnfunktion H_i (auf Tupeln von reinen Strategien)[7] für Spieler i ist dann als $H_i(s_1, \ldots, s_n) := EH_i(V)$ definiert (E bezeichne den Erwartungswert).

Die *Normalform von G* ist durch die reinen Strategien und die Gewinnfunktionen H_i auf Tupeln von reinen Strategien gegeben.

7.1.3. Spiele mit perfekter Erinnerung

Die Informationsmengen erlauben es, das Wissen der Spieler in einzelnen Spielsituationen zu modellieren. Dem Leser mag aufgefallen sein, daß diese Konstruktion hinreichend allgemein ist, um auch relativ ungewöhnliche Wissenssituationen zu spezifizieren. So ist es z. B. möglich, daß Spieler 1 zwei aufeinanderfolgende Züge macht, und dabei den *zweiten* Zug in Unkenntnis des ersten tätigt (vgl. Abbildung 7.4). Es ist also möglich, Spieler zu modellieren, die sich an ihre eigenen Beobachtungen und Handlungen nicht erinnern können. Dies mag auf den ersten Blick wie unnötige Allgemeinheit erscheinen, doch macht es – neben dem Fall, daß wir Imperfektionen in den Spielern modellieren wollen – z. B. bei Spielen wie Bridge Sinn, bei denen zwei der Spieler ein gemeinsames Ziel verfolgen. In diesem Fall macht es Sinn, die beiden menschlichen Spieler als einen Spieler im

[5] Diese Einschränkung ist notwendig, da sonst unerwünschte Effekte auftreten können: Da eine reine Strategie auf allen Knoten in der gleichen Informationsmenge die gleiche Entscheidung treffen muß, würde selbst bei einer gemischten Strategie (also wenn wir dem Spieler die Möglichkeit geben, zufällige Entscheidungen zu treffen) in einem Spieldurchgang bei verschiedenen Knoten der gleichen Informationsmenge zwingend die gleiche Entscheidung getroffen. Aber dies widerspricht der Tatsache, daß man, wenn man seine Aktionen in jedem Zug neu zufällig wählt, durchaus in ununterscheidbaren Situationen verschiedene Aktionen durchführen kann.

[6] Diese Definition ist natürlich nur für Spielbäume mit abzählbar vielen Pfaden sinnvoll. Da wir uns aber nur mit endlichen Spielen beschäftigen werden, begnügen wir uns mit dieser Darstellung.

[7] Der Begriff der Gewinnfunktion ist mehrfach überladen. H_i kann auf Blätter von G angewandt werden, auf Tupel reiner Strategien, und auf Tupel gemischter Strategien.

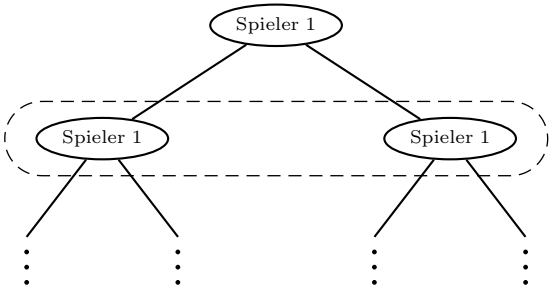


Abbildung 7.4.: Extensivform eines Spiels ohne perfekte Erinnerung.
Hier wählt Spieler 1 zu Beginn ein der beiden von der Wurzel ausgehenden Kanten. Danach darf Spieler 1 nochmals eine Entscheidung treffen, aber da die Knoten in der zweiten Ebene in einer Informationsmenge liegen, darf Spieler 1 diese Entscheidung nicht in Abhängigkeit vom Knoten, also von seiner ersten Entscheidung treffen. In anderen Worten, Spieler 1 vergißt seinen ersten Zug. Solche Konstruktionen sind bei Spielen mit perfekter Erinnerung ausgeschlossen.

Sinne der Spieltheorie aufzufassen. Dann aber darf sich dieser kombinierte Spieler nicht immer an alle seine vorangegangenen Züge erinnern, da manche von diesen von einem anderen als dem gerade am Zug seienden menschlichen Spieler getätigt wurden.

In den meisten Fällen jedoch möchte man sich auf Spiele einschränken, bei denen jeder Spieler sich (i) an alle Informationen erinnert, die ihm je zur Verfügung standen, sowie (ii) an alle seinen bisherigen Aktionen. Spiele mit dieser Eigenschaft nennt man Spiele mit *perfekter Erinnerung* [Kuh56].[8] Formal definieren wir diese wie folgt:[9]

Definition 7.4 (Perfekte Erinnerung)

(Fortsetzung nächste Seite)

[8] Nicht zu verwechseln mit Spielen mit *perfekter Information*, bei denen jede Informationsmenge ein Singleton ist, bei denen man also nicht nur seine eigene Vergangenheit, sondern sämtliche das Spiel betreffenden Informationen kennt.

[9] Diese Definition ist nicht äquivalent zur Originaldefinition aus [Kuh56]. Es gibt Spiele, die die Definition aus [Kuh56] erfüllen, aber nicht unsere. Es handelt sich dabei um die Spiele, in denen jeder Spieler sich an alles Vergangene erinnert, mit Ausnahme dessen, was er wußte, als er sich in Knoten befand, die nur eine ausgehende Kante haben (in denen er also keine Entscheidung treffen konnte). Unsere Definition impliziert die von [Kuh56], daher übertragen sich die Eigenschaften von Spielen mit perfekter Erinnerung nach [Kuh56] auf Spiele mit perfekter Erinnerung gemäß unserer Definition. Insbesondere ist der weiter unten vorgestellte Satz 7.6 auch für unsere Definition gültig.

(Fortsetzung)

Ein n-Spieler-Spiel in Extensivform G hat *perfekte Erinnerung*, wenn für jedes $i \in \{1, \ldots, n\}$ und je zwei Spieler-i-Knoten v_1, v_2 in der gleichen Informationsmenge $I \in \mathfrak{I}$ gilt:

Sei p_j der Pfad von der Wurzel von G zum Knoten v_j.

- Die Folge der Spieler-i-Informationsmengen auf p_j ist die gleiche für $j = 1, 2$.
- Die Folge der Aktionen, die den Kanten in p_j, die von Spieler-i-Knoten ausgehen, zugeordnet sind (also die von Spieler i durchgeführten Aktionen) ist die gleiche für $j = 1, 2$.

Spiele mit perfekter Erinnerung haben unter anderem den großen Vorteil, daß es bei Zwei-Spieler-Nullsummenspielen mit perfekter Erinnerung möglich ist, Nash-Equilibria in polynomieller Zeit (in der Größe des Spielbaums) zu bestimmen. Bevor wir das entsprechende Theorem formulieren können, müssen wir noch eine spezielle Klasse von Strategien vorstellen, die *Verhaltensstrategien*. Bisher hatten wir zufällige Entscheidungen durch gemischte Strategien modelliert. Diese sind Wahrscheinlichkeitsverteilungen auf der Menge der reinen Strategien. Da die Anzahl der reinen Strategien exponentiell in der Größe des Spielbaums ist, ist eine gemischte Strategie also ein Vektor, dessen Dimension exponentiell in der Größe des Spielbaums ist. Wir können also allein aufgrund der Größe der Darstellung der Ausgabe nicht erwarten, einen polynomiellen Algorithmus zu finden, der ein Nash-Equilibrium in gemischten Strategien findet. Im Gegensatz zu den gemischten Strategien wird bei einer Verhaltensstragie in jedem Knoten, d. h. in jeder Spielsituation, unabhängig eine Aktion gewählt. Jedem Knoten v ist also eine Wahrscheinlichkeitsverteilung $\beta(v)$ auf den im Knoten v möglichen Aktionen zugeordnet. Bei Erreichen des Knotens wird eine Aktion a entsprechend der Verteilung $\beta(v)$ gewählt und zwar unabhängig von den Aktionen, die in vorangegangenen Stadien des Spiels gewählt wurden. Auf den ersten Blick mag die Tatsache, daß alle Entscheidungen unabhängig getroffen werden müssen, wie eine starke Einschränkung erscheinen. Doch wir erinnern uns, daß zumindest im Falle eines Spiels mit perfekter Erinnerung jeder Knoten die gesamte Information über die Vergangenheit des Spiels enthält, so daß die Wahrscheinlichkeitsverteilung auch von Vergangenem abhängen kann und somit die Entscheidungen effektiv nicht unabhängig sind. In der Tat hat [Kuh56] gezeigt, daß bei Spielen mit perfekter Erinnerung Verhaltensstrategien so mächtig wie gemischte Strategien sind.[10] Verhaltensstrategien haben nun den Vorteil, daß ihre

[10] In dem Sinne, daß für jede gemischte Spieler-i-Strategie μ_i eine Verhaltensstrategie β_i existiert, so daß für alle gemischten Strategien μ_j der anderen Spieler die Gewinnfunktion unabhängig davon ist, ob Spieler 1 die gemischte Strategie μ_i oder die Verhaltensstrategie

Darstellung wesentlich kompakter ist, sie können durch Vektoren polynomieller Länge dargestellt werden. Sind zusätzlich die einzelnen Wahrscheinlichkeiten durch rationale Zahlen polynomieller Länge darstellbar (wie dies in Satz 7.6 der Fall sein wird), so läßt sich eine Verhaltensstrategie effizient darstellen.

Wir geben nun die Definition einer Verhaltensstrategie:

Definition 7.5 (Verhaltensstrategie)
Es sei G ein n-Spieler-Spiel in Extensivform. Eine *Verhaltensstrategie β für Spieler i* für G ist eine Abbildung, die jedem Spieler-i-Knoten v von G eine Wahrscheinlichkeitsverteilung $\beta(v)$ auf den in v möglichen Aktionen zuordnet. Dabei muß für zwei Knoten v_1, v_2, die in der gleichen Informationsmenge I von G liegen, $\beta(v_1) = \beta(v_2)$ sein. Die Menge der Verhaltensstrategien für Spieler i bezeichnen wir mit $\mathfrak{B}_i$.

Einer Verhaltensstrategie β für Spieler i ordnen wir in natürlicher Weise eine gemischte Strategie μ zu: Es sei s eine reine Strategie für Spieler i. Dann ist die Wahrscheinlichkeit $\mu(s)$, die μ der Strategie s zuordnet, gegeben durch

$$\mu(s) := \prod_{I \in \mathfrak{I}_i} \beta(I)(s(I)).$$

Dabei ist $\mathfrak{I}_i \subseteq \mathfrak{I}$ die Menge der Informationsmengen I, die einen Spieler-i-Knoten enthalten. Weiter ist $s(I) := s(v)$ für $v \in I$ (dies hängt per Definition nicht vom Repräsentanten v ab) und $\beta(I)$ analog. Es bezeichne $\beta(I)(a)$ die Wahrscheinlichkeit, die $\beta(I)$ der Aktion a zuordnet.

Diese Zuordnung erlaubt es uns, eine Verhaltensstrategie als Spezialfall einer gemischten aufzufassen, liefert also eine Einbettung $\mathfrak{B}_i \subseteq \mathfrak{M}_i$. Damit übertragen sich der Begriff der Gewinnfunktion und des Nash-Equilibriums auf Verhaltensstrategien.

Bei der algorithmischen Behandlung von Spielen in Extensivform nehmen wir im folgenden immer an, daß ein Spiel durch die explizite Beschreibung seines Spielbaums und der Informationsmengen gegeben ist. Die den von Zufallsknoten ausgehenden Kanten zugeordneten Wahrscheinlichkeiten sind rationale Zahlen, die durch Zähler und Nenner gegeben sind. Die den von Spieler-Knoten ausgehenden Kanten zugeordneten Aktionen sind durch Strings gegeben. Die Gewinnfunktion ist durch Angabe eines Tupels von rationalen Zahlen für jedes Blatt des Baums gegeben. Insbesondere können wir keine Spiele betrachten, die irrationale Wahrscheinlichkeiten oder Gewinnfunktionen haben, oder die unendlich sind.

β_i verwendet.

Eine Spieler-i-Verhaltensstrategie für ein Spiel G ist gegeben durch explizite Angabe der Wahrscheinlichkeitsverteilungen über den in den Knoten möglichen Aktionen. Die Wahrscheinlichkeitsverteilungen wiederum sind durch explizite Angaben der Wahrscheinlichkeiten als Paar von Zähler und Nenner gegeben. Wieder können wir nur Verhaltensstrategien mit rationalen Wahrscheinlichkeiten darstellen.

Wir können nun die oben angedeutete Tatsache, daß das Nash-Equilibrium (in gemischten Strategien) bei Spielen mit perfekter Erinnerung effizient zu finden ist, formal wiedergeben:

Satz 7.6 (Komplexität des Nash-Equilibriums [KM92])
Es existiert ein deterministischer, in der Länge der Eingabe polynomiell-beschränkter Algorithmus N mit den folgenden Eigenschaften:

- Bei Eingabe eines Zwei-Spieler-Nullsummenspiels G in Extensivform (mit rationalen Wahrscheinlichkeiten und Gewinnfunktionen) gibt N ein Paar von Verhaltensstrategien (β_1, β_2) (mit rationalen Wahrscheinlichkeiten) aus.
- Als gemischte Strategien aufgefaßt bilden (β_1, β_2) ein Nash-Equilibrium von G.

Beweis: Der Beweis findet sich in [KM92]. In deren Formulierung des Satzes wird jedoch lediglich garantiert, daß das Nash-Equilibrium in polynomieller Zeit gefunden werden kann, es wird nicht spezifiziert, ob dies mit einem deterministischen Algorithmus möglich ist. Untersucht man jedoch den Beweis, so wird das Problem (deterministisch) auf ein lineares Programmierungsproblem reduziert, welches nach [GLS88] effizient lösbar ist. In [GLS88] sind verschiedene Algorithmen für dieses Problem gegeben. Verwendet man den Algorithmus aus dem dortigen Theorem 6.4.9, so erhält man den hiesigen Satz 7.6 in der oben angegebenen Form. □

7.2. Sicherheit als Spiel

In diesem Abschnitt werden wir sehen, wie das reale und das ideale Protokoll als Spiel im Sinne der Spieltheorie aufgefaßt werden können. Der Grundansatz ist der natürliche: Das Protokoll unter Betrachtung legt die Regeln des Spiels (den Spielbaum) fest, und die quantifizierten Maschinen stellen die Spieler dar, ihre Aktionen sind die Nachrichten, die sie senden, und ihr Wissen (welches wiederum durch die Informationsmengen modelliert ist) besteht aus ihrer Sicht, d. h. aus allen von ihnen bislang gesandten und empfangenen Nachrichten. Das

Spiel endet, wenn die Umgebung eine Ausgabe tätigt. Bei diesem Ansatz treten die folgenden Probleme auf:

- Wir können je ein Spiel für das reale und das ideale Protokoll entwerfen, und die Umgebung an beiden Spielen teilnehmen lassen. Dies spiegelt jedoch nicht gut die Situation in der Definition der Sicherheit (weder der allgemeinen noch der speziellen) wieder: Dort erhoffen wir uns von der Umgebung, daß sie das reale und das ideale Protokoll möglichst gut unterscheidet. Die Spieltheorie (genauer: das Konzept des Nash-Equilibriums) hingegen gibt uns Mittel an die Hand, um die Strategie der Umgebung dahingehend zu optimieren, in *einem* der Spiele (real oder ideal) ein konkretes (durch die Gewinnfunktion modelliertes) Ziel möglichst gut zu erreichen. Um den Begriff der Sicherheit spieltheoretisch betrachten zu können, müssen wir daher beide Protokolle (reales und ideales) in *einem* Spiel wiedergeben. Dies tun wir wie folgt: Aus dem realen und dem idealen Spiel (dem realen und dem idealen Protokoll entsprechend) konstruieren wir ein neues Spiel, das *kombinierte Spiel*, in welchem zunächst mit Wahrscheinlichkeit $\frac{1}{2}$ gewählt wird, ob das ideale oder das reale Spiel gespielt wird. Die Umgebung wird von dieser Wahl nicht informiert. Zum Schluß des Spiels rät die Umgebung, welches der Spiele gespielt wurde, und der Gewinn der Umgebung ist 1, wenn sie richtig rät und -1 sonst.

- In unserer Modellierung treten drei verschiedene Spieler auf: Umgebung, Angreifer und Simulator. Dabei arbeiten die Umgebung und der Angreifer, da beide allquantifiziert sind, in einem gewissen Sinne zusammen. Es bieten sich nun mehrere Möglichkeiten an, dies zu modellieren. (i) Wir modellieren ein Drei-Spieler-Spiel, und ordnen Angreifer und Umgebung die gleiche Gewinnfunktion zu. (ii) Wir modellieren Angreifer und Umgebung als *ein* Spieler, und erzwingen durch eine entsprechende Wahl der Informationsmengen, daß dieser Spieler effektiv aus zwei getrennten Agenten besteht (so wie z. B. bei Bridge die gemeinsam spielenden Parteien als ein Spieler aufgefaßt werden können, vgl. die Diskussion am Anfang von Abschnitt 7.1.3). In beiden Fällen ist jedoch das resultierende Spiel kein Zwei-Spieler-Nullsummenspiel mit perfekter Erinnerung, und ein solches brauchen wir, um Satz 7.6 anwenden zu können. Wir lösen dieses Problem, indem wir den bereits mehrfach verwendeten Dummy-Angreifer nutzen. Dieser leitet die Nachrichten zwischen Umgebung und Protokoll einfach durch. Da wir uns o. B. d. A. auf den Dummy-Angreifer beschränken können und dessen Programm fest ist, ist es möglich, den Angreifer nicht als Spieler, sondern als Teil des Protokolls bzw. des Spiels zu betrachten. Es verbleiben als Spieler Simulator und Umgebung, was es erlaubt, ein Zwei-Spieler-Nullsummenspiel mit perfekter Erinnerung zu modellieren.

- Bislang haben wir noch nicht untersucht, ob das resultierende Spiel überhaupt endlich ist. Ein unendliches Spiel ist für unsere Zwecke ungeeignet, da – ganz abgesehen von den algorithmischen Schwierigkeiten – bei einem solchen die Existenz eines Nash-Equilibriums nicht garantiert ist.[11] Wir werden uns bei unserer Betrachtung auf Protokolle mit beschränkter Kommunikationskomplexität beschränkten, d. h. auf Protokolle, bei denen eine vom Sicherheitsparameter abhängige Schranke existiert, die die Länge und Anzahl aller vom Protokoll verarbeiteten Nachrichten beschränkt. Doch dies allein führt nicht zu einem endlichen Spiel. So besteht noch die Möglichkeit der Kommunikation zwischen Simulator und Umgebung, welche keine obere Schranke kennen muß. Hier jedoch rettet uns die Tatsache, daß wir einen Dummy-Angreifer angenommen haben. Da dieser nur Nachrichten zwischen Protokoll und Umgebung weiterleitet, können wir auch dessen Kommunikationskomplexität o. B. d. A. als beschränkt annehmen. Da der Simulator vom Dummy-Angreifer ununterscheidbar sein will, muß auch er seine Kommunikation zur Umgebung beschränken. Damit ist aber die Kommunikation aller Maschinen beschränkt, und die resultierenden Spiele sind endlich.

Wir definieren zunächst getrennt für das reale und das ideale Modell jeweils ein Spiel. Im *idealen Spiel* ist Spieler 1 die Umgebung und Spieler 2 der Simulator. Diese führen gemeinsam das ideale Protokoll aus, und die Umgebung kann mit einer Ausgabe 0 oder 1 terminieren, was zu einem Gewinn von -1 bzw. 1 (für die Umgebung) führt. Da aber auch der Simulator die Protokollausführung beenden kann (er ist der Scheduler) und damit bewirken, daß die Umgebung nie Ausgabe generiert, müssen wir noch diesen Fall berücksichtigen und ordnen ihm den Gewinn -1 für die Umgebung zu.[12]

Im *realen Spiel* hingegen ist die Umgebung der einzige Spieler. Das Spiel besteht aus dem realen Protokoll und dem Angreifer (d. h. der Angreifer ist nicht wie der Simulator ein Spieler, sondern eine normale Maschine). Die Gewinne sind wie oben definiert.

Definition 7.7 (Reales und ideales Spiel)
Es sei N ein Netzwerk und $\mathcal{Z}, \mathcal{S} \in N$ Maschinen mit den Namen `env` und `adv`. Sei $k \in \mathbb{N}$ und $p \in \mathbb{N}$.

(Fortsetzung nächste Seite)

[11] In der Tat führt das trennende Beispiel aus Abschnitt 5.2 zu einem Spiel ohne Nash-Equilibrium und ist gerade deshalb als trennendes Beispiel geeignet.

[12] Letztendlich wird dieser Fall keine große Rolle spielen, da der Dummy-Angreifer diesen Fall nie eintreten lassen wird, und daher ein geeigneter Simulator o. B. d. A. auch nicht in diese Situation geraten darf.

(Fortsetzung)

Das *ideale Spiel* $G = G^I_{p,k}(N)$ *zu* N ist ein Zwei-Spieler-Nullsummenspiel, das wie folgt konstruiert wird:

Der Spielbaum G entspricht einer Ausführung $run_{N,k}(\texttt{env} \leftarrow \lambda)$ des Netzwerks N bei Sicherheitsparameter k und Auxiliary input λ (vgl. Definition 2.3), wobei die von $\mathcal{Z}$ gesandten Nachrichten m' (und die Ports p', über die diese Nachrichten gesandt werden) von Spieler 1 gewählt werden (d. h. diese Nachrichten und Ports sind die in den entsprechenden Knoten möglichen Aktionen), und analog die Nachrichten von $\mathcal{S}$ von Spieler 2. Zusätzlich stehen Spieler 1 noch drei spezielle Aktionen 0, 1 und $\perp$ zur Verfügung, welche zu Blättern führen (diese entsprechen den möglichen Ausgaben von $\mathcal{Z}$, bzw. einem Abbruch durch $\mathcal{Z}$). Wir nennen diese Blätter *Blätter vom Typ* 0, 1 *bzw.* $\perp$. Spieler 2 hat die zusätzliche Aktion $\perp$ zur Verfügung, welche zu Blättern vom Typ $\perp$ führt.

Folgende Einschränkungen gelten zusätzlich: Die Nachrichten, die Spieler 1 und Spieler 2 wählen, dürfen höchstens die Länge p haben. In der p-ten Aktivierung über den gleichen Port hat Spieler 1 nur die Aktionen 0, 1 und $\perp$ zur Verfügung ($\mathcal{Z}$ muß also spätestens nach p solchen Aktivierungen eine Ausgabe liefern). In der p-ten Aktivierung über den gleichen Port hat Spieler 2 nur die Aktion $\perp$ zur Verfügung.

Die Gewinnfunktion von G ist die folgende: Für ein Blatt b vom Typ 1 ist $H_1(b) = 1$ (der Gewinn von Spieler 1), für ein Blatt b vom Typ 0 oder vom Typ $\perp$ ist $H_1(b) = -1$. Da das Spiel ein Nullsummenspiel ist, gilt $H_2 := -H_1$.

Die Informationsmengen von G sind wie folgt definiert: Für einen Spieler-1-Knoten v sei $Z(v)$ die Sicht von $\mathcal{Z}$ bis zu diesem Knoten (d. h. die Liste aller bis dahin von $\mathcal{Z}$ erhaltenen und gesandten Nachrichten). Analog sei $S(v)$ für einen Spieler-2-Knoten v die Sicht von $\mathcal{S}$ bis dorthin. Die Informationsmengen von G sind die Mengen der Form

$$\{v \in G : v \text{ ist Spieler-1-Knoten und } Z(v) = x\} \quad \text{und}$$
$$\{v \in G : v \text{ ist Spieler-2-Knoten und } S(v) = x\}$$

für beliebige x, d. h. zwei Knoten sind genau dann in der gleichen Informationsmenge, wenn die Sicht des Spielers am Zug in beiden Knoten die gleiche ist.

(Fortsetzung nächste Seite)

(Fortsetzung)

Das *reale Spiel* $G^R_{p,k}(N)$ ist analog definiert, mit dem Unterschied, daß nur der Maschine $\mathcal{Z}$ Spieler-Knoten zugeordnet werden. Es handelt sich also um ein *Ein*-Spieler-Spiel.

Es mag auf den ersten Blick merkwürdig erscheinen, daß die Definitionen des realen und des idealen Spiels ein Netzwerk N voraussetzen, welches bereits eine bestimmte Umgebung und einen bestimmten Simulator enthalten. Sollen nicht die Rollen dieser Maschinen von den Spielern übernommen werden? Wieso hängt dann das Spiel bereits von diesen Maschinen ab? Eine genaue Betrachtung der Definition offenbart jedoch, daß das Spiel nur von der äußeren Form (d. h. der Menge ihrer Ports) der Umgebung bzw. des Simulators abhängt, nicht aber von deren Verhalten. Durch die Wahl der Umgebung und des Simulators im Netzwerk N legen wir also lediglich fest, über welche Ports die Spieler Nachrichten senden dürfen. Welche *Nachrichten* gesandt werden, bestimmen die Spieler. Außerdem ist die Menge der Ports, die überhaupt von Interesse sind, beschränkt. So hat die Umgebung oder der Simulator keinen Vorteil, wenn er Ports hat, zu denen es kein Gegenstück gibt. Wir sagen, eine Umgebung oder ein Simulator hat *die natürlichen Ports*, wenn er alle Ports hat, zu denen es ein Gegenstück gibt (zur genauen Definition siehe weiter unten). Da das reale und ideale Spiel nur von den Ports von Angreifer und Umgebung abhängen, genügt es zu spezifizieren, daß Umgebung und Simulator die natürlichen Ports haben, um das Spiel eindeutig festzulegen.

Definition 7.8 (Natürliche Ports)
Es seien π und ρ Protokolle und $\mathcal{A}$ ein zulässiger Angreifer.

Wir sagen, eine Umgebung $\mathcal{Z}$ *habe die natürlichen Ports (zu π, ρ und $\mathcal{A}$)*, wenn zu jedem ausgehenden Protokollport von π oder ρ ein gleichnamiger eingehender Protokollport von $\mathcal{Z}$ existiert, zu jedem eingehenden Protokollport von π oder ρ ein gleichnamiger ausgehender Protokollport von $\mathcal{Z}$ existiert, zu jedem ausgehenden Umgebungsport von $\mathcal{A}$ ein gleichnamiger eingehender Umgebungsport von $\mathcal{Z}$ existiert, und zu jedem eingehenden Umgebungsport von $\mathcal{A}$ ein gleichnamiger ausgehender Umgebungsport von $\mathcal{Z}$ existiert, sonst $\mathcal{Z}$ aber keine weiteren Ports hat.

Wir sagen, ein Simulator $\mathcal{S}$ *hat die natürlichen Ports (zu π, ρ und $\mathcal{A}$)*, wenn er die gleichen Umgebungsports wie $\mathcal{A}$ hat, zu jedem ausgehenden Angreiferport von ρ ein gleichnamiger eingehender Angreiferport von $\mathcal{S}$

(Fortsetzung nächste Seite)

(Fortsetzung)

existiert, zu jedem eingehenden Angreiferport von ρ ein gleichnamiger ausgehender Angreiferport von $\mathcal{S}$ existiert, sonst $\mathcal{S}$ aber keine weiteren Ports hat.

Durch das reale oder ideale Spiel G entsteht ein direkter Zusammenhang zwischen Strategien von G und Umgebungen und Simulatoren. Gegeben eine gemischte Spieler-1-Strategie μ_1 können wir eine Umgebung $\mathcal{Z}$ definieren, die ihre Nachrichten gemäß dieser Strategie wählt (für einen bestimmten Sicherheitsparameter k und Auxiliary input z), wir sagen $\mathcal{Z}$ *implementiere die Strategie* μ_1. Ebenso können wir einen Simulator $\mathcal{S}$ konstruieren, der eine Spieler-2-Strategie μ_2 implementiert. Dann ist die Wahrscheinlichkeit für einen Ausgabe 1, 0 oder $\perp$ bei einem Protokollauf mit $\mathcal{Z}$ und $\mathcal{S}$ (bzw. nur $\mathcal{Z}$ im Falle des realen Spiels) gerade die Wahrscheinlichkeit, daß eine Ausführung des Spiels in einem Blatt vom Typ 1, 0 oder $\perp$ endet, wenn Spieler 1 und 2 die Strategien μ_1 und μ_2 verfolgen. Umgekehrt gibt es auch zu jeder Umgebung $\mathcal{Z}$ (mit Kommunikationskomplexität p) für jeden Sicherheitsparameter k und jeden Auxiliary input z eine Strategie μ_1, die von dieser Umgebung $\mathcal{Z}$ implementiert wird: μ_1 wählt einfach die Nachrichten so, wie $\mathcal{Z}$ sie in der gleichen Situation gewählt hätte. Analoges gilt für $\mathcal{S}$.

Bislang haben wir zwei *getrennte* Spiele für das reale und das ideale Protokoll formuliert. Dies erlaubt es uns zwar, einen Zusammenhang zwischen den Strategien der Spiele und den Umgebungen und Simulatoren der Protokolle zu formulieren. Dieser Zusammenhang ist aber insofern nicht natürlich, daß eine „gute“ Umgebung und ein „guter“ Simulator (d. h. eine Umgebung, die möglichst gut unterscheidet, und ein Simulator, der dies möglichst gut verhindert) nicht notwendig gute Strategien für die Spiele darstellen und umgekehrt. Dies liegt daran, daß der Begriff des Unterscheidens keine Eigenschaft ist, die an einem der Protokolle festgemacht werden kann, sondern eben eine Aussage darüber ist, wie sich die Verteilung der Ausgabe im realen und im idealen Protokoll unterscheiden. Daher müssen wir ein Spiel konstruieren, welches sowohl das reale als auch das ideale Spiel umfaßt, und in welchem die Umgebung erraten muß, welches Spiel vorliegt. Die Erfolgswahrscheinlichkeit beim Raten steht dann in direktem Zusammenhang zum Gewinn der Umgebung in diesem *kombinierten Spiel* (vgl. auch Lemma 7.10).

Das kombinierte Spiel sieht wie folgt aus: Zunächst wird zufällig gewählt, ob das reale oder das ideale Spiel gespielt wird. Spieler 1 (die Umgebung) wird nicht darüber informiert, welches Spiel gewählt wurde. Am Ende des Spiels gibt die Umgebung ein Bit aus, wobei 1 bedeute, daß sie vermutet, das reale Spiel

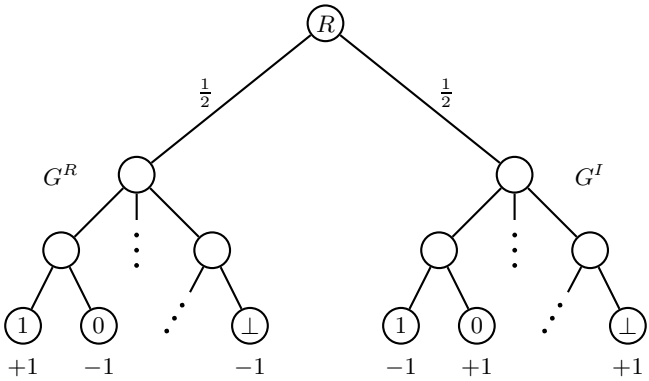


Abbildung 7.5.: Das kombinierte Spiel besteht aus der Wurzel R und darunter dem realen Spiel G^R und dem idealen Spiel G^I. Die Wurzel ist ein Zufallsknoten, und mit gleicher Wahrscheinlichkeit wird zur Wurzel von G^R oder G^I übergegangen. Das Spiel G^R bzw. G^I wird dann normal gespielt, lediglich die Gewinne sind gegenüber den ursprünglichen Spielen verändert: Die Gewinne der Blätter von G^R sind wie in G^R, die von G^I sind negiert (exemplarisch sind die Gewinne von Spieler 1 für die Blätter der Typen 1, 0 und $\perp$ eingezeichnet). Nicht dargestellt ist hier die Tatsache, daß die Informationsmengen von Spieler 1 Knoten aus beiden Spielen umfassen können, daß Spieler 1 also nicht erfährt, welches der beiden Spiele tatsächlich gespielt wird.

zu spielen, und 0, daß sie vermutet, daß ideale zu spielen. Der Gewinn der Umgebung ist nun 1, wenn sie richtig rät, und -1, wenn sie falsch rät. Nun ist es tatsächlich so, daß die Umgebung höheren Gewinn erzielen kann, wenn sie richtig rät, und der Simulator höheren Gewinn erzielt, wenn er dies verhindert. Ein Unterschied zum Begriff der Ununterscheidbarkeit besteht natürlich weiterhin: Wenn die Umgebung konsistent falsch rät, d. h. genau dann 0 ausgibt, wenn das reale Spiel vorliegt, so ist ihr Gewinn negativ, obwohl sie sehr gut unterscheidet. Wir werden aber sehen, daß dies kein Problem darstellt, da solche „falsch ratenden“ Umgebungen in „richtig ratende“ umgewandelt werden können, indem sie das ausgegebene Bit negieren.

Ein weiteres Detail gilt es zu beachten: Was passiert, wenn $\mathcal{Z}$ keine Ausgabe liefert? Wir behandeln diesen Fall, indem im realen Spiel die Umgebung davon einen Nachteil hat (einen Gewinn -1), im idealen der Simulator. Der Grund liegt darin, daß wir o. B. d. A. die Umgebung so wählen können, daß sie nicht ohne Ausgabe terminiert, und den Simulator auf diese Art davon abhalten, ohne Ausgabe zu terminieren, da es zu seinem Nachteil ist. Genaueres findet sich im Beweis von Lemma 7.10.

Die genaue Definition des kombinierten Spiels ist also die folgende:

Definition 7.9 (Das kombinierte Spiel)
Es sei N_R ein Netzwerk und $\mathcal{Z}, \mathcal{S} \in N_R$ Maschinen mit den Namen `env` und `adv`. Weiter sei N_I ein Netzwerk mit $\mathcal{Z} \in N_I$. Sei $k \in \mathbb{N}$ und $p \in \mathbb{N}$.

Es sei $G^I := G^I_{p,k}(N_I)$ das ideale Spiel zu N_I und $G^R := G^R_{p,k}(N_R)$ das reale Spiel zu N_R (vgl. Definition 7.7).

Das *kombinierte Spiel* $G = G^C_{p,k}(N_R, N_I)$ *zu* N_R *und* N_I ist dann das folgende Zwei-Spieler-Nullsummenspiel:

Der Spielbaum von G besteht aus der Vereinigung der Spielbäume von G^I und G^R zusammen mit einem neuen Zufallsknoten R als Wurzel. R hat als direkte Nachfolger die Wurzeln von G^I und G^R. Jedem der Nachfolger ist die Wahrscheinlichkeit $\frac{1}{2}$ zugeordnet (d. h. G geht mit gleicher Wahrscheinlichkeit in G^I oder G^R über, vgl. Abbildung 7.5).

Die Gewinnfunktion H_1 ist wie folgt definiert: Für ein Blatt v aus G^R ist der Gewinn wie in G^R, d. h. wenn v vom Typ 1 ist, dann ist $H_1(v) = +1$, und wenn v vom Typ 0 oder vom Typ $\perp$ ist, so ist $H_1(v) = -1$. Für ein Blatt v aus G^I ist der Gewinn das Negative des Gewinns in G^I, d. h. wenn v vom Typ 1 ist, dann ist $H_1(v) = -1$, und wenn v vom Typ 0 oder vom Typ $\perp$ ist, so ist $H_1(v) = +1$. Die Gewinnfunktion von Spieler 2 ist $H_2 = -H_1$, da G ein Nullsummenspiel ist.

Die Informationsmengen von G sind wie folgt definiert: Für einen Knoten v seien $Z(v)$ und $S(v)$ wie in Definition 7.7. Die Informationsmengen von G sind dann die Mengen der Form

$$\{v \in G : v \text{ ist Spieler-1-Knoten und } Z(v) = x\} \quad \text{und}$$
$$\{v \in G : v \text{ ist Spieler-2-Knoten und } S(v) = x\}$$

Insbesondere können also Spieler-1-Knoten aus G_I und G_R in der gleichen Informationsmenge liegen, wenn die Sicht $Z(v)$ von $\mathcal{Z}$ gleich ist, d. h. $\mathcal{Z}$ erfährt nicht, ob G_I oder G_R gespielt wird.

Wie oben schon angedeutet, liegt der Zusammenhang zwischen dem kombinierten Spiel und dem Sicherheitsmodell darin, daß eine Umgebung, die hohe Gewinne erzielt, gut unterscheidet, und daß umgekehrt eine gut unterscheidende Umgebung leicht in eine umgewandelt werden kann, die hohe Gewinne erzielt. Dieser Sachverhalt wird von dem folgenden Lemma dargestellt:

Lemma 7.10 (Eigenschaften des kombinierten Spiels)

(Fortsetzung nächste Seite)

(Fortsetzung)

Es seien π und ρ Protokolle, $\mathcal{A}$ ein zulässiger Angreifer, $\mathcal{Z}_0$ und $\mathcal{S}_0$ eine Umgebung und ein Simulator mit den natürlichen Ports. Dann seien $N_R := \pi \cup \{\mathcal{Z}_0, \mathcal{A}\}$ und $N_R := \rho \cup \{\mathcal{Z}_0, \mathcal{S}_0\}$. Weiter seien $k, p \in \mathbb{N}$, $z \in \Sigma^*$ und μ_1, μ_2 Spieler-1- und -2-Strategien für das kombinierte Spiel $G := G^C_{p,k}(N_R, N_I)$. Es bezeichne H_1 die Spieler-1-Gewinnfunktion von G.

Es sei $\mathcal{Z}$ eine zulässige Umgebung mit den natürlichen Ports, Ein-Bit-Ausgabe und Kommunikationskomplexität p, die μ_1 bei Sicherheitsparameter k und Auxiliary input z implementiert. Weiter sei $\mathcal{S}$ ein zulässiger Simulator mit den natürlichen Ports und Kommunikationskomplexität p, der μ_2 bei Sicherheitsparameter k implementiert.

Dann ist

$$\Delta\big(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)\big) \geq H_1(\mu_1,\mu_2).$$

Wenn zusätzlich $P(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) = \bot) = 0$, dann gibt es eine (von k, z unabhängige) zulässige Umgebung $\mathcal{Z}'$ (ebenfalls mit den natürlichen Ports, Ein-Bit-Ausgabe und Kommunikationskomplexität p), die eine Strategie μ_1' implementiert, so daß

$$\Delta\big(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)\big) = H_1(\mu_1',\mu_2).$$

Man beachte: Auf der rechten Seite der Gleichung kommt die Umgebung $\mathcal{Z}$, nicht die Umgebung $\mathcal{Z}'$ vor.

Beweis: Für $x \in \{0, 1, \bot\}$ sei abkürzend $P_x^R := P(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) = x)$ und $P_x^I := P(\mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z) = x)$, sowie $\Delta_x := P_x^R - P_x^I$. Außerdem stehe Δ für $\Delta\big(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)\big)$.

Zunächst wollen wir $H_1(\mu_1, \mu_2)$ berechnen. Nach Definition des kombinierten Spiels ist

$$H_1(\mu_1,\mu_2) = \frac{H_1^R(\mu_1) - H_1^I(\mu_1,\mu_2)}{2}$$

wobei H_1^R die Gewinnfunktion von Spieler 1 des realen Spiels $G^R = G^R_{p,k}(N_R)$ ist und H_1^I die des idealen Spiels $G^I = G^I_{p,k}(N_I)$.

Nach Konstruktion von G^R und der Tatsache, daß $\mathcal{Z}$ die Strategie μ_1 implementiert, ist P_x^R die Wahrscheinlichkeit, daß das Spiel G^R (wenn Spieler 1 die Strategie μ_1 verwendet) in einem Blatt vom Typ x endet. Da Blätter vom Typ 1 einen Gewinn 1, sowie Blätter vom Typ 0 und $\bot$ einen Gewinn -1 nach sich ziehen, ist damit $H_1^R(\mu_1) = P_1^R - P_0^R - P_\bot^R$.

Analog erhalten wir $H_1^I(\mu_1,\mu_2) = P_1^I - P_0^I - P_\perp^I$. Zusammen ergibt sich

$$H_1(\mu_1,\mu_2) = \tfrac{1}{2}(P_1^R - P_0^R - P_\perp^R - P_1^I + P_0^I + P_\perp^I) = \tfrac{1}{2}(\Delta_1 - \Delta_0 - \Delta_\perp).$$

Dann ist (Lemma 1.3 (iii))

$$\Delta = \tfrac{1}{2}\big(|\Delta_1| + |\Delta_0| + |\Delta_\perp|\big) \geq \tfrac{1}{2}\big(\Delta_1 - \Delta_2 - \Delta_\perp\big) = H_1(\mu_1,\mu_2),$$

womit die Ungleichung aus dem Lemma bewiesen ist.

Wir wollen nun die Umgebung $\mathcal{Z}'$ konstruieren und die Gleichung aus dem Lemma zeigen.

Da nach Voraussetzung $P_\perp^R = 0$, ist $\Delta_\perp \leq 0$. Weiter ist ist $\Delta_1 + \Delta_0 + \Delta_\perp = 0$, d. h. mindestens einer der folgenden Fälle tritt ein:

$$\begin{aligned} ++ &: \quad \Delta_1 \geq 0, \Delta_0 \geq 0 \\ +- &: \quad \Delta_1 \geq 0, \Delta_0 \leq 0 \\ -+ &: \quad \Delta_1 \leq 0, \Delta_0 \geq 0 \end{aligned}$$

Es ist dann (unter Benutzung von $\Delta_1 + \Delta_0 + \Delta_\perp = 0$ und $\Delta = \frac{1}{2}\big(|\Delta_1| + |\Delta_0| + |\Delta_\perp|\big)$):

$$\Delta = \begin{cases} \Delta_1 + \Delta_0 = -\Delta_\perp & (++) \\ \Delta_1 = -\Delta_0 - \Delta_\perp & (+-) \\ \Delta_0 = -\Delta_1 - \Delta_\perp & (-+) \end{cases}$$

und wir definieren $T = T_{k,z}$ durch

$$(T(1),T(0)) := \begin{cases} (1,1) & (++) \\ (1,0) & (+-) \\ (0,1) & (-+) \end{cases}$$

Es sei $\mathcal{Z}'$ die Maschine, die sich wie $\mathcal{Z}$ verhält, aber die, wenn $\mathcal{Z}$ den Wert x ausgeben würde, $T_{k,z}(x)$ ausgibt.

Wir definieren $P_x^{R\prime}$, $P_x^{I\prime}$, Δ_x' und Δ' analog zu den Variablen ohne Strich, aber auf $\mathcal{Z}'$ statt $\mathcal{Z}$ bezogen. Es ist für $x \in \{0,1\}$:

$$P_x^{R\prime} = \delta_{T(1)=x} P_1^R + \delta_{T(0)=x} P_0^R \quad \text{und} \quad P_x^{I\prime} = \delta_{T(1)=x} P_1^I + \delta_{T(0)=x} P_0^I,$$

wobei $\delta_{a=b} := 1$ genau dann, wenn $a = b$ und $\delta_{a=b} := 0$ sonst. Also

$$\Delta_x' = \delta_{T(1)=x}\Delta_1 + \delta_{T(0)=x}\Delta_0.$$

In den drei Fällen ergibt sich:

$$(\Delta_1', \Delta_0') = \begin{cases} (\Delta_1 + \Delta_0, 0) & (++) \\ (\Delta_1, \Delta_0) & (+-) \\ (\Delta_0, \Delta_1) & (-+) \end{cases}$$

und $\Delta_\perp = \Delta_\perp'$. Also gilt in allen Fällen: $2\Delta = \Delta_1' - \Delta_0' - \Delta_\perp'$.

Es sei μ_1' die von $\mathcal{Z}'$ implementierte Strategie. Wie oben erhalten wir

$$H_1(\mu_1', \mu_2) = \tfrac{1}{2}(\Delta_1' - \Delta_0' - \Delta_\perp') = \Delta.$$

Damit ist die Gleichung aus dem Lemma gezeigt. □

Ein Nachteil des Lemmas 7.10 ist, daß wir für die untere Gleichung die Bedingung haben, daß in einer Ausführung des realen Protokolls die Umgebung immer Ausgabe gibt. Dies mag im allgemeinen nicht allein von der Umgebung abhängen, so kann eine andere Maschine in eine Endlosschleife gehen, oder der Angreifer terminieren, und damit (da er der Scheduler ist) das Protokoll zum Halten bringen. Wenn aber alle Maschinen beschränkte Laufzeit haben (und nur diesen Fall werden wir betrachten, da sonst unendliche Spiele auftreten), und der Angreifer der Dummy-Angreifer ist und somit nicht terminiert, ohne die Umgebung zunächst „zu warnen“, kann in diesem Fall die Umgebung so gewählt werden, daß sie immer Ausgabe liefert (wir nutzen dies im Beweis von Lemma 7.14). Eine Folge dieser Tatsache ist, daß wir in Abschnitt 7.3 nur in dem Falle universelle Umgebungen und Simulatoren konstruieren, wenn der Angreifer der Dummy-Angreifer ist.

Will man größere Allgemeinheit, sprich die Bedingung fallen lassen, daß die Umgebung immer Ausgabe liefert, so hat man das Problem, daß die Wahrscheinlichkeiten, daß im realen und idealen keine Ausgabe geliefert wird, auch einen Beitrag zum statistischen Abstand liefern. Die Differenz zwischen diesen Wahrscheinlichkeiten kann jedoch positiv wie auch negativ in den statistischen Abstand eingehen. Dieses Problem haben wir im Beweis von Lemma 7.10 für die Ausgaben 0 oder 1 dadurch gelöst, daß wir eine geeignete Substitution auf den Ausgaben vorgenommen haben. Dies ist jedoch im Falle der Ausgabe $\perp$ (also dem Fall, daß keine geliefert wird) nicht möglich, da wir nicht einfach verlangen können, daß die Umgebung anstatt keiner Ausgabe eine Ausgabe x liefert. Die Schwierigkeit liegt hierbei nicht im Halteproblem, da wir völlig unbeschränkte Maschinen betrachten, sondern darin, daß die Umgebung evtl. aus dem Grunde keine Ausgabe liefert, daß sie gar nicht mehr aktiviert wird.

Einen Ausweg aus diesem Dilemma stellt der folgende Ansatz dar: Man definiert das kombinierte Spiel etwas komplizierter: Bevor zufällig gewählt wird, ob das reale oder das ideale Spiel gespielt wird, darf Spieler 1 (die Umgebung)

wählen, ob bei einem Blatt vom Typ $\perp$ der Gewinn im realen 1 und im idealen -1 sein soll, oder umgekehrt. Für dieses veränderte kombinierte Spiel ist in Lemma 7.10 die Bedingung, daß die Umgebung im realen Ausgabe liefert, nicht mehr nötig. Aufgrund der komplexeren Konstruktionen wird die Beweisführung aber aufwendiger. Da unsere Folgerungen aus der Existenz der universellen Umgebung und Simulator (Abschnitt 7.4) diese nur im Falle des Dummy-Angreifers brauchen, haben wir uns auf den einfacheren Fall beschränkt.

7.3. Universelle Umgebung und Simulator

Mit den spieltheoretischen Vorarbeiten des letzten Abschnitts sind wir nun bereit, zur zentralen Konstruktion dieses Kapitels zu kommen, den universellen Umgebungen und Simulatoren. Wir bezeichnen eine Umgebung $\mathcal{Z}^*$ und einen Simulator $\mathcal{S}^*$ als *universell,* wenn keine Umgebung besser als $\mathcal{Z}^*$ unterscheidet (zumindest nicht, wenn $\mathcal{S}^*$ der Simulator ist), und wenn kein Simulator besser simuliert als $\mathcal{S}^*$ (zumindest nicht, wenn die $\mathcal{Z}^*$ die Umgebung ist). Dabei lassen wir jedoch einen kleinen vernachlässigbaren Fehler ε zu, so darf z. B. eine Umgebung um ε besser unterscheiden als die universelle Umgebung $\mathcal{Z}^*$.

Definition 7.11 (Universelle Umgebung und Simulator)
Es seien π und ρ Protokolle und $\mathcal{A}$ ein für π zulässiger Angreifer. Wir nennen eine zulässige Umgebung $\mathcal{Z}^*$ und einen zulässigen Simulator $\mathcal{S}^*$ *universell für π, ρ und $\mathcal{A}$*, wenn eine vernachlässigbare Funktion ε existiert, so daß für jeden (unbeschränkten) zulässigen Simulator $\mathcal{S}'$ und jede (unbeschränkte) zulässige Umgebung $\mathcal{Z}'$ gilt: Für alle $k \in \mathbb{N}$ und alle $z \in \Sigma^*$ ist

$$\begin{aligned}&\Delta\big(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}^*}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}^*}(k,z)\big)\\ \geq\ &\Delta\big(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}'}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}'}(k,z)\big) - \varepsilon(k)\end{aligned}$$

(d. h. wenn die Umgebung $\mathcal{Z}^*$ mit Simulator $\mathcal{S}^*$ läuft, unterscheidet sie mindestens so gut (bis auf einen Fehler ε) wie jede andere Umgebung $\mathcal{Z}'$), und

$$\begin{aligned}&\Delta\big(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}^*}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}^*}(k,z)\big)\\ \leq\ &\Delta\big(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}^*}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}',\mathcal{Z}^*}(k,z)\big) + \varepsilon(k)\end{aligned}$$

(d. h. wenn der Simulator $\mathcal{S}^*$ mit Umgebung $\mathcal{Z}^*$ ausgeführt wird, simuliert er mindestens so gut (bis auf einen Fehler ε) wie jeder andere Simulator $\mathcal{S}'$).

Wir nennen $\mathcal{Z}^*$ und $\mathcal{S}^*$ *universell unter Umgebungen in $C_{\mathcal{Z}}$ und Simulatoren in $C_{\mathcal{S}}$*, wenn obiges für $\mathcal{Z}' \in C_{\mathcal{Z}}$ und $\mathcal{S}' \in C_{\mathcal{S}}$ gilt.

Wenn man diese Definition mit dem Begriff des Nash-Equilibriums vergleicht, wird man feststellen, daß die universellen Umgebung und Simulator eine direkte Übertragung der Idee das Nash-Equilibriums auf unser Sicherheitsmodell sind, wobei statt des Gewinns der Umgebung der von ihr erreichte statistische Abstand betrachtet wird. Lediglich der Fehler ε findet keine Entsprechung bei der Definition des Nash-Equilibriums. Wenn wir aber nicht nur ein einzelnes Spiel betrachten, sondern eine Folge von Spielen, können wir den Begriff des Nash-Equilibriums wie folgt erweitern, so daß auch dort ein vernachlässigbarer Fehler zulässig ist:

Definition 7.12 (Ungefähres Nash-Equilibrium)
Es sei eine Folge von n-Spieler-Spielen $G^{(k)}$ mit Gewinnfunktionen $H_i^{(k)}$ gegeben. Eine Folge von Tupeln von gemischten Strategien $(\mu_1^{(k)}, \ldots, \mu_n^{(k)})$ ist ein *ungefähres Nash-Equilibrium*, wenn eine vernachlässigbare Funktion ε existiert, so daß für jedes $k \in \mathbb{N}$, jedes $i \in \{1, \ldots, n\}$ und jede gemischte Spieler-i-Strategie μ^* gilt:

$$H_i(\mu_1^{(k)}, \ldots, \mu_n^{(k)}) \geq H_i(\mu_1^{(k)}, \ldots, \mu_{i-1}^{(k)}, \mu^*, \mu_{i+1}^{(k)}, \ldots, \mu_n^{(k)}) - \varepsilon(k).$$

Der Zusammenhang zwischen ungefähren Nash-Equilibria und universellen Umgebungen und Simulatoren ist jedoch mehr als nur eine Analogie. Das folgende Lemma 7.14 garantiert uns, daß Umgebungen und Simulatoren, die ein ungefähres Nash-Equilibrium implementieren, universell sind. (Genaugenommen sind sie nur universell unter den Umgebungen und Angreifern mit Kommunikationskomplexität p, da es nur zu solchen Umgebungen und Simulatoren entsprechende Strategien des kombinierten Spiels gibt; dieser Mißstand wird aber in Lemma 7.15 behoben.) Dies ist eine direkte Folgerung aus den in Lemma 7.10 vorgestellten Eigenschaften des kombinierten Spiels. Lemma 7.14 gilt nur, wenn der Angreifer der Dummy-Angreifer ist, dies liegt daran, daß in Lemma 7.10 verlangt wird, daß die Umgebung immer Ausgabe liefert. Wenn man das kombinierte Spiel so abändert, daß diese Bedingung aus Lemma 7.10 verschwindet (siehe die Diskussion nach dem Beweis von Lemma 7.10, S. 235), kann das vorliegende Lemma 7.14 (und damit Lemma 7.15 und Satz 7.16) auch auf beliebige Angreifer mit Kommunikationskomplexität p erweitert werden.

Bevor wir das eigentliche Lemma 7.14 formulieren können, müssen wir aber zunächst eine genaue Definition des Dummy-Angreifers liefern. Da wir keine in ihrer Kommunikationskomplexität unbeschränkten Angreifer zulassen wollen (dann wäre das kombinierte Spiel nicht mehr endlich), können wir nicht die Definition 4.2 des unbeschränkten Dummy-Angreifers verwenden. Stattdessen definieren wir den p-Dummy-Angreifer, welcher zwar wie der unbeschränkte Dummy-Angreifer Nachrichten einfach weiterleitet, dessen Kommunikations-

komplexität aber auf p beschränkt wird (jede darüber hinausgehende Kommunikation wird verworfen):

Definition 7.13 (p-Dummy-Angreifer)
Es sei p eine Funktion und π ein Protokoll. Der *p-Dummy-Angreifer $\tilde{\mathcal{A}}_p$ für π* ist wie folgt definiert:

Für jeden ausgehenden freien Angreiferport `adv_`x von π hat $\tilde{\mathcal{A}}_p$ einen eingehenden Angreiferport `adv_`x und einen ausgehenden Umgebungsport `env_adv_`x.

Für jeden eingehenden freien Angreiferport `adv_`x von π hat $\tilde{\mathcal{A}}_p$ einen ausgehenden Angreiferport `adv_`x und einen eingehenden Umgebungsport `env_adv_`x.

Darüber hinaus hat $\tilde{\mathcal{A}}_p$ den eingehenden Spezialport `clk` und den ausgehenden Umgebungsport `env_clk`.

Wenn $\tilde{\mathcal{A}}_p$ mit einer Nachricht m auf dem eingehenden Umgebungsport `env_adv_`x aktiviert wird, schickt $\tilde{\mathcal{A}}_p$ die Nachricht m_p über den Angreiferport `adv_`x, wobei m_p das Präfix der Länge p von m_p ist.

Wenn $\tilde{\mathcal{A}}_p$ mit einer Nachricht m auf dem eingehenden Angreiferport `adv_`x aktiviert wird, schickt $\tilde{\mathcal{A}}_p$ die Nachricht m_p über den Umgebungsport `env_adv_`x.

Wenn $\tilde{\mathcal{A}}_p$ mit einer Nachricht m auf dem eingehenden Spezialport `clk` aktiviert wird, schickt $\tilde{\mathcal{A}}_p$ die Nachricht m_p über den Umgebungsport `env_clk`.

Es werden aber über jeden Port höchstens p Nachrichten weitergeleitet.

Man beachte, daß der p-Dummy-Angreifer $\tilde{\mathcal{A}}_p$ für π zulässig ist und Kommunikationskomplexität p hat.

Wir können nun das Lemma vorstellen:

Lemma 7.14 (Nash-Equilibria und eingeschränkt universelle Umgebungen und Simulatoren)
Es seien π und ρ Protokolle mit beschränkter Kommunikationskomplexität. Es sei p eine Funktion und $\tilde{\mathcal{A}}$ der p-Dummy-Angreifer für π. Es sei $\mathcal{Z}_0$ eine beliebige Umgebung und $\mathcal{S}_0$ ein beliebiger Simulator, beide mit den natürlichen Ports. Es sei $N_R := \{\pi, \mathcal{A}, \mathcal{Z}_0\}$ und $N_I := \{\rho, \mathcal{S}_0, \mathcal{Z}_0\}$. Und $G^{(k)} := G^C_{p,k}(N_R, N_I)$ sei das kombinierte Spiel zu N_R und N_I.

Es sei $(\mu_1^{(k)}, \mu_2^{(k)})$ ein ungefähres Nash-Equilibrium von $G^{(k)}$. Weiter seien $\mathcal{Z}^*$ eine zulässige Umgebung mit den natürlichen Ports, die $\mu_1^{(k)}$ implementiert (für Sicherheitsparameter k und jeden Auxiliary input) und $\mathcal{S}^*$ ein

(Fortsetzung nächste Seite)

(Fortsetzung)

zulässiger Simulator mit den natürlichen Ports, der $\mu_2^{(k)}$ implementiert.

Dann sind $\mathcal{Z}^*$ und $\mathcal{S}^*$ universell für π, ρ und $\mathcal{A}$ unter den Umgebungen mit Ein-Bit-Ausgabe und Kommunikationskomplexität p und den Simulatoren mit Kommunikationskomplexität p.

Beweis: Im folgenden nennen wir eine Umgebung *erlaubt*, wenn sie zulässig ist, sowie Kommunikationskomplexität p und Ein-Bit-Ausgabe hat. Einen Simulator nennen wir *erlaubt*, wenn er zulässig ist und Kommunikationskomplexität p hat.

Da $(\mu_1^{(k)}, \mu_2^{(k)})$ ein ungefähres Nash-Equilibrium ist, existiert eine vernachlässigbare Funktion ε, so daß für alle $k \in \mathbb{N}$ und alle Strategien (μ_1', μ_2') für $G^{(k)}$ gilt:

$$\begin{aligned} & H_1(\mu_1^{(k)}, \mu_2^{(k)}) \geq H_1(\mu_1', \mu_2^{(k)}) - \varepsilon(k) \\ \text{und} \quad & H_1(\mu_1^{(k)}, \mu_2^{(k)}) \leq H_1(\mu_1^{(k)}, \mu_2') + \varepsilon(k). \end{aligned} \tag{7.1}$$

Hierbei ist H_1 die Gewinnfunktion von Spieler 1 von $G^{(k)}$. Man beachte, daß $G^{(k)}$ ein Nullsummenspiel ist, was für die zweite Ungleichung benötigt wird.

Es seien nun eine erlaubte Umgebung $\mathcal{Z}'$ und ein erlaubte Simulator $\mathcal{S}'$ gegeben. O. B. d. A. können wir annehmen, daß $\mathcal{Z}'$ und $\mathcal{S}'$ die natürlichen Ports haben. Es sei im folgenden $k \in \mathbb{N}$ und $z \in \Sigma^*$ fest.

Wir schreiben im folgenden abkürzend: $\Delta(\mathcal{Z}, \mathcal{S})$ für $\Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}}(k,z); \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)\big)$ und $H(\mathcal{Z}, \mathcal{S})$ für $H_1(\mu_1, \mu_2)$, wobei μ_1, μ_2 die von $\mathcal{Z}$ bzw. $\mathcal{S}$ implementierten Strategien seien (bei Sicherheitsparameter k und Auxiliary input z).

Da der Angreifer der Dummy-Angreifer ist (welcher Nachrichten auf dem Port `clk` immer weiterleitet) und das Protokoll π beschränkte Kommunikationskomplexität hat, liegt es an $\mathcal{Z}'$, wenn $\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}'}(k,z) = \perp$. Wir können daher $\mathcal{Z}'$ so abändern, daß $P(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}'}(k,z) = \perp) = 0$. Wir nehmen daher o. B. d. A. an, daß $P(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}'}(k,z) = \perp) = 0$.

Nach Lemma 7.10 gibt es dann eine erlaubte Umgebung $\mathcal{Z}''$ mit den natürlichen Ports, so daß

$$\Delta(\mathcal{Z}^*, \mathcal{S}^*) \overset{(*)}{\geq} H(\mathcal{Z}^*, \mathcal{S}^*) \overset{(7.1)}{\geq} H(\mathcal{Z}'', \mathcal{S}^*) - \varepsilon(k) \overset{(*)}{=} \Delta(\mathcal{Z}', \mathcal{S}^*) - \varepsilon(k). \tag{7.2}$$

Dabei steht $(*)$ für eine Anwendung von Lemma 7.10. Dies ist gerade die erste Ungleichung aus Definition 7.11.

Im Spezialfall $\mathcal{Z}' = \mathcal{Z}^*$ erhalten wir aus (7.2) insbesondere:

$$\Delta(\mathcal{Z}^*, \mathcal{S}^*) \leq H(\mathcal{Z}^*, \mathcal{S}^*) + \varepsilon(k). \tag{7.3}$$

Es folgt

$$\Delta(\mathcal{Z}^*, \mathcal{S}^*) \overset{(7.3)}{\leq} H(\mathcal{Z}^*, \mathcal{S}^*) + \varepsilon(k) \overset{(7.1)}{\leq} H(\mathcal{Z}^*, \mathcal{S}') + 2\varepsilon(k) \overset{(*)}{\leq} \Delta(\mathcal{Z}^*, \mathcal{S}') + 2\varepsilon(k) \quad (7.4)$$

(Wieder ist $(*)$ eine Anwendung von Lemma 7.10.) Das ist aber gerade die zweite Ungleichung aus Definition 7.11 (für ein größeres, aber immer noch vernachlässigbares ε).

Da (7.2) und (7.4) für alle erlaubten $\mathcal{Z}'$ und $\mathcal{S}'$ und für alle k, z gelten, ist Definition 7.11 erfüllt und das Lemma bewiesen. □

Im Lemma 7.14 haben wir nur gezeigt, daß die dort konstruierten Umgebung $\mathcal{Z}^*$ und Simulator $\mathcal{S}^*$ universell unter Umgebungen und Simulatoren mit Kommunikationskomplexität p und Ein-Bit-Ausgabe sind. Für die Anwendungen aus Abschnitt 7.4 brauchen wir aber, daß $\mathcal{Z}^*$ und $\mathcal{S}^*$ universell sind. Dies liefert das folgende Lemma. Der Beweis basiert darauf, daß gezeigt wird, daß für Protokolle der Kommunikationskomplexität p die Umgebung und der Simulator durch über p hinausgehende Kommunikation keinen Vorteil haben, weil diese entweder ignoriert wird (bei ausgehenden Nachrichten der Umgebung ans Protokoll), nicht auftreten kann (bei eingehenden Nachrichten von Protokoll an Umgebung), oder nicht im Interesse der entsprechenden Maschine ist (bei Kommunikation zwischen Umgebung und Simulator, die über das hinausgeht, was der Angreifer verarbeiten würde).

Lemma 7.15 (Nash-Equilibria und universelle Umgebungen und Simulatoren)
Es seien π, ρ, $\tilde{\mathcal{A}}$, $\mathcal{Z}^*$ und $\mathcal{S}^*$ wie in Lemma 7.14. Dann sind $\mathcal{Z}^*$ und $\mathcal{S}^*$ universell für π, ρ und $\tilde{\mathcal{A}}$.

Beweis: Nach Lemma 7.14 sind $\mathcal{Z}^*$ und $\mathcal{S}^*$ universell unter den Umgebungen mit Ein-Bit-Ausgabe und Kommunikationskomplexität p und den Simulatoren mit Kommunikationskomplexität p. Es sei ε wie in Definition 7.11 der zugehörige Fehler.

Es seien nun eine zulässige Umgebung $\mathcal{Z}'$ und ein zulässiger Simulator $\mathcal{S}'$ gegeben (nicht notwendig mit Kommunikationskomplexität p oder Ein-Bit-Ausgabe). Um das Lemma zu beweisen, müssen wir zeigen, daß die Ungleichungen aus Definition 7.11 auch für diese $\mathcal{Z}'$ und $\mathcal{S}'$ gelten.

Nach Definition des statistischen Abstands existiert eine Funktion T mit Wertebereich $\{0,1\}$, so daß

$$\begin{aligned}&\Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}'}(k,z);\ \mathrm{OUTPUT}_{\pi,\mathcal{S}^*,\mathcal{Z}'}(k,z)\big)\\&\quad=\Delta\big(T(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}'}(k,z),k,z);\ T(\mathrm{OUTPUT}_{\pi,\mathcal{S}^*,\mathcal{Z}'}(k,z),k,z)\big)\end{aligned}\tag{7.5}$$

(Dabei setzen wir $T(\perp) := \perp$.)

Es sei nun $\mathcal{Z}^T$ die Umgebung, die sich wie $\mathcal{Z}'$ verhält, die aber, wenn $\mathcal{Z}'$ Ausgabe o liefern würde, stattdessen die Ausgabe $T(o,k,z)$ liefert. Nach Konstruktion ist

$$\begin{aligned}&\Delta\big(T(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}'}(k,z),k,z);\ T(\mathrm{OUTPUT}_{\pi,\mathcal{S}^*,\mathcal{Z}'}(k,z),k,z)\big)\\&\qquad=\Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}^T}(k,z);\ \mathrm{OUTPUT}_{\pi,\mathcal{S}^*,\mathcal{Z}^T}(k,z)\big)\end{aligned}\tag{7.6}$$

Wir haben also eine Umgebung $\mathcal{Z}^T$ mit Ein-Bit-Ausgabe konstruiert, die (bei Simulator $\mathcal{S}^*$) genauso gut unterscheidet wie die ursprüngliche Umgebung $\mathcal{Z}'$.

Nun konstruieren wir aus $\mathcal{Z}^T$ eine weitere Umgebung $\mathcal{Z}^p$. Diese unterscheidet sich von $\mathcal{Z}^T$ dadurch, daß alle gesandten und eingehenden Nachrichten auf die Länge p gekürzt werden, und auf jedem Port höchstens p Nachrichten gesandt und empfangen werden (d. h., weitere eingehende Nachrichten werden ignoriert).

Da sowohl π als auch $\tilde{\mathcal{A}}$ Kommunikationskomplexität p haben, werden somit von $\mathcal{Z}^p$ nur ausgehende Nachrichten verworfen (bzw. gekürzt), die von $\tilde{\mathcal{A}}$ oder π sowieso nicht bearbeitet würden. Außerdem werden von $\mathcal{Z}^p$ nur eingehende Nachrichten verworfen (bzw. gekürzt), die π und $\tilde{\mathcal{A}}$ gar nicht senden können. Also ist

$$\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}^T}(k,z) = \mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}^p}(k,z).\tag{7.7}$$

Ebenso gilt, da ρ und $\mathcal{S}^*$ Kommunikationskomplexität p haben:

$$\mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}^T}(k,z) = \mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}^p}(k,z).\tag{7.8}$$

Analog zu $\mathcal{Z}^p$ konstruieren wir aus $\mathcal{S}'$ den Simulator $\mathcal{S}^p$ und erhalten (da auch $\mathcal{Z}^*$ Kommunikationskomplexität p hat):

$$\mathrm{OUTPUT}_{\rho,\mathcal{S}',\mathcal{Z}^*}(k,z) = \mathrm{OUTPUT}_{\rho,\mathcal{S}^p,\mathcal{Z}^*}(k,z)\tag{7.9}$$

Da $\mathcal{Z}^p$ Ein-Bit-Ausgabe hat und sowohl $\mathcal{Z}^p$ als auch $\mathcal{S}^p$ Kommunikationskomplexität p haben, sind die Ungleichungen aus Definition 7.11 anwendbar (mit $\mathcal{Z}^p$ statt $\mathcal{Z}'$ und $\mathcal{S}^p$ statt $\mathcal{S}'$), und wir erhalten:

$$\begin{aligned}&\Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}^*}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}^*}(k,z)\big)\\&\qquad\geq\Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}^p}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}^p}(k,z)\big)-\varepsilon(k)\end{aligned}$$

und

$$\Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}^*}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}^*}(k,z)\big) \leq \Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}^*}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}^p,\mathcal{Z}^*}(k,z)\big) + \varepsilon(k).$$

Mit (7.5–7.9) ergibt sich daraus

$$\Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}^*}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}^*}(k,z)\big) \geq \Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}'}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}'}(k,z)\big) - \varepsilon(k)$$

und

$$\Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}^*}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}^*}(k,z)\big) \leq \Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}^*}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}',\mathcal{Z}^*}(k,z)\big) + \varepsilon(k).$$

Dies sind aber gerade die Ungleichungen aus Definition 7.11, somit sind $\mathcal{Z}^*$ und $\mathcal{S}^*$ universell. □

Das vorangegangene Lemma 7.15 sagt uns, daß wir aus einem ungefähren Nash-Equilibrium universelle Umgebung und Simulator konstruieren können. Wir können nun abschätzen, wie aufwendig es ist, ein solches ungefähres Nash-Equilibrium zu finden: Satz 7.6 garantiert uns, daß die Komplexität dafür polynomiell in der Größe des Spielbaums des kombinierten Spiels G ist. Dieser Spielbaum wiederum ist exponentiell in der Kommunikationskomplexität der Protokolle und des Angreifers, und kann sogar in exponentieller Zeit explizit berechnet werden (in der Laufzeit ebendieser Maschinen). Somit können wir, wenn Protokolle und Angreifer Laufzeit p haben, in $EXP(p)$ ein ungefähres Nash-Equilibrium berechnen; universelle Umgebung und Simulator existieren also mit Laufzeit in $EXP(p)$. Haben wir nur eine Schranke für die Kommunikationskomplexität von Protokollen und Angreifern, so können wir den Spielbaum G und damit das ungefähre Nash-Equilibrium nicht berechnen. Da das Nash-Equilibrium aber höchstens Länge $EXP(p)$ hat, können wir es als nichtuniforme Eingabe betrachten, und erhalten somit *nichtuniforme* universelle Umgebung und Simulator mit Laufzeit in $EXP(p)$. Der folgende Satz faßt diese Ergebnisse zusammen:

Satz 7.16 (Existenz universeller Umgebungen und Simulatoren) Es seien π und ρ Protokolle. Es sei p eine Funktion und $\tilde{\mathcal{A}}$ der p-Dummy-Angreifer für π.

(Fortsetzung nächste Seite)

(Fortsetzung)

Weiter haben π und ρ Kommunikationskomplexität p. Dann existieren universelle Umgebung $\mathcal{Z}^*$ und Simulator $\mathcal{S}^*$ für π, ρ und $\tilde{\mathcal{A}}$.

$\mathcal{S}^*$ und $\mathcal{Z}^*$ haben Kommunikationskomplexität p und haben nichtuniforme Laufzeit in $EXP(p)$. Beide Maschinen speichern keine Informationen zwischen Aktivierungen mit Ausnahme ihrer Sicht. $\mathcal{Z}^*$ hat Ein-Bit-Ausgabe und ignoriert ihren Auxiliary input.

Haben π und ρ zusätzlich Laufzeit p, und ist p in deterministischer Zeit $POLY(p)$ berechenbar,[13] so haben $\mathcal{Z}^*$ und $\mathcal{S}^*$ Laufzeit in $EXP(p)$.

Beweis: Wir untersuchen zunächst den Fall, daß π und ρ Kommunikationskomplexität p haben, daß aber keine Schranke für deren Laufzeit vorliegt. Es sei dann $G^{(k)}$ wie in Lemma 7.14. Nach Konstruktion hat $G^{(k)}$ $s_k \in EXP(p)$ Knoten, ist also insbesondere endlich. Nach Satz 7.2 existiert damit ein Nash-Equilibrium $(\mu_1^{(k)}, \mu^{(k)})$ für jedes der Spiele $G^{(k)}$. Wie in [Kuh56] gezeigt, können die gemischten Strategien durch Verhaltensstrategien ersetzt werden (vgl. Fußnote 10), und wir erhalten Verhaltensstrategien $(\beta_1^{(k)}, \beta_2^{(k)})$, so daß $(\beta_1^{(k)}, \beta_2^{(k)})$ ein Nash-Equilibrium in gemischten Strategien für $G^{(k)}$ ist. Jede der Strategien $\beta_i^{(k)}$ besteht aus höchstens s_k Wahrscheinlichkeiten (die angeben, mit welcher Wahrscheinlichkeit bei welcher Informationsmenge welche Aktion ausgeführt werden soll, oder anders: welche Nachricht bei welcher Sicht geschickt werden soll). Diese Wahrscheinlichkeiten sind aber nicht notwendig endlich darstellbar. Es sei $\tilde{\beta}_i^{(k)}$ die Verhaltensstrategie, die entsteht, wenn man jede der Wahrscheinlichkeiten in $\beta_i^{(k)}$ auf $k + \lceil \log_2 s_k \rceil$ (binäre) Nachkommastellen rundet.[14] Ersetzt $\beta_i^{(k)}$ durch $\tilde{\beta}_i^{(k)}$, so ändert sich, unabhängig davon, welches die anderen Strategien sind, der Wert der Gewinnfunktion höchstens um 2^{-k} (hier benutzen wir, daß die Gewinnfunktion von $G^{(k)}$ nur Werte von 0 bis 1 annimmt). Somit ist $(\tilde{\beta}_1^{(k)}, \tilde{\beta}_2^{(k)})$ ein ungefähres Nash-Equilibrium von $G^{(k)}$. Da $\tilde{\beta}_1^{(k)}$ aus maximal $EXP(p)$ Wahrscheinlichkeiten mit einer Darstellungslänge von jeweils $O(k + \log EXP(p))$ besteht, und damit insgesamt eine Länge von $O((k + \log EXP(p)) \cdot EXP(p)) \subseteq EXP(p)$ hat, kann $\tilde{\beta}_1^{(k)}$ von einer nichtuniformen Maschine $\mathcal{Z}^*$ mit Laufzeit in $EXP(p)$ implementiert werden (diese Maschine erhält einfach $\tilde{\beta}_1^{(k)}$ als nichtuniforme Eingabe bei Sicherheitsparameter k und wählt dann alle Nachrichten abhängig von der Sicht entsprechend der von $\tilde{\beta}_1^{(k)}$ vorgegebenen Wahrscheinlichkeiten). Ana-

[13] Damit meinen wir, daß ein deterministischer Algorithmus existiert, der bei Eingabe k nach $POLY(p(k))$ Schritten $p(k)$ ausgibt.

[14] Dabei nehmen wir an, daß so auf- bzw. abgerunden wird, daß sich die zu einem Knoten gehörigen Wahrscheinlichkeiten weiterhin zu 1 addieren, da sonst die resultierenden Objekte keine Verhaltensstrategien mehr wären.

log gibt es eine nichtuniforme Maschine $\mathcal{S}^*$, die in Laufzeit $EXP(p)$ die Strategie $\tilde{\beta}_2^{(k)}$ implementiert. Nach Lemma 7.15 sind $\mathcal{Z}^*$ und $\mathcal{S}^*$ universell. Nach Konstruktion von $G^{(k)}$ hat jede Maschine, die eine Strategie von $G^{(k)}$ implementiert, notwendig Kommunikationskomplexität p und die Ausgabe einer Maschine, die eine Spieler-1-Strategie implementiert (hier $\mathcal{Z}^*$) hat notwendig Ein-Bit-Ausgabe (da das reale und das ideale Spiel in Definition 7.7 gerade so definiert wurden, daß Simulator und Umgebung maximal Kommunikationskomplexität p haben dürfen, und der Umgebung nur die Ausgaben 0 und 1 erlaubt wurden). Da die Wahrscheinlichkeiten der Nachrichten nur von der Sicht abhängen, brauchen $\mathcal{Z}^*$ und $\mathcal{S}^*$ außer letzterer keine Informationen zu speichern.

Wir wenden uns nun dem Fall zu, daß π und ρ zusätzlich Laufzeit p haben und wollen zeigen, daß $\mathcal{Z}^*$ und $\mathcal{S}^*$ so konstruiert werden können, daß sie (uniforme) Laufzeit in $EXP(p)$ haben. Da wir in diesem Fall den Maschinen die Verhaltensstrategien $\tilde{\beta}_i^{(k)}$ nicht als nichtuniforme Eingabe liefern können, müssen diese von den Maschinen selbst berechnet werden. Dafür muß zunächst eine explizite Darstellung von $G^{(k)}$ berechnet werden. Um die Wahrscheinlichkeiten für die verschiedenen Nachrichten, die eine Maschine mit Laufzeit p in einem gegebenen Zustand senden kann, zu berechnen, sind höchstens $EXP(p)$ Schritte notwendig. Daher kann die explizite Darstellung des Spiels $G^{(k)}$ (welches $EXP(p)$ Knoten hat) in deterministischer Zeit $EXP(p) \cdot EXP(p) \subseteq EXP(p)$ berechnet werden. Nach Satz 7.6 können dann in deterministischer Zeit $POLY(EXP(p)) \subseteq EXP(p)$ Verhaltensstrategien $(\beta_1^{(k)}, \beta_2^{(k)})$ berechnet werden, die ein Nash-Equilibrium von $G^{(k)}$ bilden. Leider ist es nicht unbedingt möglich, diese Strategien zu implementieren, da die Wahrscheinlichkeiten durch rationale Zahlen gegeben sind (und nicht durch abbrechende Binärdarstellung). So ist es bereits nicht möglich, mit einer Turingmaschine, der ein binäres Zufallsband zur Verfügung steht, in beschränkter Zeit mit Wahrscheinlichkeit $\frac{1}{3}$ eine 1 auszugeben (man kann natürlich beliebig nah an $\frac{1}{3}$ herankommen). Daher runden wir die Wahrscheinlichkeiten in $\beta_i^{(k)}$ wie oben und erhalten somit ein ungefähres Nash-Equilibrium $(\tilde{\beta}_1^{(k)}, \tilde{\beta}_2^{(k)})$. Dieses kann in deterministischer $EXP(p)$-Zeit berechnet werden, und alle Wahrscheinlichkeiten sind in abbrechender Binärdarstellung der Länge $O(k + \log EXP(p))$ gegeben. Somit kann $\tilde{\beta}_1^{(k)}$ in nichtuniformer Laufzeit $EXP(p)$ implementiert werden durch eine Maschine $\mathcal{Z}^*$ mit Laufzeit in $EXP(p)$, die in jeder Aktivierung zunächst $\tilde{\beta}_1^{(k)}$ berechnet (hier benötigen wir, daß $\tilde{\beta}_1^{(k)}$ *deterministisch* berechnet wird, damit in den verschiedenen Aktivierungen immer die gleiche Strategie verwendet wird), und dann die zu sendenden Nachrichten in Abhängigkeit von der Sicht gemäß $\tilde{\beta}_1^{(k)}$ wählt. $\mathcal{S}^*$ wird analog konstruiert. Damit sind $\mathcal{Z}^*$ und $\mathcal{S}^*$ nach Lemma 7.15 universell. Daß $\mathcal{S}^*$ und $\mathcal{Z}^*$ Kommunikationskomplexität p haben und daß $\mathcal{Z}^*$ Ein-Bit-Ausgabe hat, ergibt sich wie oben. Beide Maschinen speichern keine Information außer ihrer

Sicht, da die Strategie bei jeder Aktivierung neu berechnet wird. □

Wir haben in diesem Abschnitt universelle Umgebungen und Simulatoren kennengelernt, festgestellt, daß diese tatsächlich existieren, und sogar explizite Schranken für deren Laufzeit bestimmt. Wofür diese universellen Umgebungen und Simulatoren aber überhaupt genutzt werden können, wird der nächste Abschnitt verraten.

7.4. Folgerungen

Der vorangehende Abschnitt hat uns das Werkzeug der universellen Umgebungen und Simulatoren an die Hand gegeben. Wir wollen nun zeigen, wie man diese verwenden kann, um im Falle von Protokollen mit beschränkter Kommunikationskomplexität die Äquivalenz verschiedener Sicherheitsbegriffe zu zeigen, darunter insbesondere allgemeine und spezielle statistische Sicherheit.

Bevor wir mit diesen Analysen beginnen, führen wir noch eine weitere Variante der Sicherheit ein, die *Sicherheit mit beschränktem Risiko*. Wir führen diese hier ein, da der Beweis von Satz 7.19 sie in natürlicher Weise mit einbezieht. Diese Variante der Sicherheit ist aber auch unabhängig davon von Interesse, da sie eine natürliche Motivation hat und in manchen Fällen der Begriff sein mag, den man eigentlich möchte. Die meisten Sicherheitsdefinitionen (insbesondere alle, die wir in dieser Arbeit bisher vorgestellt haben, mit Ausnahme der perfekten Sicherheit), verlangen, daß die Erfolgswahrscheinlichkeit jedes Angreifers (im Falle simulierbarer Sicherheit meint das den Erfolg der Umgebung im Unterscheiden) durch eine vernachlässigbare Funktion beschränkt ist. Wenn wir aber ein Protokoll tatsächlich einsetzen wollen, mag uns dies nicht genügen. Wir werden uns dann nämlich der Frage stellen müssen, wie hoch der Sicherheitsparameter gewählt werden muß, um die Erfolgschance des Angreifers auf, sagen wir, 2^{-80} zu beschränken. Doch diese Schranke für die Erfolgschance des Angreifers kann durchaus vom Angreifer abhängen. Damit obige Frage sinnvoll ist, wir das „Risiko der Protokollausführung“ also explizit begrenzen können, ist es nötig, daß die Schranke für den Erfolg des Angreifers nur vom Sicherheitsparameter, nicht aber vom konkreten Angreifer abhängt. Auf den Fall der statistischen[15] Sicherheit übertragen, erhalten wir die folgende Definition:

[15] Man könnte die gleiche Definition natürlich auch auf die perfekte und die komplexitätstheoretische Sicherheit anwenden. Im Falle der perfekten jedoch ist dies nicht nötig, da dort der statistische Abstand durch 0, also insbesondere durch eine feste Schranke beschränkt ist, und im Falle der komplexitätstheoretischen Sicherheit wäre dies wohl zu streng, da es dort natürlich ist, daß der Erfolg des Angreifers mit seinem Aufwand steigt. Im komplexitätstheoretischen Falle wäre es sinnvoller, eine Definition zu verwenden, die z. B. den Quotienten aus Erfolgswahrscheinlichkeit und Laufzeit des Angreifers betrachtet. Das jedoch würde hier zu weit führen.

Definition 7.17 (Sicherheit mit beschränktem Risiko)
Es seien π und ρ Protokolle.

Dann ist π so sicher wie ρ bezüglich *statistischer Sicherheit mit beschränktem Risiko mit Auxiliary input bzgl. der Ausgabe der Umgebung*, wenn eine vernachlässigbare Funktion μ existiert, so daß für jeden zulässigen Angreifer $\mathcal{A}$ ein zulässiger Simulator $\mathcal{S}$ existiert, so daß für alle zulässigen Umgebungen $\mathcal{Z}$ und für alle $k \in \mathbb{N}$ und $z \in \Sigma^*$ gilt:

$$\Delta\big(\mathrm{OUTPUT}_{\pi,\mathcal{A},\mathcal{Z}}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)\big) \leq \mu(k).$$

Die Varianten der speziellen Sicherheit mit beschränktem Risiko *ohne Auxiliary input, bzgl. der Sicht der Umgebung* und *mit Angreifern, Simulatoren und Umgebungen in C* sind analog zu den entsprechenden Varianten der allgemeinen Sicherheit definiert (vgl. Definition 2.14).

Bevor wir mit unseren Betrachtungen beginnen, wollen wir noch eine einfache Abgeschlossenheitseigenschaft für Mengen von Maschinen definieren. Wir brauchen diese Eigenschaft, um Satz 7.19 möglichst allgemein formulieren zu können.

Definition 7.18 (Abgeschlossenheit)
Wir nennen eine Menge M von Maschinen *abgeschlossen unter Kombination,* wenn für zwei Maschinen $A, B \in M$ auch die Maschine C in M liegt, wobei C die Maschine ist, die simultan die Maschinen A und B simuliert.[16]

Wir nennen M *abgeschlossen unter Weglassen von Nachrichten,* wenn für jede Maschine $A \in M$, jeden ausgehenden Port p von A und jede Nachricht m gilt, daß auch A' in M ist, wobei A' das folgende Programm hat: Es wird A simuliert, aber wenn A die Nachricht m über p senden würde, terminiert A' stattdessen.

Wir nennen M *abgeschlossen,* wenn M abgeschlossen unter Kombination und unter Weglassen von Nachrichten ist.

Man sieht leicht, daß für jede Menge F von Funktionen, die Menge der Maschinen mit Laufzeit in $POLY(F)$ abgeschlossen ist. Das gleiche gilt für Maschinen mit nichtuniformer Laufzeit in $POLY(F)$.

Wir können nun das erste Ergebnis dieses Abschnittes formulieren. Es besagt, daß Sicherheit mit unbeschränkten Angreifern, Umgebungen und Simulatoren äquivalent ist zu Sicherheit mit Angreifern, Umgebungen und Simulatoren, deren Laufzeit exponentiell in der des Protokolls ist. Genauer: Wenn die Klasse

[16] Mit evtl. notwendigen Portumbenennungen.

der Umgebungen und der Simulatoren die exponentiell-beschränkten enthält, und die Klasse der Angreifer die polynomiell-beschränkten, so ist Sicherheit mit diesen Angreifern, Umgebungen und Simulatoren äquivalent zur Sicherheit mit unbegrenzten Angreifern, Umgebungen und Simulatoren. Insbesondere sind auch davon Sicherheitsbegriffe eingeschlossen, bei denen z. B. die Umgebungen exponentiell-beschränkt und die Simulatoren unbeschränkt sind, oder umgekehrt. Grob gesprochen besagt der Satz, daß wir unbeschränkte Angreifer, Umgebungen und Simulatoren o. B. d. A. als exponentiell-beschränkt annehmen können.

Die Grundidee des Beweises ist die folgende: Zunächst kann man die Menge der Angreifer auf eine den Dummy-Angreifer enthaltende einschränken. Dann konstruieren wir universelle Umgebung und Simulator, und sehen ein, daß der Sicherheitsbegriff sich nicht ändert, wenn wir uns auf diese einschränken. Dies ist nicht weiter überraschend, da die universellen Umgebung und Simulator ja optimal im Sinne eines Nash-Equilibriums sind. Die Möglichkeit, uns auf universelle Umgebung und Simulator einzuschränken, liefert auch eine Rechtfertigung für die Bezeichnung „universell“. Da die universellen Umgebung und Simulator nach Satz 7.19 exponentielle Laufzeit haben, enthält eine Menge, die alle exponentiellen Maschinen enthält, insbesondere auch die universellen Umgebung und Simulator, woraus der Satz folgt.

Satz 7.19 (Exponentielle Angreifer sind „unbeschränkt“)
Es sei p eine in deterministischer Zeit $POLY(p)$ berechenbare Funktion,[17] und π und ρ Protokolle mit Laufzeit p.

Es seien weiter $C_{\mathcal{Z}}$ und $C_{\mathcal{S}}$ Mengen von Maschinen, die die Menge aller Maschinen mit Laufzeit in $EXP(p)$ mit Ein-Bit-Ausgabe enthalten, es sei $C_{\mathcal{S}}$ abgeschlossen (gem. Definition 7.18), und $C_{\mathcal{A}} \subseteq C_{\mathcal{S}}$ sei eine Menge von Maschinen, die alle Maschinen mit Laufzeit in $POLY(p)$ enthält.

Dann ist π so sicher wie ρ bezüglich statistischer Sicherheit genau dann, wenn π so sicher wie ρ ist bezüglich statistischer Sicherheit mit Angreifern in $C_{\mathcal{A}}$, Simulatoren in $C_{\mathcal{S}}$ und Umgebungen in $C_{\mathcal{Z}}$.

Dies gilt für die statistische *allgemeine, spezielle* Sicherheit und Sicherheit *mit beschränktem Risiko*, jeweils *mit und ohne Auxiliary input*, *bzgl. der Ausgabe* und *bzgl. der Sicht der Umgebung*.

Insbesondere können wir, wenn π und ρ polynomiell-beschränkte Protokolle sind, $C_{\mathcal{Z}} = C_{\mathcal{S}} = C_{\mathcal{A}}$ als die Menge der exponentiell-beschränkten Maschinen wählen, d. h. Sicherheit mit unbeschränkten und mit exponentiell-beschränkten Angreifern, Simulatoren und Umgebungen fallen zusammen.

[17] Damit meinen wir, daß ein deterministischer Algorithmus existiert, der bei Eingabe k nach

Der tatsächliche Beweis ist etwas umfangreicher als die obige Skizze, da wir alle im Satz angegebenen Varianten der Sicherheit betrachten müssen. Um es zu vermeiden, viele im wesentlichen gleiche, sich in den Details unterscheidenden Variationen obiger Beweisidee niederschreiben zu müssen, beweisen wir eine etwas stärkere Aussage, die im wesentlichen die Äquivalenz aller im Satz betrachteter Sicherheitsbegriffe ist. Diese stärkere Aussage können wir dann auch wiederverwerten für den Beweis von Korollar 7.20.

Beweis: Um diesen Beweis übersichtlicher formulieren zu können, definieren wir zunächst einige Abkürzungen für die verschiedenen Varianten der statistischen Sicherheit:

(BR) π ist so sicher wie ρ bezüglich Sicherheit mit beschränktem Risiko.

(AS) π ist so sicher wie ρ bezüglich allgemeiner Sicherheit.

(SS) π ist so sicher wie ρ bezüglich spezieller Sicherheit.

Diese Abkürzungen können noch ergänzt werden durch

a mit Angreifern in $C_{\mathcal{A}}$

p mit dem p-Dummy-Angreifer

s mit Simulatoren in $C_{\mathcal{S}}$

z mit Umgebungen in $C_{\mathcal{Z}}$

o ohne Auxiliary input (ansonsten nehmen wir Auxiliary input an)

v bzgl. der Sicht der Umgebung (ansonsten bzgl. der Ausgabe)

Es bedeutet z. B. (BRpzv): „π ist so sicher wie ρ bezüglich statistischer Sicherheit mit beschränktem Risiko mit Auxiliary input mit dem p-Dummy-Angreifer und Umgebungen in $C_{\mathcal{Z}}$ bzgl. der Sicht der Umgebung“.

Um den Satz zu beweisen, zeigen wir eine stärkere Aussage:

Wenn π und ρ Kommunikationskomplexität p haben, und es universelle Umgebung $\mathcal{Z}^*$ und Simulator $\mathcal{S}^*$ für den p-Dummy-Angreifer $\tilde{\mathcal{A}}_p$ gibt, so daß $\mathcal{Z}^* \in C_{\mathcal{Z}}$, $\mathcal{S}^* \in C_{\mathcal{S}}$, $C_{\mathcal{S}}$ abgeschlossen ist, und $C_{\mathcal{A}} \subseteq C_{\mathcal{S}}$, und jede Maschine mit Laufzeit in $POLY(p)$ in $C_{\mathcal{A}}$ liegt, und $\mathcal{Z}^*$ seinen Auxiliary input nicht verwendet, dann sind alle folgenden Aussagen äquivalent:

(I) (BR), (BRp), (BRps), (BRas), (BRazs), (AS), (ASazs), (SS), (SSas), (SSpz), (SSazs),

$POLY(p(k))$ Schritten $p(k)$ ausgibt.

(II) (BRo), (BRpo), (BRpso), (BRaso), (BRazso), (ASo), (ASazso), (SSo), (SSaso), (SSpzo), (SSazso),

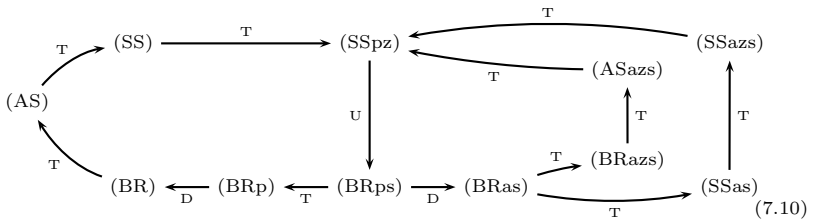
(III) (BRv), (BRpv), (BRpsv), (BRasv), (BRazsv), (ASv), (ASazsv), (SSv), (SSasv), (SSpzv), (SSazsv),

(IV) (BRov), (BRpov), (BRpsov), (BRasov), (BRazsov), (ASov), (ASazsov), (SSov), (SSasov), (SSpzov), (SSazsov).

(Wir haben die Aussagen in vier Gruppen (I)–(IV) eingeteilt, abhängig davon, ob Auxiliary vorliegt oder nicht, und abhängig davon, ob die Sicht oder die Ausgabe der Umgebung betrachtet wird. Dies dient lediglich dazu, den Beweis weiter unten besser formulieren zu können.)

Aus diesen Äquivalenzen ergibt sich dann wie folgt der Satz: Mit Satz 7.16 erhalten wir die Existenz von universellen $\mathcal{Z}^*$ und $\mathcal{S}^*$, die Laufzeit in $EXP(p)$ haben. Daher sind $\mathcal{Z}^* \in C_{\mathcal{Z}}$ und $\mathcal{S}^* \in C_{\mathcal{S}}$, und auch die anderen oben geforderten Bedingungen an $C_{\mathcal{A}}$, $C_{\mathcal{S}}$ und $C_{\mathcal{Z}}$ sind erfüllt. Somit sind die Aussagen der Gruppen (I)–(IV) alle äquivalent, und insbesondere die vom zu zeigenden Korollar behaupteten Äquivalenzen gelten (dies sind nämlich $(XXyy) \Leftrightarrow (XX\text{azs}yy)$ für $XX \in \{\text{BR}, \text{AS}, \text{SS}\}$ und $yy \in \{\lambda, \text{o}, \text{v}, \text{ov}\}$).

Wir beweisen zunächst die Implikationen im folgenden Diagramm (7.10). Daraus folgt dann die Äquivalenz aller Aussagen aus Gruppe (I).



(7.10)

Die mit T markierten Implikationen folgen trivial aus der Definition der verschiedenen Varianten der Sicherheit. Es bleibt also, die Implikationen (SSpz) ⇒ (BRps) und (BRp) ⇒ (BR) und (BRps) ⇒ (BRas) zu zeigen. Zwei von diesen, (BRp) ⇒ (BR) und (BRps) ⇒ (BRas) (in (7.10) mit D markiert) lassen sich mit der Dummy-Angreifer-Technik beweisen. Da dies ganz analog zur Beweisskizze von Lemma 4.3 geht, beschränken wir uns darauf, anzumerken, was man gegenüber Lemma 4.3 zusätzlich beachten muß:

- Da der p-Dummy-Angreifer nur p Nachrichten der Länge höchstens p weiterleitet, ist zu prüfen, ob nicht Nachrichten(suffixe) verschluckt werden. Da aber auch das Protokoll π Kommunikationskomplexität p hat, ist dem nicht so.

- Da wir (zumindest bei der zweiten Implikation) nur Simulatoren in $C_{\mathcal{S}}$ zulassen, muß geprüft werden, ob der durch Kombination des Angreifers und des Simulators entstehende neue Simulator (im Beweis von Lemma 4.3 $\mathcal{S}_{\mathcal{Z}}$ genannt) auch in $C_{\mathcal{S}}$ liegt. Dem ist aber so, da $C_{\mathcal{A}} \subseteq C_{\mathcal{S}}$ und $C_{\mathcal{S}}$ unter Kombination abgeschlossen ist.

- Der p-Dummy-Angreifer muß in der Menge $C_{\mathcal{A}}$ der zulässigen Angreifer liegen (bei der zweiten Implikation). Dies ist aber nach Voraussetzung gegeben.

Es bleibt die Implikation (SSpz) $\Rightarrow$ (BRps) zu zeigen (in (7.10) mit U markiert). Dazu zeigen wir die etwas stärkere Aussage (SSpzo) $\Rightarrow$ (BRps). Hierzu benötigen wir die universelle Umgebung $\mathcal{Z}^*$ und Simulator $\mathcal{S}^*$.

Es gelte also (SSpzo). Es gibt dann für jede Umgebung $\mathcal{Z} \in C_{\mathcal{Z}}$ einen Simulator $\mathcal{S}_{\mathcal{Z}}$ und eine vernachlässigbare Funktion $\mu_{\mathcal{Z}}$, so daß für alle $k \in \mathbb{N}$ gilt:

$$\Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}}_p,\mathcal{Z}}(k);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}_{\mathcal{Z}},\mathcal{Z}}(k)\big) \leq \mu_{\mathcal{Z}}(k).$$

Aus der Universalität von $\mathcal{Z}^* \in C_{\mathcal{Z}}$ und $\mathcal{S}^*$ folgt daraus (mit einer geeigneten vernachlässigbare Funktion ε, vgl. Definition 7.11) für eine beliebige Umgebung $\mathcal{Z}$ und alle $k, z \in \Sigma^*$:

$$\begin{aligned}
&\Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}}_p,\mathcal{Z}'}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}'}(k,z)\big)\\
&\quad\overset{(*)}{\leq} \Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}}_p,\mathcal{Z}^*}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}^*}(k,z)\big) + \varepsilon(k)\\
&\quad\overset{(**)}{=} \Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}}_p,\mathcal{Z}^*}(k);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}^*,\mathcal{Z}^*}(k)\big) + \varepsilon(k)\\
&\quad\overset{(*)}{\leq} \Delta\big(\mathrm{OUTPUT}_{\pi,\tilde{\mathcal{A}}_p,\mathcal{Z}^*}(k);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}_{\mathcal{Z}^*},\mathcal{Z}^*}(k)\big) + 2\varepsilon(k)\\
&\quad\leq \mu_{\mathcal{Z}^*}(k) + 2\varepsilon(k) =: \mu^*(k).
\end{aligned}$$

Hierbei bezeichnet $(*)$ eine Anwendung der Universalität von $\mathcal{Z}^*$ und $\mathcal{S}^*$, und $(**)$ nutzt die Tatsache, daß $\mathcal{Z}^*$ seinen Auxiliary input ignoriert.

Da $\mathcal{S}^* \in C_{\mathcal{S}}$, und sowohl $\mathcal{S}^*$ als auch die vernachlässigbare Funktion μ nicht von $\mathcal{Z}'$ abhängen, impliziert dies (BRps). Somit ist (SSpzo) $\Rightarrow$ (BRps) gezeigt, woraus (SSpz) $\Rightarrow$ (BRps) folgt; alle Implikationen in (7.10) sind korrekt und somit alle Aussagen aus Gruppe (I) äquivalent.

Die Äquivalenzen der Gruppe (II) zeigt man ganz analog, da die verwendeten Argumente sowohl mit als auch ohne Auxiliary input Gültigkeit haben.

Weiterhin ist trivialerweise (BRps) $\Rightarrow$ (BRpso), also ist

$$\text{(BRps)} \Rightarrow \text{(BRpso)} \Leftrightarrow \text{(SSpzo)} \Rightarrow \text{(BRps)}$$

Somit sind alle Aussagen aus den Gruppen (I) und (II) äquivalent.

Um zu beweisen, daß auch die Aussagen aus Gruppe (III) äquivalent zu denen aus (I) und (II) sind, zeigen wir zunächst (BRas) ⇒ (BRpsv).

Sei also (BRas) gegeben. Um (BRpsv) zu beweisen, müssen wir eine vernachlässigbare Funktion μ und einen Simulator $\mathcal{S} \in C_{\mathcal{S}}$ konstruieren, so daß für jede Umgebung $\mathcal{Z}$ gilt:

$$\Delta\big(\mathrm{VIEW}_{\pi,\tilde{\mathcal{A}}_p,\mathcal{Z}}(k,z);\ \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)\big) \leq \mu(k).$$

Es sei $\mathcal{A}' \in C_{\mathcal{A}}$ ein Angreifer, der sich wie der Dummy-Angreifer $\tilde{\mathcal{A}}_p$ verhält, mit dem folgenden Unterschied: Wenn $\mathcal{A}'$ die erste Nachricht über den Spezialport `clk` erhält, die $\tilde{\mathcal{A}}_p$ ignorieren würde, schickt $\mathcal{A}'$ eine spezielle Nachricht *terminated* über den Umgebungsport `env_clk` an die Umgebung. Es sei $\mathcal{S}' \in C_{\mathcal{S}}$ der nach (BRas) zu $\mathcal{A}$ gehörige Simulator, d. h.

$$\Delta\big(\mathrm{OUTPUT}_{\pi,\mathcal{A}',\mathcal{Z}}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}',\mathcal{Z}}(k,z)\big) \leq \mu(k)$$

für geeignetes vernachlässigbares μ und alle Umgebungen $\mathcal{Z}$.

Weiter sei $\mathcal{S} \in C_{\mathcal{S}}$ der Simulator, der sich wie $\mathcal{S}'$ verhält, nur daß $\mathcal{S}$ anstatt *terminated* über den Umgebungsport `env_clk` zu senden, einfach terminiert. (Wir nutzen, daß $C_{\mathcal{S}}$ abgeschlossen ist.)

Zu einer Umgebung $\mathcal{Z}$ sei $\mathcal{Z}'$ die Umgebung, die ihre bisherige Sicht ausgibt, wenn sie *terminated* über den Umgebungsport `env_clk` erhält.

Mit dieser Konstruktion ergibt sich für jede Umgebung $\mathcal{Z}$:

$$\begin{aligned}&\Delta\big(\mathrm{VIEW}_{\pi,\tilde{\mathcal{A}}_p,\mathcal{Z}}(k,z);\ \mathrm{VIEW}_{\rho,\mathcal{S},\mathcal{Z}}(k,z)\big)\\ &\qquad = \Delta\big(\mathrm{OUTPUT}_{\pi,\mathcal{A}',\mathcal{Z}'}(k,z);\ \mathrm{OUTPUT}_{\rho,\mathcal{S}',\mathcal{Z}'}(k,z)\big) \leq \mu(k),\end{aligned}$$

also folgt (BRpsv).

Die Beweise für die Implikationen aus (7.10) mit Ausnahme der mit U markierten übertragen sich direkt auf die Aussagen der Gruppe (III). Somit wird jede Aussage der Gruppe (III) von (BRpsv) impliziert. Weiterhin ist trivialerweise (BRpsv) ⇒ (BRps), und somit

$$\text{(I)} \Rightarrow \text{(BRas)} \Rightarrow \text{(BRpsv)} \Rightarrow \text{(III)} \Rightarrow \text{(BRpsv)} \Rightarrow \text{(I)}.$$

Also sind die Aussagen aus Gruppe (III) äquivalent zu denen aus Gruppe (I).

Analog zeigt man, daß die Aussagen aus Gruppe (IV) äquivalent zu denen aus Gruppe (II) sind.

Insgesamt sind die Aussagen aus allen vier Gruppen äquivalent, und der Satz folgt. □

Wir können die Existenz von universellen Umgebung und Simulator nicht nur dafür benutzen, zu zeigen, daß exponentielle Angreifer, Umgebungen und Simulatoren im wesentlichen so mächtig sind wie unbeschränkte (Satz 7.19), es fallen

durch deren Existenz noch viel mehr Sicherheitsbegriffe zusammen. Zum einen verschwindet der Unterschied zwischen spezieller und allgemeiner Sicherheit (intuitiv ist der Grund, daß die Reihenfolge der Quantoren, mit denen Umgebung und Simulator gewählt werden, unwichtig ist, wenn sowieso nur universelle Umgebung und Simulator betrachtet werden, da diese ja nach Wahl des Angreifers bereits festliegen), zum anderen stellt es sich als irrelevant heraus, ob wir z. B. Sicherheit mit oder ohne Auxiliary input betrachten, etc. Die komplette Liste der Äquivalenzen ist recht umfangreich (es werden 80 verschiedene Sicherheitsbegriffe betrachtet), man entnehme diese dem Korollar 7.20.

Interessant ist es hier, das Korollar 7.20 mit Satz 5.1 zu vergleichen. Letzterer Satz besagte, daß spezielle und allgemeine Sicherheit im statistischen Falle *nicht* zusammenfallen. Das dazugehörige trennende Beispiel bestand aus einem Spiel, bei dem Umgebung und Simulator (unabhängig) Zahlen wählen mußten, und wer die größere Zahl wählte, gewann. Dieses Spiel hat kein Nash-Equilibrium, und darin liegt auch der Grund, daß wir in diesem Fall keine universelle Umgebung und Simulator konstruieren konnten. Ist das Protokoll jedoch in seiner Kommunikation beschränkt, so gibt es immer ein Nash-Equilibrium, und somit universelle Umgebung und Simulator, und darin liegt der Grund, warum Korollar 7.20 auf Protokolle mit beschränkter Kommunikation anwendbar ist, aber auf Protokolle mit unbeschränkter Kommunikation wie das Beispiel von Satz 5.1 nicht.

Korollar 7.20 (Äquivalenz verschiedener Varianten der Sicherheit)

Es seien π und ρ Protokolle.

Wir betrachten fünf Gruppen von je sechzehn Aussagen.

Gruppe A:

- π ist so sicher wie ρ bezüglich *allgemeiner* statistischer Sicherheit mit/ohne Auxiliary input bzgl. der Sicht/der Ausgabe der Umgebung.
- ... bezüglich *spezieller* statistischer Sicherheit mit/ohne Auxiliary input bzgl. der Sicht/der Ausgabe der Umgebung.
- ... bezüglich *spezieller* statistischer Sicherheit mit/ohne Auxiliary input bzgl. der Sicht/der Ausgabe der Umgebung mit Umgebungen *mit Ein-Bit-Ausgabe*.
- ... bezüglich statistischer Sicherheit *mit beschränktem Risiko* mit/ohne Auxiliary input bzgl. der Sicht/der Ausgabe der Umgebung.

Gruppe A':

- Die Aussagen der Gruppe A jeweils mit *nichtuniform Turing-unbeschränkten* Angreifern, Simulatoren und Umgebungen.

(Fortsetzung nächste Seite)

(Fortsetzung)

Gruppe B:

- Die Aussagen der Gruppe A jeweils mit *Turing-unbeschränkten* Angreifern, Simulatoren und Umgebungen.

Gruppe C:

- Die Aussagen der Gruppe A jeweils mit *nichtuniform exponentiell-beschränkten* Angreifern, Simulatoren und Umgebungen.

Gruppe D:

- Die Aussagen der Gruppe A jeweils mit *exponentiell-beschränkten* Angreifern, Simulatoren und Umgebungen.

Haben π und ρ beschränkte Kommunikationskomplexität, so sind die Aussagen der Gruppen A und A' äquivalent.

Haben π und ρ beschränkte Laufzeit, so sind die Aussagen der Gruppen A, A' und B äquivalent.

Haben π und ρ polynomiell-beschränkte Kommunikationskomplexität, so sind die Aussagen der Gruppen A, A' und C äquivalent.

Haben π und ρ polynomiell-beschränkte Laufzeit, so sind die Aussagen der Gruppen A, A', B, C und D äquivalent.

Im wesentlichen haben wir dieses Korollar bereits im Beweis von Satz 7.19 mit abgehandelt, daher faßt der Beweis des Korollars nur die verschiedenen Spezialfälle des Beweises von Satz 7.19 zusammen.

Beweis: Abkürzend bezeichnen wir die Aussage, daß π so sicher wie ρ bezüglich allgemeiner statistischer Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung ist, als $(*)$. Die Aussage $(*)$ ist in Gruppe A.

Zunächst betrachten wir den Fall, daß π und ρ beschränkte Kommunikationskomplexität haben und wollen die Äquivalenz der Aussagen der Gruppen A und B zeigen.

Da π und ρ beschränkte Kommunikationskomplexität haben, gibt es eine Funktion p, so daß π und ρ Kommunikationskomplexität p haben. Also existieren nach Satz 7.16 universelle Umgebung $\mathcal{Z}^*$ und Simulator $\mathcal{S}^*$ mit Kommunikationskomplexität p, wobei $\mathcal{Z}^*$ seinen Auxiliary input nicht verwendet. $\mathcal{Z}^*$ und $\mathcal{S}^*$ haben nichtuniforme Laufzeit in $EXP(p)$, sind also nichtuniform Turing-unbeschränkt.

Wir setzen dann $C_{\mathcal{S}} = C_{\mathcal{A}}$ als die Menge aller Maschinen, und $C_{\mathcal{Z}}$ als die Menge aller Maschinen mit Ein-Bit-Ausgabe. Nach dem Beweis von Satz 7.19 sind dann die auf Seite 248 aufgeführten Aussagen der Gruppen (I)–(IV) äquivalent. In der Notation des Beweises von Satz 7.19 lauten die Aussagen der

Gruppe A: (AS), (ASo), (ASv), (ASov), (SS), (SSo), (SSv), (SSov), (SSazs), (SSazso), (SSazsv), (SSazsov), (BR), (BRo), (BRv), (BRov). Da diese äquivalent sind, sind also alle Aussagen der Gruppe A äquivalent.

Nun setzen wir $C_{\mathcal{S}} = C_{\mathcal{A}} = C_{\mathcal{Z}}$ als die Menge aller nichtuniform Turing-unbeschränkten Maschinen.

Auch in diesem Fall greift der Beweis von Satz 7.19. In der Notation von Satz 7.19 lauten die Aussagen aus Gruppe A' (ohne die Sicherheitsbegriffe mit Ein-Bit-Ausgabe): (ASazs), (ASazso), (ASazsv), (ASazsov), (SSazs), (SSazso), (SSazsv), (SSazsov), (BRazs), (BRazso), (BRazsv), (BRazsov). Außerdem ist $(*)$ das gleiche wie (AS). Somit sind die Begriffe aus Gruppe A' mit Ausnahme derer mit Ein-Bit-Ausgabe zu $(*)$ äquivalent.

Nun setzen wir $C_{\mathcal{A}} = C_{\mathcal{S}}$ als die Menge aller nichtuniform Turing-unbeschränkten Maschinen, und $C_{\mathcal{S}}$ als die Menge aller nichtuniform Turing-unbeschränkten Maschinen mit Ein-Bit-Ausgabe. Wieder greift der Beweis von Satz 7.19, und die Aussagen aus Gruppe A' mit Ein-Bit-Ausgabe sind (SSazs), (SSazso), (SSazsv), (SSazsov) und zu (AS), also $(*)$ äquivalent.

Damit folgt, daß alle Aussagen aus den Gruppen A und A' zu $(*)$ und somit zueinander äquivalent sind.

Nun betrachten wir den Fall, daß π und ρ beschränkte Laufzeit haben. Da dies beschränkte Kommunikationskomplexität impliziert, sind die Aussagen aus den Gruppen A und A' äquivalent. Wir müssen nur zeigen, daß die Aussagen aus Gruppe B zu $(*)$ äquivalent sind. Es haben π und ρ Laufzeit p für geeignetes p.[18] Nach Satz 7.16 gibt es universelle Umgebung $\mathcal{Z}^*$ und Simulator $\mathcal{S}^*$ mit Kommunikationskomplexität p, wobei $\mathcal{Z}^*$ seinen Auxiliary input nicht verwendet. $\mathcal{Z}^*$ und $\mathcal{S}^*$ haben Laufzeit in $EXP(p)$, sind also Turing-unbeschränkt.

Mit $C_{\mathcal{A}} = C_{\mathcal{S}} = C_{\mathcal{Z}}$ als der Menge der Turing-unbeschränkten Maschinen greift wieder der Beweis von Satz 7.19, und wir erhalten wie oben, daß die Aussagen aus Gruppe B mit Ausnahme derer mit Ein-Bit-Ausgabe zu $(*)$ äquivalent sind. Die Aussagen aus Gruppe B mit Ein-Bit-Ausgabe erhalten wir, wenn wir $C_{\mathcal{Z}}$ auf Maschinen mit Ein-Bit-Ausgabe einschränken.

Somit sind die Aussagen der Gruppen A, A' und B äquivalent.

Nun betrachten wir den Fall, daß π und ρ polynomiell-beschränkte Kommunikationskomplexität haben. Da daraus beschränkte Kommunikationskomplexität folgt, sind die Aussagen aus den Gruppen A und A' äquivalent. Es bleibt die Äquivalenz der Aussagen aus Gruppe C zu $(*)$ zu zeigen. In diesem Fall können $\mathcal{Z}^*$ und $\mathcal{S}^*$ nach Satz 7.19 als nichtuniform exponentiell-beschränkt gewählt

[18] Insbesondere können wir p so groß wählen, daß p in deterministischer Zeit $POLY(p)$ berechenbar ist: Die Laufzeit p' einer laufzeitbeschränkten probabilistischen Turingmaschine ist immer deterministisch berechenbar (durch Simulation aller Pfade). Es sei p die Zeit, die man braucht, um p' zu berechnen (aber mindestens $p \geq p'$). Dann ist p in $POLY(p)$ berechenbar.

werden, und mit $C_{\mathcal{A}} = C_{\mathcal{S}} = C_{\mathcal{Z}}$ als der Menge der nichtuniform exponentiell-beschränkten Maschinen (mit der Einschränkung von $C_{\mathcal{Z}}$ auf Maschinen mit Ein-Bit-Ausgabe für manche der Aussagen) kann wie oben die Äquivalenz der Aussagen der Gruppe C zu $(*)$ gezeigt werden.

Somit sind die Aussagen der Gruppen A, A' und C äquivalent.

Nun betrachten wir den Fall, daß π und ρ polynomiell-beschränkte Laufzeit. Da daraus polynomiell-beschränkte Kommunikationskomplexität und beschränkte Laufzeit folgen, sind die Aussagen aus den Gruppen A, A', B und C äquivalent. Es bleibt die Äquivalenz der Aussagen aus Gruppe D zu $(*)$ zu zeigen. Satz 7.16 garantiert exponentiell-beschränkte $\mathcal{Z}^*$ und $\mathcal{S}^*$, und mit $C_{\mathcal{A}} = C_{\mathcal{S}} = C_{\mathcal{Z}}$ als der Menge der exponentiell-beschränkten Maschinen (mit der Einschränkung von $C_{\mathcal{Z}}$ auf Maschinen mit Ein-Bit-Ausgabe für manche der Aussagen) kann wie oben die Äquivalenz der Aussagen der Gruppe D zu $(*)$ gezeigt werden.

Somit sind die Aussagen der Gruppen A, A', B, C und D äquivalent. □

Eine einfacher Folgerung aus Korollar 7.20 ist, daß die dort genannten Sicherheitsbegriffe auch allgemeine Komposition erlauben (jeweils unter den Bedingungen, unter denen sie zur Gruppe A äquivalent sind): Da die allgemeine Sicherheit allgemeine Komposition erlaubt, so erlauben auch zur allgemeinen Sicherheit äquivalente Begriffe allgemeine Komposition.

8. Schlußbemerkungen

8.1. Fazit

In dieser Arbeit haben wir uns die Frage gestellt, welche Implikationen und Trennungen zwischen den Begriffen der speziellen und allgemeinen Sicherheit und der einfachen und allgemeinen Komponierbarkeit bestehen. Es hat sich gezeigt, daß die Antwort in einigen Fällen von der betrachteten Variante der Sicherheit abhängt, sprich ob wir perfekte, statistische oder komplexitätstheoretische Sicherheit betrachten, und ob wir Sicherheit mit oder ohne Auxiliary input verwenden. Für jeden dieser Fälle haben wir alle Beziehungen zwischen den vier Begriffen erarbeitet.

Im Zusammenhang mit diesen Untersuchungen haben wir darüber hinaus die Time-lock puzzles als mächtiges Werkzeug für die Konstruktion von trennenden Beispielen erkannt. Wir haben den Begriff der Time-lock puzzles formalisiert und deren Existenz untersucht. Die Time-lock puzzles scheinen das Potential zu haben, auch jenseits der hier betrachteten Fragestellung die Konstruktion von nichttrivialen Gegenbeispielen zu erlauben.

Weiterhin haben wir erkannt, wie man spieltheoretische Ergebnisse für die Analyse der Beziehungen zwischen den Sicherheitsbegriffen verwenden kann. Dies führte zur Definition und Untersuchung der universellen Umgebungen und Simulatoren, ein Konzept, das nicht nur dazu dient, die Äquivalenz verschiedener Varianten der Sicherheit zu zeigen, sondern das auch dazu benutzt werden kann, z. B. zu zeigen, daß exponentielle Angreifer, Umgebungen und Simulatoren vollständig sind.

Angesichts der Untersuchungen in dieser Arbeit kann man sich nun die Frage stellen, welcher der untersuchten Begriffe denn nun zu verwenden ist. Zwar ist die Antwort auf diese Frage sicherlich von der jeweiligen Anwendung abhängig, aber wir wollen die folgenden Faustregeln vorschlagen:

- *Soll allgemeine oder spezielle Sicherheit verwendet werden?* Wir schlagen die Verwendung der allgemeinen Sicherheit vor: Die allgemeine Komposition ist für viele Anwendungen unabdingbar, und zumindest im komplexitätstheoretischen Fall ist die spezielle Sicherheit nicht hinreichend für allgemeine Komponierbarkeit. Darüber hinaus gibt es bislang keine Indizien dafür, daß allgemeine Sicherheit, obgleich der echte striktere Begriff,

in *praktischen* Anwendungen schwieriger zu zeigen ist. Im Falle der statistischen und perfekten Sicherheit ist spezielle Sicherheit zwar hinreichend für allgemeine Komposition, aber es bietet sich an, auch hier allgemeine Sicherheit zu verwenden, um einheitlichere Definitionen zu erhalten.

- *Soll Sicherheit mit oder ohne Auxiliary input verwendet werden?* Da Komplexitätsannahmen existieren, die nur ohne Auxiliary input Sinn machen (z. B. Kollisionsresistenz einer Hashfunktion), ist Sicherheit mit Auxiliary input sicherlich sogar aus praktischer Sicht eine echt stärkere Forderung. Man muß sich daher die Frage stellen, welchen Vorteil der Auxiliary input überhaupt hat, um diesen Nachteil aufzuwiegen. Historisch ist der Auxiliary input daraus erwachsen, daß manche Begriffe der Sicherheit (z. B. Zero-Knowledge, sichere Funktionsauswertung ohne Umgebung, vgl. Abschnitt 2.1) nur mit Auxiliary input komponieren. Man kann sich daher auf den Standpunkt stellen, daß Sicherheit mit Auxiliary input gewissermaßen „mehr Komponierbarkeit“ erlaubt. Im Falle der Sicherheit mit Umgebung jedoch bringt der Auxiliary input allem Anschein nach keinen Vorteil: Die Komponierbarkeit wird durch die Umgebung ermöglicht, der Auxiliary input ist hierfür nicht mehr nötig.[1] Auch gibt in den von uns untersuchten Fällen die Sicherheit mit Auxiliary input in keiner Situation eine stärkere Kompositionsgarantie als die Sicherheit ohne Auxiliary input. Wir stellen uns daher auf den Standpunkt, daß der Auxiliary input bei der Sicherheit *mit Umgebung* nur historisch begründet ist. Wir empfehlen daher als Standardbegriff die Sicherheit ohne Auxiliary input zu verwenden (wenn man natürlich Sicherheit mit Auxiliary input zeigen kann, so ist es jedoch sicherlich nicht von Nachteil, dies zu tun).

- *Soll Sicherheit bzgl. der Ausgabe oder der Sicht der Umgebung verwendet werden?* Diese Frage ist schwer zu beantworten. Wir haben gesehen, daß die Begriffe tatsächlich unterschiedlich sind, jedoch bezieht sich der Unterschied prinzipiell nur auf Fälle, in denen das Protokoll nicht terminiert oder zumindest die Umgebung nicht mehr aktiviert wird. Diese Fälle sind sicherlich nicht von praktischer Relevanz, und man mag sich auf den Standpunkt stellen, daß daher der einfacher zu handhabende der beiden Begriffe zu wählen ist. Dies scheint die Sicherheit bzgl. der Ausgabe der Umgebung zu sein; so sind hier z. B. keine Überlegungen über die Kodierung der Sicht notwendig, und bei der Kombination der Umgebung

[1] Der Hauptgrund, warum der Auxiliary input bei anderen Sicherheitsdefinitionen Komposition ermöglicht, liegt darin, daß er aus anderen Protokolläufen erhaltene Informationen repräsentieren kann. In unserem Szenario jedoch können andere Protokollinstanzen als Teil der Umgebung aufgefaßt werden, welche dann gewissermaßen die Aufgabe des Auxiliary inputs übernimmt.

mit anderen Maschinen (wie dies z. B. im Beweis des Kompositionstheorems stattfindet) kann man einfach die kombinierte Maschine die Ausgabe der ursprünglichen Umgebung ausgeben lassen; es ist nicht nötig sicherzustellen, daß die Sicht der ursprünglichen Umgebung tatsächlich aus der gesamten Sicht rekonstruiert werden kann.[2] Will man also den einfacheren Begriff verwenden, so ist wohl die Sicherheit bzgl. der Ausgabe zu verwenden. Will man aber sicherheitshalber den strengeren Begriff verwenden, so sollte man die Sicherheit bzgl. der Sicht benutzen. Wegen der oben erwähnten Tatsache, daß die Trennungen zwischen den beiden Varianten der Sicherheit sehr akademisch sind (und anscheinend prinzipiell sein müssen), tendieren wir zur Sicherheit bzgl. der Ausgabe der Umgebung.

8.2. Offene Fragen

Im folgenden listen wir einige Fragen auf, die die Untersuchungen dieser Arbeit offen lassen oder erst aufwerfen:

- Die in Kapiteln 5 und 6 präsentierten trennenden Beispiele sind unbestritten keine Protokolle aus der Anwendung. Es stellt sich daher die Frage, ob nicht natürlichere trennende Beispiele existieren, d. h. Protokolle, wie man sie tatsächlich verwenden würde, und die speziell sicher, aber nicht allgemein sicher oder nicht allgemein komponierbar sind. Besser noch wären trennende kryptographische Aufgabenstellungen, also eine (wenn möglich natürliche) Funktionalität, die bezüglich des einen Sicherheitsbegriffs implementierbar ist, bezüglich des anderen aber nicht.

- In Kapitel 4 wurde gezeigt, daß spezielle statistische Sicherheit für die nebenläufige Komposition hinreichend ist. Wir haben dabei von der Tatsache Gebrauch gemacht, daß der Simulator unbeschränkt sein darf. Es wird aber manchmal eine Variante der speziellen Sicherheit propagiert, in der der Simulator polynomiell-beschränkt in der Laufzeit des Angreifers sein soll. Es stellt sich die Frage, ob auch in dieser Modellierung nebenläufige Komposition möglich ist.

- In letzter Zeit wurde an neuen Formulierungen des Polynomialzeitbegriffs in der Sicherheit mit Umgebung geforscht [HMQU05a, Can05, Küs06]. Es wäre interessant, welche der (komplexitätstheoretischen) Ergebnisse bei diesen neuen Begriffen noch gelten.

[2] Als Beispiel für die Schwierigkeiten, die bei der Arbeit mit der Sicht der Umgebung auftreten können, sei die Tatsache genannt, daß das Modell von [BPW04b] eine auf der Sicht basierende Definition der statistischen Sicherheit hatte, die nicht einmal einfache Komposition erlaubte [HU05b]. Das Problem lag tatsächlich in der Tatsache, daß die dortige Definition der Sicht nicht garantierte, daß die Sicht der ursprünglichen Umgebung in der Sicht einer sie enthaltenden kombinierten Maschine enthalten ist.

- Die Idee der simulationsbasierten Sicherheit mit Umgebung läßt sich in natürlicher Weise auch auf die Quantenkryptographie übertragen [BOM04, Unr04c]. Es stellt sich dann die Frage, welche der hier präsentierten Ergebnisse in diesem Szenario noch gelten. Vermutlich lassen sich die Ergebnisse der Kapitel 3 bis 6 relativ direkt übertragen; inwiefern die Ergebnisse aus Kapitel 7 jedoch weiterhin gelten, ist unklar.

- Das hier vorgestellte Sicherheitsmodell ist relativ komplex, und schon kleine Änderungen des Modells können zu Änderungen der Resultate oder gar zu Inkonsistenzen führen. Dies ist nicht nur ein Problem des vorliegenden Modells, auch die Modelle von [BPW04b] und [Can01, Can05] sind sehr komplex. Ein wichtiges Ziel in der Untersuchung der Sicherheit ist es also, ein einfaches, übersichtliches und verständliches Modell der Sicherheit zu finden, mit dem sich gut arbeiten läßt. Erste Versuche in diese Richtung wurden z. B. von [MMS03, DKMR05] getätigt, die das Sicherheitsmodell auf Prozeß-Calculi aufsetzen. Doch bislang scheint kein einfaches und übersichtliches Modell zu existieren. Dies ist vor allem deshalb kritisch, weil es dazu führen kann, daß Ergebnisse „bewiesen“ werden, ohne daß die Details der dem Beweis zugrunde liegenden Definitionen bekannt oder spezifiziert sind. Die Suche nach dem einfachen Modell ist daher ein wichtiges Ziel in der Kryptographie, und wir hoffen, zu dieser Suche beigetragen zu haben, indem wir klarer herausgestellt haben, welche Details überhaupt welche Auswirkung haben.

- Verwandt mit dieser Frage ist auch das folgende Problem: Die simulationsbasierte Sicherheit mit Umgebung ist sehr streng, so streng, daß viele natürliche Aufgabenstellungen ohne Zusatzannahmen nicht realisierbar sind [CF01, CKL03]. Da diese Unmöglichkeitsresultate sich auch auf die spezielle Sicherheit beziehen, kann man nicht erwarten, diese zu umgehen, solange man mindestens einfache Komponierbarkeit haben möchte. Doch mag es sinnvoll sein, gewisse kontrollierte Einschränkungen bei der Komposition in Kauf zu nehmen, um dafür einen weniger strengen Sicherheitsbegriff zu erhalten. Verschiedene Ansätze wurden für dieses Problem präsentiert [PS04, BS05, BDHK06], doch bislang wurde keine universell zufriedenstellende Lösung gefunden.

- Das Konzept der Time-lock puzzles scheint ein für die Konstruktion trennender Beispiele nützliches Werkzeug zu sein. Deshalb wäre es interessant, genauer zu untersuchen, unter welchen Bedingungen diese existieren. Unter welchen Bedingungen ist z. B. eine Hashfunktion schwer iterierbar?

8.3. Danksagungen

Ich danke Prof. Dr. Michael Backes, Prof. Dr. Thomas Beth, Prof. Dr. Roland Vollmar, Dr. Dennis Hofheinz, Dr. Jörn Müller-Quade und nicht zuletzt meiner geliebten Gattin Anna.

A. Einige Lemmata

In diesem Anhang listen wir der Vollständigkeit halber einige Lemmata auf, die triviale oder wohlbekannte Ergebnisse darstellen. Wir geben jeweils nur kurze Beweisansätze.

Lemma A.1 (Umgebungen mit Ein-Bit-Ausgabe sind vollständig) Es seien π und ρ Protokolle.

Dann ist π genau dann so sicher wie ρ bezüglich perfekter/statistischer/komplexitätstheoretischer *allgemeiner* Sicherheit mit/ohne Auxiliary input bzgl. der *Ausgabe* der Umgebung, wenn π so sicher wie ρ ist bezüglich des gleichen Sicherheitsbegriffs mit Umgebungen mit Ein-Bit-Ausgabe.

Beweisskizze: Nach Definition der komplexitätstheoretischen Ununterscheidbarkeit sind zwei Familien von Verteilungen X und Y komplexitätstheoretisch ununterscheidbar, wenn es eine polynomiell-beschränkte Maschine D mit Ein-Bit-Ausgabe gibt, so daß $D(X)$ und $D(Y)$ ununterscheidbar sind. Da bei der allgemeinen Sicherheit die Umgebung zuletzt gewählt wird, kann man diese Maschine D in die Umgebung integrieren, d. h. die Umgebung gibt statt x das Bit $D(x)$ aus. Damit folgt aus der komplexitätstheoretischen Sicherheit mit Umgebungen mit Ein-Bit-Ausgabe die komplexitätstheoretische Sicherheit.

Im Falle der statistischen Sicherheit gehen wir analog vor. Zwei Familien von Verteilungen X und Y sind statistisch ununterscheidbar, wenn es eine Familie von Mengen T gibt, so daß die Wahrscheinlichkeiten $P(X \in T)$ und $P(Y \in T)$ nicht vernachlässigbare Differenz haben. Modifizieren wir die Umgebung dahingehend, daß sie statt x auszugeben, ausgibt, ob $x \in T$, erhalten wir eine Umgebung mit Ein-Bit-Ausgabe, die ebensogut unterscheidet.

Im Falle der perfekten Sicherheit brauchen wir nur auszunutzen, daß für je zwei verschiedene Verteilungen X und Y eine Menge T existiert, so daß $P(X \in T)$ und $P(Y \in T)$ verschieden sind. Dann gehen wir vor wie im Falle der statistischen Sicherheit. □

Lemma A.2 (Auxiliary input kann in die Umgebung integriert werden)
Es seien π und ρ Protokolle. Für perfekte/statistische/komplexitätstheoretische *allgemeine* Sicherheit bzgl. der Sicht/der Ausgabe der Umgebung sind die folgenden Aussagen äquivalent:

- π ist so sicher wie ρ bezüglich Sicherheit *mit* Auxiliary input.
- π ist so sicher wie ρ bezüglich Sicherheit *ohne* Auxiliary input (im komplexitätstheoretischen Fall: Sicherheit ohne Auxiliary input mit *nichtuniform* polynomiell-beschränkten Umgebungen).

Beweisskizze: Da im Falle der allgemeinen Sicherheit die Umgebung zuletzt gewählt wird, können wir eine unterscheidende Umgebung durch eine ersetzen, die den Auxiliary input, bei dem sie am besten unterscheidet, fest eingebaut hat. Im Falle der komplexitätstheoretischen Sicherheit war die ursprüngliche Umgebung polynomiell-beschränkt, die resultierende ist also nichtuniform polynomiell-beschränkt (denn der fest eingebaute Auxiliary input hängt in nichtuniformer Weise vom Sicherheitsparameter ab). □

Lemma A.3 (Komplexitätstheoretische Sicherheit ist statistische Sicherheit mit polynomiell-beschränkten Angreifern, Simulatoren und Umgebungen mit Ein-Bit-Ausgabe)
Es seien π und ρ Protokolle. Dann ist π genau dann so sicher wie ρ bzgl. allgemeiner *komplexitätstheoretischer* Sicherheit mit/ohne Auxiliary input bzgl. der Ausgabe der Umgebung, wenn π so sicher wie ρ ist bzgl. allgemeiner *statistischer* Sicherheit mit/ohne Auxiliary input bzgl. der Ausgabe der Umgebung mit polynomiell-beschränkten Angreifern und Simulatoren und polynomiell-beschränkten Umgebungen mit Ein-Bit-Ausgabe.

Beweisskizze: Nach Lemma A.1 ist allgemeine komplexitätstheoretische Sicherheit mit/ohne Auxiliary input bzgl. der Ausgabe der Umgebung äquivalent zur allgemeinen komplexitätstheoretischen Sicherheit mit/ohne Auxiliary input bzgl. der Ausgabe der Umgebung mit polynomiell-beschränkten Umgebungen mit Ein-Bit-Ausgabe und polynomiell-beschränkten Angreifern und Simulatoren. Diese unterscheidet sich von der statistischen Sicherheit mit/ohne Auxiliary input bzgl. der Ausgabe der Umgebung mit polynomiell-beschränkten Umgebungen mit Ein-Bit-Ausgabe und polynomiell-beschränkten Angreifern und

Simulatoren nur darin, daß die statistische Sicherheit die statistische Ununterscheidbarkeit der Ausgabe der Umgebung verlangt (und nicht nur die komplexitätstheoretische). Da aber die Ausgabe der Umgebung nur die Werte $\{0, 1, \perp\}$ annehmen kann, fallen nach Lemma 1.5 (iii) in diesem Fall komplexitätstheoretische und statistische Ununterscheidbarkeit zusammen. □

Lemma A.4 (Allgemeine Sicherheit impliziert spezielle Sicherheit)
Es seien π und ρ Protokolle. Wenn π so sicher wie ρ bezüglich allgemeiner Sicherheit ist, dann ist π so sicher wie ρ bezüglich spezieller Sicherheit.

Dies gilt für perfekte/statistische/komplexitätstheoretische Sicherheit mit/ohne Auxiliary input bzgl. der Ausgabe/Sicht der Umgebung.

Beweisskizze: Dies folgt direkt aus der Definition der allgemeinen und speziellen Sicherheit (bei der ersteren muß es einen von der Umgebung unabhängigen guten Simulator geben, bei der zweiten darf der Simulator von der Umgebung abhängen). □

Lemma A.5 (Sicherheit mit Auxiliary input impliziert Sicherheit ohne Auxiliary input)
Es seien π und ρ Protokolle. Wenn π so sicher wie ρ bezüglich Sicherheit *mit* Auxiliary input ist, dann ist π so sicher wie ρ bezüglich Sicherheit *ohne* Auxiliary input.

Dies gilt für perfekte/statistische/komplexitätstheoretische allgemeine/spezielle Sicherheit bzgl. der Sicht/der Ausgabe der Umgebung.

Beweisskizze: Die Umgebung kann den Auxiliary input ignorieren. □

Lemma A.6 (Perfekte Sicherheit ist echt strenger als statistische Sicherheit)
Das folgende gilt für Sicherheit mit und ohne Auxiliary input, sowie bzgl. der Sicht und bzgl. der Ausgabe der Umgebung.

Es seien π und ρ Protokolle. Wenn π so sicher wie ρ ist bezüglich perfekter Sicherheit, dann ist π so sicher wie ρ bezüglich statistischer Sicherheit.

(Fortsetzung nächste Seite)

(Fortsetzung)

Außerdem existieren polynomiell-beschränkte Protokolle π und ρ, so daß π so sicher wie ρ ist bezüglich statistischer Sicherheit, aber nicht bezüglich perfekter Sicherheit.

Beweisskizze: Perfekte Sicherheit impliziert per Definition statistische Sicherheit (da die perfekte Sicherheit die Gleichheit, die statistische nur die Ununterscheidbarkeit von Verteilungen fordert).

Es sei π ein Protokoll, das bei der ersten Aktivierung die Nachricht 0 an die Umgebung schickt. Das ideale Protokoll ρ hingegen sende mit Wahrscheinlichkeit 2^{-k} eine 1, sonst eine 0 (wobei k der Sicherheitsparameter sei). Die von π und ρ geschickten Nachrichten sind statistisch ununterscheidbar, aber haben nicht die gleiche Verteilung. Somit ist π so sicher wie ρ bezüglich statistischer, aber nicht bezüglich perfekter Sicherheit. □

Lemma A.7 (Statistische/perfekte und komplexitätstheoretische Sicherheit sind verschieden)
Wenn Einweg-Funktionen (Definition 1.6) existieren, gilt das folgende für Sicherheit mit und ohne Auxiliary input, sowie bzgl. der Sicht und bzgl. der Ausgabe der Umgebung.

Es existieren polynomiell-beschränkte Protokolle π und ρ, so daß π so sicher wie ρ ist bezüglich statistischer und perfekter Sicherheit, aber nicht bezüglich komplexitätstheoretischer Sicherheit.

Weiterhin existieren polynomiell-beschränkte Protokolle π und ρ, so daß π so sicher wie ρ ist bezüglich komplexitätstheoretischer Sicherheit, aber nicht bezüglich statistischer oder perfekter Sicherheit.

Beweisskizze: Wir geben zunächst Protokolle an, so daß π so sicher wie ρ ist bezüglich komplexitätstheoretischer Sicherheit, nicht aber bezüglich statistischer oder perfekter Sicherheit. Sowohl π als auch ρ schicken zunächst ein Bild $f(x)$ unter einer Einweg-Funktion f an die Umgebung. Wenn die Umgebung mit einem Urbild x' von $f(x)$ antwortet, geben π und ρ ihre jeweilige Identität aus. Die Umgebung kann also genau dann unterscheiden, wenn sie die Einweg-Funktion f invertieren kann. Dies kann nur eine unbeschränkte Umgebung, somit ist π so sicher wie ρ bezüglich komplexitätstheoretischer Sicherheit, aber nicht bezüglich statistischer oder perfekter Sicherheit.

Nun geben wir Protokolle an, so daß π so sicher wie ρ ist bezüglich statistischer und perfekter Sicherheit, aber nicht bezüglich komplexitätstheoretischer

Sicherheit. Bei Aktivierung durch die Umgebung sendet π eine feste Nachricht m an die Umgebung. Das ideale Protokoll ρ hingegen sendet bei Erhalt einer Nachricht von der Umgebung zunächst $f(x)$ an den Simulator, und nur wenn der Simulator mit einem Urbild x' von $f(x)$ antwortet, wird m an die Umgebung geschickt. Offensichtlich ist ein Simulator nur erfolgreich, wenn er die Einweg-Funktion f invertieren kann, dies ist nur bei der statistischen und der perfekten Sicherheit gegeben. Damit ist π so sicher wie ρ bezüglich statistischer und perfekter Sicherheit, aber nicht bezüglich komplexitätstheoretischer. □

Lemma A.8 (Komplexitätstheoretische Sicherheit mit und ohne Auxiliary input sind verschieden)
Wenn eine kollisionsresistente Hashfunktion (Definition 1.7) existiert, dann existieren Protokolle π und ρ, so daß gilt:

- π ist so sicher wie ρ bezüglich allgemeiner und spezieller komplexitätstheoretischer Sicherheit *ohne* Auxiliary input sowohl bzgl. der Ausgabe als auch bzgl. der Sicht der Umgebung.
- π ist *nicht* so sicher wie ρ weder bezüglich allgemeiner noch spezieller komplexitätstheoretischer Sicherheit *mit* Auxiliary input weder bzgl. der Ausgabe noch bzgl. der Sicht der Umgebung.

Beweisskizze: Es sei f die kollisionsresistente Hashfunktion. Wenn π von der Umgebung eine Kollision unter f (d. h. ein Paar $x \neq x'$ mit $f(x) = f(x')$) erhält, dann sendet π die Nachricht *real*. Das Protokoll ρ sendet nie eine Nachricht. Da der Auxiliary input eine solche Kollision enthalten kann, kann die Umgebung mit Auxiliary input unterscheiden. Eine Umgebung ohne Auxiliary input jedoch kann nach Definition 1.7 keine solche Kollision finden und damit auch nicht unterscheiden. □

Lemma A.9 (Sicherheit bzgl. der Sicht impliziert Sicherheit bzgl. der Ausgabe)
Es seien π und ρ Protokolle. Wenn π so sicher wie ρ bezüglich Sicherheit bzgl. der Sicht der Umgebung ist, dann ist π so sicher wie ρ bezüglich Sicherheit bzgl. der Ausgabe der Umgebung.

Dies gilt für perfekte/statistische/komplexitätstheoretische allgemeine/spezielle Sicherheit mit/ohne Auxiliary input. Im Falle der komplexitätstheoretischen Sicherheit müssen π und ρ polynomiell-beschränkte ausgehende Kommunikationskomplexität haben.

Beweisskizze: Da die Ausgabe Teil der Sicht ist, impliziert Gleichheit/statistische Ununterscheidbarkeit/komplexitätstheoretische Ununterscheidbarkeit der Sicht die Gleichheit/statistische Ununterscheidbarkeit/komplexitätstheoretische Ununterscheidbarkeit der Ausgabe. Im komplexitätstheoretischen Falle benötigt man dabei noch, daß die Ausgabe effizient aus der Sicht extrahierbar ist. Da aber π und ρ polynomiell-beschränkte ausgehende Kommunikationskomplexität haben, ist die Sicht von polynomiell-beschränkter Länge und damit die Ausgabe effizient extrahierbar. □

Lemma A.10 (Sicherheit bzgl. Sicht und Ausgabe sind äquivalent für beschränkte Protokolle)
Es seien π und ρ Protokolle *mit beschränkter ausgehender Kommunikationskomplexität* (im komplexitätstheoretischen Falle mit *polynomiell-beschränkter* ausgehender Kommunikationskomplexität).

Dann ist π genau dann so sicher wie ρ bezüglich perfekter/statistischer/komplexitätstheoretischer spezieller/allgemeiner Sicherheit mit/ohne Auxiliary input bzgl. der *Ausgabe* der Umgebung, wenn π so sicher wie ρ bezüglich perfekter/statistischer/komplexitätstheoretischer mit/ohne Auxiliary input bzgl. der *Sicht* der Umgebung ist.

Beweisskizze: Wegen Lemma A.9 brauchen wir nur zu zeigen, daß aus Sicherheit bzgl. der Ausgabe Sicherheit bzgl. der Sicht folgt.

Da das Protokoll beschränkte Kommunikationskomplexität hat, können wir o. B. d. A. auch annehmen, daß der Angreifer beschränkte Kommunikationskomplexität hat.

Weiterhin können wir o. B. d. A. Angreifer annehmen, die, bevor sie terminieren, eine spezielle Nachricht *end* an die Umgebung schicken (formal bedeutet dies, daß wenn der ursprüngliche Angreifer den `clk`-Port schließen würde, und eine Nachricht an diesen gesandt würde, der modifizierte Angreifer die Nachricht *end* an die Umgebung schickt). Die Umgebung können wir entsprechend modifizieren: bei Empfang der Nachricht *end* gibt sie ihre bisherige Sicht aus.

Da das Protokoll beschränkte ausgehende Kommunikationskomplexität hat, wird also im realen Modell mit Wahrscheinlichkeit 1 die Nachricht *end* irgendwann an die Umgebung geschickt. Damit ist die Ausgabe der veränderten Umgebung gleich der Sicht der ursprünglichen Umgebung, die neue Umgebung unterscheidet also bzgl. der Ausgabe, wenn die alte bzgl. der Sicht unterschieden hat.

Im komplexitätstheoretischen Falle braucht obige Konstruktion noch, daß die Sicht der Umgebung (welche sie ausgibt) polynomiell-beschränkt ist. Dies ist

aber gegeben, da die Protokolle polynomiell-beschränkte ausgehende Kommunikationskomplexität haben. □

B. Beziehungen zwischen Sicherheitsbegriffen

B.1. Erläuterungen

Im folgende geben wir eine kompakte Auflistung der Beziehungen zwischen den verschiedenen Sicherheitsbegriffen, die in dieser Arbeit vorkommen. Angegeben sind nicht nur die gezeigten Implikationen und Trennungen, sondern auch alle durch Anwendung elementarer Aussagenlogik daraus folgenden.

Aus Platzgründen schreiben wir nicht die vollständigen Namen der Sicherheitsbegriffe aus, sondern verwenden die folgende Kodierung:

■	allgemeine Sicherheit
□	spezielle Sicherheit
◀	(polynomiell-beschränkte) allgemeine Komponierbarkeit
◁	einfache Komponierbarkeit
s	statistische Sicherheit
k	komplexitätstheoretische Sicherheit
$\downarrow$	mit Auxiliary input
$\not\downarrow$	ohne Auxiliary input
$\times$	bzgl. der Ausgabe der Umgebung
$\otimes$	bzgl. der Sicht der Umgebung
1	mit Umgebungen mit Ein-Bit-Ausgabe
N	mit nichtuniform polynomiell-beschränkten Umgebungen
P	mit polynomiell-beschränkten Angreifern, Umgebungen und Simulatoren

Es bezeichnet also z. B. s■$\not\downarrow\times$1P die statistische allgemeine Sicherheit ohne Auxiliary input bzgl. der Ausgabe der Umgebung mit polynomiell-beschränkten Umgebungen mit Ein-Bit-Ausgabe und mit polynomiell-beschränkten Angreifern und Simulatoren.

Dem Leser mag auffallen, daß in dieser Tabelle die „Sicherheit mit beschränktem Risiko“ und die Einschränkungen „mit (nichtuniform) Turing-unbeschränkten/exponentiell-beschränkten Angreifern, Umgebungen und Simulatoren“ fehlen, welche in Korollar 7.20 vorkommen. Dies liegt daran, daß die Hinzunahme dieser Begriffe zu einer deutlichen Verlängerung dieses Anhangs führen würde. Bei Interesse an einem dieser Begriffe möge der Leser mittels Korollar 7.20 zunächst einen dazu äquivalenten bestimmen, und diesen dann in der unten

stehenden Tabelle aufsuchen. Zur Tatsache, daß die perfekte Sicherheit keine Abkürzung hat, siehe unten.

Um die Relationen zwischen den Sicherheitsbegriffen zu charakterisieren, verwenden wir die folgenden Symbole:

$A \to B$	Sicherheitsbegriff A impliziert B.
$A \leftarrow B$	B impliziert A.
$A \leftrightarrow B$	A ist äquivalent zu B.
$A \nrightarrow B$	A impliziert nicht B, d. h. es existiert ein Beispiel, das bzgl. A aber nicht bzgl. B sicher ist.
$A \nleftarrow B$	B impliziert nicht A.
$A \nleftrightarrow B$	A impliziert nicht B und B impliziert nicht A.

Man beachte, daß $\nleftrightarrow$ nicht das Gegenteil von $\leftrightarrow$ ist.

Schließlich verwenden wir noch zwei Symbole, um den Geltungsbereich der Beziehung zu modifizieren:

$\flat$	Die Aussage gilt für eine eingeschränkte Klasse von Protokollen (z. B. polynomiell-beschränkte).
$\aleph$	Der Beweis verwendet Komplexitätsannahmen.

Diese zusätzliche Angabe steht über dem den Typ der Relation spezifizierenden Pfeil (z. B. $\overset{\aleph}{\nrightarrow}$). Welche Komplexitätsannahmen und welche Einschränkung der Klasse von Protokollen verwendet wurden, geht aus dieser Kurzschreibweise nicht hervor, hierzu muß man die eigentlichen Ergebnisse im Hauptteil dieser Arbeit zu Rate ziehen. Bei negativen Ergebnissen ist die Klasse der Protokolle aber immer die der polynomiell-beschränkten, und bei positiven enthält sie immer mindestens alle polynomiell-beschränkten Protokolle.

Das Hinzufügen des Symbols $\aleph$ macht die Aussage immer schwächer. Bei $\flat$ hängt es davon ab, ob ein positives oder ein negatives Resultat vorliegt. So bedeutet $\overset{\flat}{\to}$, daß ein aus polynomiell-beschränkten Protokollen bestehendes trennendes Beispiel besteht, wohingegen $\nrightarrow$ einfach nur die Existenz eines irgendwie gearteten Gegenbeispiels darstellt und somit schwächer ist. Bei den positiven Aussagen jedoch ist $\flat$ eine Einschränkung der Gültigkeit.

Um die Details nachschlagen zu können, ist jede Beziehung mit einer Angabe der Sätze versehen, welche notwendig sind, um diese Beziehung zu folgern. Hier werden nur die Nummern der entsprechenden Sätze aufgeführt, da Sätze, Lemmata und Korollare in dieser Arbeit gemeinsam numeriert wurden.

Ein Beispiel für eine solche komprimierte Angabe eines Ergebnisses ist die folgende:

$$\mathrm{k}\blacksquare\downarrow\times \overset{\aleph\flat}{\nleftarrow} \mathrm{k}\triangleleft\downarrow : 2.23{,}5.10$$

Dies bedeutet, daß nach Theorem 2.23 und Satz 5.10 unter gewissen Komplexitätsannahmen (in diesem Falle die Existenz von TL-Puzzles) polynomiellbeschränkte Protokolle π und ρ existieren, so daß π *nicht* so sicher wie ρ ist bezüglich komplexitätstheoretischer allgemeiner Sicherheit mit Auxiliary input bzgl. der Ausgabe der Umgebung, wohl aber π so sicher wie ρ ist bezüglich komplexitätstheoretischer einfacher Komponierbarkeit mit Auxiliary input.

Da alle in dieser Arbeit betrachteten Varianten der perfekten Sicherheit äquivalent sind, haben wir eine weitere Abkürzung in die unten folgende Auflistung eingeführt. Jeden der Begriffe *perfekte allgemeine/spezielle Sicherheit oder einfache/allgemeine Komponierbarkeit mit/ohne Auxiliary input bzgl. der Ausgabe/der Sicht der Umgebung (mit Umgebungen mit Ein-Bit-Ausgabe)* ersetzen wir durch die Angabe *perfect* (die Begriffe sind äquivalent nach Satz 3.8 und Lemma A.1).

Die folgende Auflistung ist nach der linken Seite der Relationen sortiert. Zu jeder Aussage der Form $A \to B$ führen wir auch $B \leftarrow A$ auf, so daß es nicht nötig ist, sowohl nach A als auch nach B zu suchen (das gilt natürlich auch für die anderen Pfeile).

B.2. Perfekte Sicherheit und Komponierbarkeit (*perfect*)

$perfect \leftrightarrow perfect$: 3.8,A.1
$perfect \overset{b}{\leftarrow} s\blacksquare\downarrow\times$: A.6
$perfect \to s\blacksquare\downarrow\times$: A.6
$perfect \overset{b}{\leftarrow} s\blacksquare\downarrow\times 1$: A.10,A.1,A.6
$perfect \to s\blacksquare\downarrow\times 1$: A.6,A.1
$perfect \overset{Nb}{\leftrightarrow} s\blacksquare\downarrow\times 1P$: A.9,A.3,A.7
$perfect \overset{b}{\leftarrow} s\blacksquare\downarrow\otimes$: A.6
$perfect \to s\blacksquare\downarrow\otimes$: A.6
$perfect \overset{b}{\leftarrow} s\blacksquare\not\downarrow\times$: A.6
$perfect \to s\blacksquare\not\downarrow\times$: A.6
$perfect \overset{b}{\leftarrow} s\blacksquare\not\downarrow\times 1$: A.10,A.1,A.6
$perfect \to s\blacksquare\not\downarrow\times 1$: A.6,A.2,A.1
$perfect \overset{Nb}{\leftrightarrow} s\blacksquare\not\downarrow\times 1P$: A.10,A.3,A.7
$perfect \overset{b}{\leftarrow} s\blacksquare\not\downarrow\otimes$: A.6
$perfect \to s\blacksquare\not\downarrow\otimes$: A.6
$perfect \overset{b}{\leftarrow} s\square\downarrow\times$: A.6
$perfect \to s\square\downarrow\times$: A.6
$perfect \overset{b}{\leftarrow} s\square\downarrow\otimes$: A.6
$perfect \to s\square\downarrow\otimes$: A.6
$perfect \overset{b}{\leftarrow} s\square\not\downarrow\times$: A.6
$perfect \to s\square\not\downarrow\times$: A.6
$perfect \overset{b}{\leftarrow} s\square\not\downarrow\otimes$: A.6
$perfect \to s\square\not\downarrow\otimes$: A.6
$perfect \overset{b}{\leftarrow} s\blacktriangleleft\downarrow$: A.10,4.4,A.6
$perfect \to s\blacktriangleleft\downarrow$: A.6,4.4
$perfect \overset{b}{\leftarrow} s\blacktriangleleft\not\downarrow$: 2.29,A.6
$perfect \to s\blacktriangleleft\not\downarrow$: A.6,4.4
$perfect \overset{b}{\leftarrow} s\triangleleft\downarrow$: 2.23,A.6
$perfect \to s\triangleleft\downarrow$: A.6,2.23
$perfect \overset{b}{\leftarrow} s\triangleleft\not\downarrow$: 7.20,A.10,2.23,A.6
$perfect \to s\triangleleft\not\downarrow$: A.6,2.23
$perfect \overset{Nb}{\leftrightarrow} k\blacksquare\downarrow\times$: A.7
$perfect \overset{Nb}{\leftrightarrow} k\blacksquare\downarrow\times 1$: A.1,A.7
$perfect \overset{Nb}{\leftrightarrow} k\blacksquare\downarrow\otimes$: A.7
$perfect \overset{Nb}{\leftrightarrow} k\blacksquare\not\downarrow\times$: A.7
$perfect \overset{Nb}{\leftrightarrow} k\blacksquare\not\downarrow\times N$: A.9,A.2,A.7
$perfect \overset{Nb}{\leftrightarrow} k\blacksquare\not\downarrow\times 1$: A.10,A.1,A.7
$perfect \overset{Nb}{\leftrightarrow} k\blacksquare\not\downarrow\otimes$: A.7
$perfect \overset{Nb}{\leftrightarrow} k\blacksquare\not\downarrow\otimes N$: A.2,A.7
$perfect \overset{Nb}{\leftrightarrow} k\square\downarrow\times$: A.7
$perfect \overset{Nb}{\leftrightarrow} k\square\downarrow\otimes$: A.7
$perfect \overset{Nb}{\leftrightarrow} k\square\not\downarrow\times$: A.7
$perfect \overset{Nb}{\leftrightarrow} k\square\not\downarrow\otimes$: A.7
$perfect \overset{Nb}{\leftrightarrow} k\blacktriangleleft\downarrow$: 2.29,A.7
$perfect \overset{Nb}{\leftrightarrow} k\blacktriangleleft\not\downarrow$: 2.29,A.7
$perfect \overset{Nb}{\leftrightarrow} k\triangleleft\downarrow$: 2.23,A.7
$perfect \overset{Nb}{\leftrightarrow} k\triangleleft\not\downarrow$: 2.23,A.10,A.7

B.3. Statistische allgemeine Sicherheit (s■)

s■↓× $\overset{\flat}{\nrightarrow}$ *perfect* : A.6
s■↓× ← *perfect* : A.6
s■↓× ↔ s■↓×1 : A.1
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P : A.9,A.3,7.20,A.7
s■↓× ↛ s■↓⊗ : 3.4
s■↓× ← s■↓⊗ : A.9
s■↓× $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : 7.20
s■↓× ↔ s■↓× : A.2
s■↓× ↔ s■↓×1 : A.1,A.2
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P : A.10,A.3,7.20,A.7,A.9
s■↓× ↛ s■↓⊗ : 3.4
s■↓× ← s■↓⊗ : A.9,A.2
s■↓× $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : A.9,A.2,7.20
s■↓× ↔ s□↓× : 4.5
s■↓× ↛ s□↓⊗ : 3.4
s■↓× ← s□↓⊗ : A.9,2.23,4.5
s■↓× $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : A.10,7.20
s■↓× ↚ s□↓× : A.2,5.1
s■↓× $\overset{\flat}{\leftrightarrow}$ s□↓× : 7.20
s■↓× → s□↓× : A.2,A.4
s■↓× ↮ s□↓⊗ : A.2,5.1,3.4
s■↓× $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : 7.20,A.10
s■↓× ↔ s◀↓ : 2.29,4.5
s■↓× ↚ s◀↓× : 4.4,A.2,5.1
s■↓× $\overset{\flat}{\leftrightarrow}$ s◀↓× : 7.20,2.29
s■↓× → s◀↓× : A.2,2.29
s■↓× ↔ s◁↓ : 4.5,2.23
s■↓× ↚ s◁↓× : A.2,2,23,5.1
s■↓× $\overset{\flat}{\leftrightarrow}$ s◁↓× : 7.20,2.23
s■↓× → s◁↓× : A.2,A.4,2.23
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓× : A.7
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 : A.1,A.7
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : 7.20,A.7,A.9
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓× : 7.20,A.7,A.9,A.10
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×N : A.9,A.2,7.20,A.7
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 : A.10,A.1,7.20,A.7,A.9
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : 7.20,A.7,A.9,A.10
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗N : A.2,7.20,A.7,A.9
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× : 7.20,A.7,A.10
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : 7.20,A.7,A.10,A.9
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× : 7.20,A.7,A.10
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : 7.20,A.7,A.10
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k◀↓ : 2.29,A.7,7.20,A.10
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k◀↓ : 2.29,7.20,A.7,A.10
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k◁↓ : 7.20,2.23,A.7,A.10
s■↓× $\overset{\aleph\flat}{\leftrightarrow}$ k◁↓ : 2.23,7.20,A.7,A.10
s■↓×1 $\overset{\flat}{\nrightarrow}$ *perfect* : A.10,A.1,A.6
s■↓×1 ← *perfect* : A.6,A.1
s■↓×1 ↔ s■↓× : A.1
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P : A.1,A.9,A.3,7.20,A.7,A.10
s■↓×1 ↛ s■↓⊗ : A.1,3.4
s■↓×1 ← s■↓⊗ : A.9,A.1
s■↓×1 $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : A.10,A.1,7.20
s■↓×1 ↔ s■↓× : A.2,A.1
s■↓×1 ↔ s■↓×1 : A.1,A.2
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P : A.1,A.10,A.3,7.20,A.7,A.9
s■↓×1 ↛ s■↓⊗ : A.1,3.4
s■↓×1 ← s■↓⊗ : A.2,A.9,A.1
s■↓×1 $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : A.10,7.20,A.1
s■↓×1 ↔ s□↓× : 4.5,A.1
s■↓×1 ↛ s□↓⊗ : A.1,3.4
s■↓×1 ← s□↓⊗ : 4.5,A.9,A.1
s■↓×1 $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : A.9,7.20,A.10,A.1
s■↓×1 ↚ s□↓× : A.1,A.2,5.1
s■↓×1 $\overset{\flat}{\leftrightarrow}$ s□↓× : 7.20,A.10,A.1
s■↓×1 → s□↓× : A.1,A.2,A.4
s■↓×1 ↮ s□↓⊗ : A.1,3.4,A.2,5.1
s■↓×1 $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : A.10,7.20,A.1
s■↓×1 ↔ s◀↓ : A.1,2.29,4.5
s■↓×1 ↚ s◀↓× : A.1,4.4,A.2,5.1
s■↓×1 $\overset{\flat}{\leftrightarrow}$ s◀↓× : 2.29,7.20,A.10,A.1,A.2
s■↓×1 → s◀↓× : A.1,A.2,2.29
s■↓×1 ↔ s◁↓ : 2.23,4.5,A.1
s■↓×1 ↚ s◁↓× : A.1,A.2,2.23,5.1
s■↓×1 $\overset{\flat}{\leftrightarrow}$ s◁↓× : 2.23,7.20,A.10,A.1
s■↓×1 → s◁↓× : A.1,A.2,A.4,2.23
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■↓× : 7.20,A.10,A.1,A.7
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 : A.1,A.7,7.20,A.10
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : A.1,A.9,A.7,7.20,A.10
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■↓× : 7.20,A.10,A.1,A.9,A.7
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×N : A.1,A.9,A.2,7.20,A.7,A.10
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 : A.1,A.10,7.20,A.7,A.9
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : 7.20,A.10,A.1,A.9,A.7
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗N : A.1,A.2,7.20,A.7,A.10,A.9
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× : A.1,7.20,A.10,A.7

s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : A.1,7.20,A.7,A.10,A.9
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k□𝒻× : A.1,7.20,A.7,A.10
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k□𝒻⊗ : A.1,7.20,A.10,A.7
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k◂↓ : A.1,2.29,A.7,7.20,A.10
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k◂𝒻 : A.1,7.20,A.10,2.29,A.7
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k◁↓ : A.1,7.20,2.23,A.7,A.10
s■↓×1 $\overset{\aleph\flat}{\leftrightarrow}$ k◁𝒻 : A.1,7.20,A.10,2.23,A.7
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ *perfect* : A.3,A.7,A.9
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s■↓× : A.9,A.3,7.20,A.7
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1 : A.3,7.20,A.10,A.1,A.7,A.9
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s■↓⊗ : A.9,A.3,A.7,7.20
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s■𝒻× : A.3,7.20,A.7,A.9,A.10
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s■𝒻×1 : A.3,A.10,A.1,7.20,A.7,A.9
s■↓×1P $\overset{\aleph\flat}{\leftarrow}$ s■𝒻×1P : A.3,A.10,A.8
s■↓×1P → s■𝒻×1P : A.3,A.5
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s■𝒻⊗ : A.3,7.20,A.7,A.9,A.10
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s□↓× : A.3,7.20,A.7,A.9,A.10
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s□↓⊗ : A.3,A.6,A.7,A.9,A.10,7.20
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s□𝒻× : A.9,A.3,7.20,A.10,A.7
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s□𝒻⊗ : A.9,A.3,7.20,A.10,A.7
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s◂↓ : A.3,A.10,4.4,A.6,A.7,2.29,A.9,7.20
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s◂𝒻 : A.3,7.20,4.4,A.7,2.29,A.9,A.10
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s◁↓ : A.3,A.10,2.23,A.6,A.7,A.9,7.20
s■↓×1P $\overset{\aleph\flat}{\leftrightarrow}$ s◁𝒻 : A.3,2.23,7.20,A.10,A.7,A.9
s■↓×1P ↔ k■↓× : A.3
s■↓×1P ↔ k■↓×1 : A.1,A.3
s■↓×1P $\overset{\flat}{\leftrightarrow}$ k■↓⊗ : A.10,A.3
s■↓×1P $\overset{\aleph\flat}{\leftarrow}$ k■𝒻× : A.3,A.8
s■↓×1P → k■𝒻× : A.3,A.5
s■↓×1P ↔ k■𝒻×N : A.2,A.3
s■↓×1P $\overset{\aleph\flat}{\leftarrow}$ k■𝒻×1 : A.3,A.10,A.1,A.8
s■↓×1P → k■𝒻×1 : A.3,A.5,A.1
s■↓×1P $\overset{\aleph\flat}{\leftarrow}$ k■𝒻⊗ : A.3,A.8
s■↓×1P $\overset{\flat}{\rightarrow}$ k■𝒻⊗ : A.3,A.5,A.10
s■↓×1P $\overset{\flat}{\leftrightarrow}$ k■𝒻⊗N : A.2,A.10,A.3
s■↓×1P ↞ k□↓× : A.3,5.2
s■↓×1P $\overset{\aleph\flat}{\leftarrow}$ k□↓× : A.3,5.10
s■↓×1P → k□↓× : A.3,A.4
s■↓×1P ↞ k□↓⊗ : A.3,5.2
s■↓×1P $\overset{\aleph\flat}{\leftarrow}$ k□↓⊗ : A.3,5.10
s■↓×1P $\overset{\flat}{\rightarrow}$ k□↓⊗ : A.3,2.29,A.10
s■↓×1P ↞ k□𝒻× : A.3,5.2
s■↓×1P $\overset{\aleph\flat}{\leftarrow}$ k□𝒻× : A.3,5.10
s■↓×1P → k□𝒻× : A.3,A.5,A.4
s■↓×1P ↞ k□𝒻⊗ : A.3,5.2
s■↓×1P $\overset{\aleph\flat}{\leftarrow}$ k□𝒻⊗ : A.3,5.10
s■↓×1P $\overset{\flat}{\rightarrow}$ k□𝒻⊗ : A.3,2.29,A.10,A.5
s■↓×1P $\overset{\aleph\flat}{\leftarrow}$ k◂↓ : A.3,6.2
s■↓×1P $\overset{\flat}{\rightarrow}$ k◂↓ : A.3,2.29
s■↓×1P $\overset{\aleph\flat}{\leftarrow}$ k◂𝒻 : A.3,6.2
s■↓×1P $\overset{\flat}{\rightarrow}$ k◂𝒻 : A.3,A.5,2.29
s■↓×1P $\overset{\aleph\flat}{\leftarrow}$ k◁↓ : A.3,2.23,5.10
s■↓×1P $\overset{\flat}{\rightarrow}$ k◁↓ : A.3,2.29,2.23
s■↓×1P $\overset{\aleph\flat}{\leftarrow}$ k◁𝒻 : A.3,2.23,5.10
s■↓×1P $\overset{\flat}{\rightarrow}$ k◁𝒻 : A.3,A.5,2.29,2.23
s■↓⊗ $\overset{\flat}{\rightarrow}$ *perfect* : A.6
s■↓⊗ ← *perfect* : A.6
s■↓⊗ ↞ s■↓× : 3.4
s■↓⊗ $\overset{\flat}{\leftrightarrow}$ s■↓× : A.10
s■↓⊗ → s■↓× : A.9
s■↓⊗ ↞ s■↓×1 : A.1,3.4
s■↓⊗ $\overset{\flat}{\leftrightarrow}$ s■↓×1 : A.1,7.20,A.10
s■↓⊗ → s■↓×1 : A.9,A.1
s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P : A.9,A.3,A.7,7.20
s■↓⊗ ↞ s■𝒻× : 3.4
s■↓⊗ $\overset{\flat}{\leftrightarrow}$ s■𝒻× : 7.20,A.10
s■↓⊗ → s■𝒻× : A.5,A.9
s■↓⊗ ↞ s■𝒻×1 : A.1,3.4
s■↓⊗ $\overset{\flat}{\leftrightarrow}$ s■𝒻×1 : 7.20,A.10,A.1
s■↓⊗ → s■𝒻×1 : A.9,A.2,A.1
s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ s■𝒻×1P : A.10,A.3,7.20,A.7
s■↓⊗ ↔ s■𝒻⊗ : A.2
s■↓⊗ ↞ s□↓× : 3.4
s■↓⊗ $\overset{\flat}{\leftrightarrow}$ s□↓× : A.10,7.20
s■↓⊗ → s□↓× : 4.5,A.9
s■↓⊗ ↔ s□↓⊗ : 4.5
s■↓⊗ ↞ s□𝒻× : 3.4
s■↓⊗ $\overset{\flat}{\leftrightarrow}$ s□𝒻× : 7.20,A.10
s■↓⊗ → s□𝒻× : A.5,A.4,A.9
s■↓⊗ ↞ s□𝒻⊗ : A.5,5.1
s■↓⊗ $\overset{\flat}{\leftrightarrow}$ s□𝒻⊗ : A.10,7.20,A.4
s■↓⊗ → s□𝒻⊗ : A.5,A.4
s■↓⊗ ↞ s◂↓ : 4.4,3.4
s■↓⊗ $\overset{\flat}{\leftrightarrow}$ s◂↓ : A.10,2.29,7.20
s■↓⊗ → s◂↓ : A.9,2.29

s■↓⊗ $\nleftarrow$ s◀ $\not\downarrow$: 2.29,3.4

s■↓⊗ $\overset{\flat}{\leftrightarrow}$ s◀ $\not\downarrow$: 2.29,7.20,A.4,A.10,4.4
s■↓⊗ $\rightarrow$ s◀ $\not\downarrow$: A.9,A.2,2.29
s■↓⊗ $\nleftarrow$ s◁↓ : 2.23,3.4

s■↓⊗ $\overset{\flat}{\leftrightarrow}$ s◁↓ : 2.23,7.20,A.10
s■↓⊗ $\rightarrow$ s◁↓ : A.4,A.9,2.23
s■↓⊗ $\nleftarrow$ s◁ $\not\downarrow$: 2.23,3.4

s■↓⊗ $\overset{\flat}{\leftrightarrow}$ s◁ $\not\downarrow$: 7.20,A.10,2.23
s■↓⊗ $\rightarrow$ s◁ $\not\downarrow$: A.9,A.2,A.4,2.23

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓× : A.9,A.7,7.20

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 : A.1,A.9,A.7,7.20

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : A.7

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■ $\not\downarrow$× : A.10,7.20,A.7

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■ $\not\downarrow$×N : A.9,A.2,A.7,7.20

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■ $\not\downarrow$×1 : A.10,A.1,7.20,A.7

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■ $\not\downarrow$⊗ : 7.20,A.7,A.10

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■ $\not\downarrow$⊗N : A.2,A.7

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× : A.9,7.20,A.7,A.6

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : 7.20,A.7,A.6

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k□ $\not\downarrow$× : 7.20,A.10,A.7,A.6

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k□ $\not\downarrow$⊗ : 7.20,A.7,A.6

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k◀↓ : 2.29,A.9,A.7,A.6

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k◀ $\not\downarrow$: 2.29,A.10,7.20,A.7,A.6

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k◁↓ : A.10,7.20,2.23,A.7,A.6

s■↓⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k◁ $\not\downarrow$: 2.23,7.20,A.10,A.7,A.6

s■ $\not\downarrow$× $\overset{\flat}{\nrightarrow}$ *perfect* : A.6
s■ $\not\downarrow$× $\leftarrow$ *perfect* : A.6
s■ $\not\downarrow$× $\leftrightarrow$ s■↓× : A.2
s■ $\not\downarrow$× $\leftrightarrow$ s■↓×1 : A.1,A.2

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P : A.9,A.3,7.20,A.10,A.7
s■ $\not\downarrow$× $\nrightarrow$ s■↓⊗ : 3.4
s■ $\not\downarrow$× $\leftarrow$ s■↓⊗ : A.5,A.9

s■ $\not\downarrow$× $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : A.10,7.20
s■ $\not\downarrow$× $\leftrightarrow$ s■ $\not\downarrow$×1 : A.1

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ s■ $\not\downarrow$×1P : A.10,A.3,7.20,A.7
s■ $\not\downarrow$× $\nrightarrow$ s■ $\not\downarrow$⊗ : 3.4
s■ $\not\downarrow$× $\leftarrow$ s■ $\not\downarrow$⊗ : A.9

s■ $\not\downarrow$× $\overset{\flat}{\leftrightarrow}$ s■ $\not\downarrow$⊗ : 7.20
s■ $\not\downarrow$× $\leftrightarrow$ s□↓× : 4.5,A.2
s■ $\not\downarrow$× $\nrightarrow$ s□↓⊗ : 3.4
s■ $\not\downarrow$× $\leftarrow$ s□↓⊗ : 4.5,A.5,A.9

s■ $\not\downarrow$× $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : A.10,7.20
s■ $\not\downarrow$× $\nleftarrow$ s□ $\not\downarrow$× : 5.1

s■ $\not\downarrow$× $\overset{\flat}{\leftrightarrow}$ s□ $\not\downarrow$× : 7.20
s■ $\not\downarrow$× $\rightarrow$ s□ $\not\downarrow$× : A.4
s■ $\not\downarrow$× $\nleftrightarrow$ s□ $\not\downarrow$⊗ : 5.1,3.4

s■ $\not\downarrow$× $\overset{\flat}{\leftrightarrow}$ s□ $\not\downarrow$⊗ : 7.20,A.10
s■ $\not\downarrow$× $\leftrightarrow$ s◀↓ : A.2,2.29,4.5
s■ $\not\downarrow$× $\nleftarrow$ s◀ $\not\downarrow$: 4.4,5.1

s■ $\not\downarrow$× $\overset{\flat}{\leftrightarrow}$ s◀ $\not\downarrow$: 2.29,7.20
s■ $\not\downarrow$× $\rightarrow$ s◀ $\not\downarrow$: 2.29
s■ $\not\downarrow$× $\leftrightarrow$ s◁↓ : A.2,4.5,2.23
s■ $\not\downarrow$× $\nleftarrow$ s◁ $\not\downarrow$: 2.23,5.1

s■ $\not\downarrow$× $\overset{\flat}{\leftrightarrow}$ s◁ $\not\downarrow$: 2.23,7.20
s■ $\not\downarrow$× $\rightarrow$ s◁ $\not\downarrow$: A.4,2.23

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓× : 7.20,A.10,A.9,A.7

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 : A.1,7.20,A.10,A.9,A.7

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : 7.20,A.10,A.7,A.9

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k■ $\not\downarrow$× : A.7

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k■ $\not\downarrow$×N : 7.20,A.9,A.2,A.10,A.7

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k■ $\not\downarrow$×1 : A.10,A.1,7.20,A.7

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k■ $\not\downarrow$⊗ : 7.20,A.7,A.10

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k■ $\not\downarrow$⊗N : 7.20,A.10,A.2,A.7,A.9

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× : 7.20,A.9,A.7,A.10

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : 7.20,A.7,A.10,A.9

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k□ $\not\downarrow$× : 7.20,A.7,A.10

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k□ $\not\downarrow$⊗ : 7.20,A.7,A.10

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k◀↓ : 7.20,A.10,2.29,A.9,A.7

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k◀ $\not\downarrow$: 2.29,A.7,A.10,7.20

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k◁↓ : 7.20,A.10,2.23,A.7

s■ $\not\downarrow$× $\overset{\aleph\flat}{\leftrightarrow}$ k◁ $\not\downarrow$: 2.23,7.20,A.7,A.10

s■ $\not\downarrow$×1 $\overset{\flat}{\nrightarrow}$ *perfect* : A.10,A.1,A.6
s■ $\not\downarrow$×1 $\leftarrow$ *perfect* : A.6,A.2,A.1
s■ $\not\downarrow$×1 $\leftrightarrow$ s■↓× : A.2,A.1
s■ $\not\downarrow$×1 $\leftrightarrow$ s■↓×1 : A.1,A.2

s■ $\not\downarrow$×1 $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P : A.1,A.9,A.3,7.20,A.10,A.7
s■ $\not\downarrow$×1 $\nrightarrow$ s■↓⊗ : A.1,3.4
s■ $\not\downarrow$×1 $\leftarrow$ s■↓⊗ : A.9,A.2,A.1

s■ $\not\downarrow$×1 $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : 7.20,A.10,A.1
s■ $\not\downarrow$×1 $\leftrightarrow$ s■ $\not\downarrow$× : A.1

s■ $\not\downarrow$×1 $\overset{\aleph\flat}{\leftrightarrow}$ s■ $\not\downarrow$×1P : A.1,A.10,A.3,7.20,A.7
s■ $\not\downarrow$×1 $\nrightarrow$ s■ $\not\downarrow$⊗ : A.1,3.4
s■ $\not\downarrow$×1 $\leftarrow$ s■ $\not\downarrow$⊗ : A.9,A.1

s■ $\not\downarrow$×1 $\overset{\flat}{\leftrightarrow}$ s■ $\not\downarrow$⊗ : A.10,A.1,7.20
s■ $\not\downarrow$×1 $\leftrightarrow$ s□↓× : 4.5,A.2,A.1
s■ $\not\downarrow$×1 $\nrightarrow$ s□↓⊗ : A.1,3.4
s■ $\not\downarrow$×1 $\leftarrow$ s□↓⊗ : 4.5,A.9,A.2,A.1

s■ $\not\downarrow$×1 $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : A.9,7.20,A.1
s■ $\not\downarrow$×1 $\nleftarrow$ s□ $\not\downarrow$× : A.1,5.1

s■ $\not\downarrow$×1 $\overset{\flat}{\leftrightarrow}$ s□ $\not\downarrow$× : 7.20,A.1
s■ $\not\downarrow$×1 $\rightarrow$ s□ $\not\downarrow$× : A.1,A.4
s■ $\not\downarrow$×1 $\nleftrightarrow$ s□ $\not\downarrow$⊗ : A.1,3.4,5.1

s■ ⊀×1 $\overset{\flat}{\leftrightarrow}$ s□ ⊀⊗ : A.10,7.20,A.1
s■ ⊀×1 ↔ s◂↓ : 2.29,4.5,A.2,A.1
s■ ⊀×1 $\nleftrightarrow$ s◂ ⊀ : A.1,4.4,5.1

s■ ⊀×1 $\overset{\flat}{\leftrightarrow}$ s◂ ⊀ : 2.29,7.20,A.1
s■ ⊀×1 → s◂ ⊀ : A.1,2.29
s■ ⊀×1 ↔ s◁↓ : 2.23,4.5,A.2,A.1
s■ ⊀×1 $\nleftrightarrow$ s◁ ⊀ : A.1,2.23,5.1

s■ ⊀×1 $\overset{\flat}{\leftrightarrow}$ s◁ ⊀ : 2.23,7.20,A.1
s■ ⊀×1 → s◁ ⊀ : A.1,A.4,2.23

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■↓× : A.1,7.20,A.7,A.10,A.9

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 : A.1,7.20,A.10,A.9,A.7

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : A.1,7.20,A.9,A.7,A.10

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■ ⊀× : A.1,A.7,A.10,7.20

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■ ⊀×N :
A.1,7.20,A.9,A.2,A.10,A.7

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■ ⊀×1 : A.1,A.10,7.20,A.7

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■ ⊀⊗ : A.1,A.10,A.7,7.20

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k■ ⊀⊗N : A.1,7.20,A.10,A.2,A.7

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× :
A.1,7.20,A.9,A.7,A.10,A.6

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : A.1,7.20,A.7,A.10,A.6

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k□ ⊀× : A.1,7.20,A.7,A.10,A.6

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k□ ⊀⊗ : 7.20,A.10,A.1,A.6,A.7

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k◂↓ :
A.1,7.20,A.10,2.29,A.9,A.7,A.6

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k◂ ⊀ : 7.20,A.10,A.1,A.6,2.29,A.7

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k◁↓ : A.1,7.20,A.10,2.23,A.7,A.6

s■ ⊀×1 $\overset{\aleph\flat}{\leftrightarrow}$ k◁ ⊀ : 2.23,7.20,A.10,A.1,A.6,A.7

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ *perfect* : A.3,A.7,A.10

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s■↓× : A.3,A.9,A.10,7.20,A.7

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1 :
A.3,7.20,A.10,A.1,A.9,A.7

s■ ⊀×1P $\overset{\aleph\flat}{\nrightarrow}$ s■↓×1P : A.3,A.10,A.8
s■ ⊀×1P ← s■↓×1P : A.3,A.5

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s■↓⊗ : A.10,A.3,7.20,A.7

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s■ ⊀× : A.3,A.7,A.10,7.20

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s■ ⊀×1 : A.3,A.10,A.1,7.20,A.7

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s■ ⊀⊗ : A.10,A.3,A.7,7.20

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s□↓× : A.3,7.20,A.9,A.10,A.7

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s□↓⊗ : A.3,A.6,A.7,A.10,7.20

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s□ ⊀× : A.10,A.3,7.20,A.7

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s□ ⊀⊗ : A.3,7.20,A.7,A.10

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s◂↓ :
2.29,A.10,A.3,7.20,A.7,4.4,A.6

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s◂ ⊀ :
A.3,7.20,A.10,4.4,A.6,A.7,2.29

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s◁↓ :
A.3,A.10,2.23,A.6,A.7,7.20

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ s◁ ⊀ : A.3,2.23,7.20,A.7,A.10

s■ ⊀×1P $\overset{\aleph\flat}{\nrightarrow}$ k■↓× : A.10,A.3,A.8
s■ ⊀×1P ← k■↓× : A.5,A.3

s■ ⊀×1P $\overset{\aleph\flat}{\nrightarrow}$ k■↓×1 : A.1,A.10,A.3,A.8
s■ ⊀×1P ← k■↓×1 : A.1,A.5,A.3

s■ ⊀×1P $\overset{\aleph\flat}{\nrightarrow}$ k■↓⊗ : A.10,A.3,A.8

s■ ⊀×1P $\overset{\flat}{\leftarrow}$ k■↓⊗ : A.5,A.10,A.3
s■ ⊀×1P ↔ k■ ⊀× : A.3

s■ ⊀×1P $\overset{\aleph\flat}{\nrightarrow}$ k■ ⊀×N : A.2,A.10,A.3,A.8
s■ ⊀×1P ← k■ ⊀×N : A.2,A.5,A.3
s■ ⊀×1P ↔ k■ ⊀×1 : A.1,A.3

s■ ⊀×1P $\overset{\flat}{\leftrightarrow}$ k■ ⊀⊗ : A.10,A.3

s■ ⊀×1P $\overset{\aleph\flat}{\nrightarrow}$ k■ ⊀⊗N : A.2,A.10,A.3,A.8

s■ ⊀×1P $\overset{\flat}{\leftarrow}$ k■ ⊀⊗N : A.2,A.5,A.10,A.3
s■ ⊀×1P $\nleftrightarrow$ k□↓× : A.3,5.2

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× : A.3,5.10,A.10,A.8
s■ ⊀×1P $\nleftrightarrow$ k□↓⊗ : A.3,5.2

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : A.10,A.3,A.8,5.10
s■ ⊀×1P $\nleftrightarrow$ k□ ⊀× : A.3,5.2

s■ ⊀×1P $\overset{\aleph\flat}{\nleftrightarrow}$ k□ ⊀× : A.3,5.10
s■ ⊀×1P → k□ ⊀× : A.3,A.4
s■ ⊀×1P $\nleftrightarrow$ k□ ⊀⊗ : A.3,5.2

s■ ⊀×1P $\overset{\aleph\flat}{\nleftrightarrow}$ k□ ⊀⊗ : A.3,5.10

s■ ⊀×1P $\overset{\flat}{\rightarrow}$ k□ ⊀⊗ : A.3,A.10,A.4

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ k◂↓ : A.3,6.2,A.10,2.29,A.8

s■ ⊀×1P $\overset{\aleph\flat}{\nleftrightarrow}$ k◂ ⊀ : A.3,6.2

s■ ⊀×1P $\overset{\flat}{\rightarrow}$ k◂ ⊀ : A.3,2.29

s■ ⊀×1P $\overset{\aleph\flat}{\leftrightarrow}$ k◁↓ : A.3,2.23,5.10,A.10,A.8

s■ ⊀×1P $\overset{\aleph\flat}{\nleftrightarrow}$ k◁ ⊀ : A.3,2.23,5.10

s■ ⊀×1P $\overset{\flat}{\rightarrow}$ k◁ ⊀ : A.3,2.29,2.23

s■ ⊀⊗ $\overset{\flat}{\nrightarrow}$ *perfect* : A.6
s■ ⊀⊗ ← *perfect* : A.6
s■ ⊀⊗ $\nleftrightarrow$ s■↓× : 3.4

s■ ⊀⊗ $\overset{\flat}{\leftrightarrow}$ s■↓× : 7.20,A.10
s■ ⊀⊗ → s■↓× : A.9,A.2
s■ ⊀⊗ $\nleftrightarrow$ s■↓×1 : A.1,3.4

s■ ⊀⊗ $\overset{\flat}{\leftrightarrow}$ s■↓×1 : A.10,7.20,A.1
s■ ⊀⊗ → s■↓×1 : A.2,A.9,A.1

s■ ⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P : A.9,A.3,A.10,7.20,A.7
s■ ⊀⊗ ↔ s■↓⊗ : A.2
s■ ⊀⊗ $\nleftrightarrow$ s■ ⊀× : 3.4

s■ ⊀⊗ $\overset{\flat}{\leftrightarrow}$ s■ ⊀× : A.10

s■⊬⊗ → s■⊬× : A.9
s■⊬⊗ ↚ s■⊬×1 : A.1,3.4
s■⊬⊗ $\overset{\flat}{\leftrightarrow}$ s■⊬×1 : A.10,A.1,7.20
s■⊬⊗ → s■⊬×1 : A.9,A.1
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ s■⊬×1P : A.10,A.3,A.7,7.20
s■⊬⊗ ↚ s□↓× : 3.4
s■⊬⊗ $\overset{\flat}{\leftrightarrow}$ s□↓× : 7.20,A.4,A.10
s■⊬⊗ → s□↓× : A.2,4.5,A.9
s■⊬⊗ ↔ s□↓⊗ : 4.5,A.5,A.2,A.4
s■⊬⊗ ↚ s□⊬× : 5.1
s■⊬⊗ $\overset{\flat}{\leftrightarrow}$ s□⊬× : A.10,7.20
s■⊬⊗ → s□⊬× : A.4,A.9
s■⊬⊗ ↚ s□⊬⊗ : 5.1
s■⊬⊗ $\overset{\flat}{\leftrightarrow}$ s□⊬⊗ : 7.20
s■⊬⊗ → s□⊬⊗ : A.4
s■⊬⊗ ↚ s◄↓ : 4.4,3.4
s■⊬⊗ $\overset{\flat}{\leftrightarrow}$ s◄↓ : 2.29,7.20,A.10,4.4
s■⊬⊗ → s◄↓ : A.2,A.9,2.29
s■⊬⊗ ↚ s◄⊬ : 2.29,3.4
s■⊬⊗ $\overset{\flat}{\leftrightarrow}$ s◄⊬ : 2.29,7.20,A.4,A.10,4.4
s■⊬⊗ → s◄⊬ : A.9,2.29
s■⊬⊗ ↚ s◁↓ : 2.23,3.4
s■⊬⊗ $\overset{\flat}{\leftrightarrow}$ s◁↓ : 7.20,A.10,2.23
s■⊬⊗ → s◁↓ : A.2,A.4,A.9,2.23
s■⊬⊗ ↚ s◁⊬ : 2.23,5.1
s■⊬⊗ $\overset{\flat}{\leftrightarrow}$ s◁⊬ : 7.20,A.10,2.23
s■⊬⊗ → s◁⊬ : A.9,A.4,2.23
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓× : A.10,7.20,A.9,A.7
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 : A.1,A.10,7.20,A.9,A.7
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : A.10,7.20,A.7
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■⊬× : A.10,A.7,7.20
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■⊬×N : A.9,A.2,A.10,7.20,A.7
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■⊬×1 : A.10,A.1,A.7,7.20
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■⊬⊗ : A.7
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■⊬⊗N : A.10,7.20,A.2,A.7
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× : A.10,7.20,A.9,A.7
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : A.10,7.20,A.7,A.9
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k□⊬× : A.10,7.20,A.7
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k□⊬⊗ : 7.20,A.7
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k◄↓ : A.10,7.20,2.29,A.9,A.7
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k◄⊬ : 2.29,A.10,A.7,7.20
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k◁↓ : A.10,7.20,2.23,A.7
s■⊬⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k◁⊬ : 2.23,A.10,7.20,A.7

B.4. Statistische spezielle Sicherheit (s□)

s□↓× $\overset{\flat}{\nrightarrow}$ *perfect* : A.6
s□↓× ← *perfect* : A.6
s□↓× ↔ s■↓× : 4.5
s□↓× ↔ s■↓×1 : A.1,4.5
s□↓× $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P : A.9,A.3,7.20,A.10,A.7
s□↓× ↛ s■↓⊗ : 3.4
s□↓× ← s■↓⊗ : 4.5,A.9
s□↓× $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : A.10,7.20
s□↓× ↔ s■⊬× : A.2,4.5
s□↓× ↔ s■⊬×1 : 4.5,A.2,A.1
s□↓× $\overset{\aleph\flat}{\leftrightarrow}$ s■⊬×1P : A.10,A.3,7.20,A.7,A.9
s□↓× ↛ s■⊬⊗ : 3.4
s□↓× ← s■⊬⊗ : A.2,4.5,A.9
s□↓× $\overset{\flat}{\leftrightarrow}$ s■⊬⊗ : A.4,A.10,7.20
s□↓× ↛ s□↓⊗ : 3.4
s□↓× ← s□↓⊗ : A.9
s□↓× $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : 7.20
s□↓× ↚ s□⊬× : 4.5,A.2,5.1
s□↓× $\overset{\flat}{\leftrightarrow}$ s□⊬× : 7.20
s□↓× → s□⊬× : A.5
s□↓× ↛ s□⊬⊗ : 3.4,4.5,A.2,5.1
s□↓× $\overset{\flat}{\leftrightarrow}$ s□⊬⊗ : 7.20,A.10
s□↓× ↔ s◄↓ : 2.29,4.4
s□↓× ↚ s◄⊬ : 4.5,4.4,A.2,5.1
s□↓× $\overset{\flat}{\leftrightarrow}$ s◄⊬ : 2.29,7.20,4.4
s□↓× → s◄⊬ : A.5,4.4
s□↓× ↔ s◁↓ : 2.23
s□↓× ↚ s◁⊬ : 4.5,A.2,2.23,5.1
s□↓× $\overset{\flat}{\leftrightarrow}$ s◁⊬ : 7.20,2.23
s□↓× → s◁⊬ : A.5,2.23
s□↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓× : 7.20,A.7,A.10,A.9
s□↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 : A.1,7.20,A.10,A.9,A.7
s□↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : A.10,7.20,A.7
s□↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■⊬× : 7.20,A.10,A.7,A.9
s□↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■⊬×N : 7.20,A.9,A.2,A.10,A.7
s□↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■⊬×1 : 7.20,A.10,A.1,A.7,A.9
s□↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■⊬⊗ : 7.20,A.7,A.9,A.10
s□↓× $\overset{\aleph\flat}{\leftrightarrow}$ k■⊬⊗N : 7.20,A.10,A.2,A.7
s□↓× $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× : A.7
s□↓× $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : 7.20,A.7,A.9
s□↓× $\overset{\aleph\flat}{\leftrightarrow}$ k□⊬× : 7.20,A.7,A.10,A.6

s□↓× $\overset{\aleph_0}{\leftrightarrow}$ k□⫽⊗ : 7.20,A.7,A.10,A.6

s□↓× $\overset{\aleph_0}{\leftrightarrow}$ k◄↓ : 7.20,A.10,2.29,A.9,A.7

s□↓× $\overset{\aleph_0}{\leftrightarrow}$ k◄⫽ : 7.20,2.29,A.10,A.7,A.6

s□↓× $\overset{\aleph_0}{\leftrightarrow}$ k◁↓ : 2.23,A.7

s□↓× $\overset{\aleph_0}{\leftrightarrow}$ k◁⫽ : 2.23,7.20,A.7,A.10,A.6

s□↓⊗ $\overset{\flat}{\rightarrow}$ *perfect* : A.6

s□↓⊗ ← *perfect* : A.6

s□↓⊗ $\nleftarrow$ s■↓× : 3.4

s□↓⊗ $\overset{\flat}{\leftrightarrow}$ s■↓× : 7.20,A.10

s□↓⊗ → s■↓× : A.9,2.23,4.5

s□↓⊗ $\nleftarrow$ s■↓×1 : A.1,3.4

s□↓⊗ $\overset{\flat}{\leftrightarrow}$ s■↓×1 : A.1,7.20,A.9,A.10

s□↓⊗ → s■↓×1 : 4.5,A.9,A.1

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ s■↓×1P : A.9,A.3,A.10,7.20,A.7,A.6

s□↓⊗ ↔ s■↓⊗ : 4.5

s□↓⊗ $\nleftarrow$ s■⫽× : 3.4

s□↓⊗ $\overset{\flat}{\leftrightarrow}$ s■⫽× : 7.20,A.10

s□↓⊗ → s■⫽× : 4.5,A.5,A.9

s□↓⊗ $\nleftarrow$ s■⫽×1 : A.1,3.4

s□↓⊗ $\overset{\flat}{\leftrightarrow}$ s■⫽×1 : A.1,7.20,A.9

s□↓⊗ → s■⫽×1 : 4.5,A.9,A.2,A.1

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ s■⫽×1P : A.10,A.3,7.20,A.7,A.6

s□↓⊗ ↔ s■⫽⊗ : A.2,A.4,4.5,A.5

s□↓⊗ $\nleftarrow$ s□↓× : 3.4

s□↓⊗ $\overset{\flat}{\leftrightarrow}$ s□↓× : A.10

s□↓⊗ → s□↓× : A.9

s□↓⊗ $\nleftarrow$ s□⫽× : 3.4

s□↓⊗ $\overset{\flat}{\leftrightarrow}$ s□⫽× : 7.20,A.10

s□↓⊗ → s□⫽× : A.9,A.5

s□↓⊗ $\nleftarrow$ s□⫽⊗ : 4.5,A.5,5.1

s□↓⊗ $\overset{\flat}{\leftrightarrow}$ s□⫽⊗ : 7.20

s□↓⊗ → s□⫽⊗ : A.5

s□↓⊗ $\nleftarrow$ s◄↓ : 4.4,3.4

s□↓⊗ $\overset{\flat}{\leftrightarrow}$ s◄↓ : A.10,4.4,2.29,7.20

s□↓⊗ → s◄↓ : A.9,4.4

s□↓⊗ $\nleftarrow$ s◄⫽ : 2.29,3.4

s□↓⊗ $\overset{\flat}{\leftrightarrow}$ s◄⫽ : 2.29,7.20,A.10,4.4

s□↓⊗ → s◄⫽ : A.5,A.9,4.4

s□↓⊗ $\nleftarrow$ s◁↓ : 2.23,3.4

s□↓⊗ $\overset{\flat}{\leftrightarrow}$ s◁↓ : 2.23,7.20,A.10

s□↓⊗ → s◁↓ : A.9,2.23

s□↓⊗ $\nleftarrow$ s◁⫽ : 2.23,3.4

s□↓⊗ $\overset{\flat}{\leftrightarrow}$ s◁⫽ : 7.20,A.10,2.23

s□↓⊗ → s◁⫽ : A.5,A.9,2.23

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k■↓× : A.10,7.20,A.9,A.7,A.6

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k■↓×1 : A.1,A.10,7.20,A.9,A.7,A.6

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k■↓⊗ : 7.20,A.7,A.10

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k■⫽× : A.10,7.20,A.7,A.6

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k■⫽×N : A.10,7.20,A.9,A.2,A.7,A.6

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k■⫽×1 : A.10,7.20,A.1,A.7,A.6

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k■⫽⊗ : A.10,7.20,A.7,A.6

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k■⫽⊗N : A.10,7.20,A.2,A.7

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k□↓× : A.9,A.7,7.20

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k□↓⊗ : A.7

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k□⫽× : 7.20,A.10,A.7,A.6

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k□⫽⊗ : 7.20,A.7,A.6

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k◄↓ : 2.29,7.20,A.7,A.10,A.9

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k◄⫽ : A.10,7.20,2.29,A.7,A.6

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k◁↓ : A.10,2.23,A.7,7.20

s□↓⊗ $\overset{\aleph_0}{\leftrightarrow}$ k◁⫽ : 7.20,2.23,A.10,A.7,A.6

s□⫽× $\overset{\flat}{\rightarrow}$ *perfect* : A.6

s□⫽× ← *perfect* : A.6

s□⫽× ↛ s■↓× : A.2,5.1

s□⫽× ← s■↓× : A.2,A.4

s□⫽× $\overset{\flat}{\leftrightarrow}$ s■↓× : 7.20

s□⫽× ↛ s■↓×1 : A.1,A.2,5.1

s□⫽× ← s■↓×1 : A.1,A.2,A.4

s□⫽× $\overset{\flat}{\leftrightarrow}$ s■↓×1 : A.1,7.20,A.10

s□⫽× $\overset{\aleph_0}{\leftrightarrow}$ s■↓×1P : A.9,A.3,7.20,A.10,A.7

s□⫽× ↛ s■↓⊗ : 3.4

s□⫽× ← s■↓⊗ : A.5,A.4,A.9

s□⫽× $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : 7.20,A.10

s□⫽× ↛ s■⫽× : 5.1

s□⫽× ← s■⫽× : A.4

s□⫽× $\overset{\flat}{\leftrightarrow}$ s■⫽× : 7.20

s□⫽× ↛ s■⫽×1 : A.1,5.1

s□⫽× ← s■⫽×1 : A.1,A.4

s□⫽× $\overset{\flat}{\leftrightarrow}$ s■⫽×1 : A.1,7.20

s□⫽× $\overset{\aleph_0}{\leftrightarrow}$ s■⫽×1P : A.10,A.3,7.20,A.7

s□⫽× ↛ s■⫽⊗ : 5.1

s□⫽× ← s■⫽⊗ : A.4,A.9

s□⫽× $\overset{\flat}{\leftrightarrow}$ s■⫽⊗ : A.10,7.20

s□⫽× ↛ s□↓× : 4.5,A.2,5.1

s□⫽× ← s□↓× : A.5

s□⫽× $\overset{\flat}{\leftrightarrow}$ s□↓× : 7.20

s□⫽× ↛ s□↓⊗ : 3.4

s□⫽× ← s□↓⊗ : A.9,A.5

s□⫽× $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : 7.20,A.10

s□⫽× ↛ s□⫽⊗ : 3.4

s□⫽× ← s□⫽⊗ : A.9

s□⫽× $\overset{\flat}{\leftrightarrow}$ s□⫽⊗ : 7.20

s□⫽× ↛ s◄↓ : 2.29,4.5,A.2,5.1

s□⫽× ← s◄↓ : 2.29,A.5

s□⊀× $\overset{\flat}{\leftrightarrow}$ s◀↓ : 2.29,7.20,4.4
s□⊀× ↔ s◀⊀ : 2.29,4.4
s□⊀× ↛ s◁↓ : 2.23,4.5,A.2,5.1
s□⊀× ← s◁↓ : 2.23,A.5
s□⊀× $\overset{\flat}{\leftrightarrow}$ s◁↓ : 7.20,2.23
s□⊀× ↔ s◁⊀ : 2.23
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓× : 7.20,A.10,A.9,A.7
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 : A.1,7.20,A.10,A.9,A.7
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : 7.20,A.10,A.7
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k■⊀× : 7.20,A.7,A.10
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k■⊀×N : A.9,A.2,7.20,A.10,A.7
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k■⊀×1 : A.10,A.1,7.20,A.7
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k■⊀⊗ : 7.20,A.7,A.10
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k■⊀⊗N : 7.20,A.10,A.2,A.7
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× : 7.20,A.10,A.9,A.7
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : 7.20,A.10,A.7,A.9
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k□⊀× : A.7
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k□⊀⊗ : 7.20,A.7,A.10
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k◀↓ : 7.20,A.10,2.29,A.9,A.7
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k◀⊀ : 7.20,2.29,A.10,A.7
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k◁↓ : 7.20,A.10,2.23,A.7
s□⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k◁⊀ : 2.23,A.7
s□⊀⊗ $\overset{\flat}{\nrightarrow}$ *perfect* : A.6
s□⊀⊗ ← *perfect* : A.6
s□⊀⊗ ↮ s■↓× : 3.4,A.2,5.1
s□⊀⊗ $\overset{\flat}{\leftrightarrow}$ s■↓× : 7.20,A.10
s□⊀⊗ ↮ s■↓×1 : A.1,3.4,A.2,5.1
s□⊀⊗ $\overset{\flat}{\leftrightarrow}$ s■↓×1 : A.10,7.20,A.1
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P : A.9,A.3,7.20,A.10,A.7
s□⊀⊗ ↛ s■↓⊗ : A.5,5.1
s□⊀⊗ ← s■↓⊗ : A.5,A.4
s□⊀⊗ $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : 7.20,A.4,A.10
s□⊀⊗ ↮ s■⊀× : 3.4,5.1
s□⊀⊗ $\overset{\flat}{\leftrightarrow}$ s■⊀× : 7.20,A.10
s□⊀⊗ ↮ s■⊀×1 : A.1,3.4,5.1
s□⊀⊗ $\overset{\flat}{\leftrightarrow}$ s■⊀×1 : A.1,7.20,A.10
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ s■⊀×1P : A.10,A.3,7.20,A.7
s□⊀⊗ ↛ s■⊀⊗ : 5.1
s□⊀⊗ ← s■⊀⊗ : A.4
s□⊀⊗ $\overset{\flat}{\leftrightarrow}$ s■⊀⊗ : 7.20
s□⊀⊗ ↮ s□↓× : 3.4,4.5,A.2,5.1
s□⊀⊗ $\overset{\flat}{\leftrightarrow}$ s□↓× : 7.20,A.10
s□⊀⊗ ↛ s□↓⊗ : 4.5,A.5,5.1
s□⊀⊗ ← s□↓⊗ : A.5
s□⊀⊗ $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : 7.20
s□⊀⊗ ↚ s□⊀× : 3.4
s□⊀⊗ $\overset{\flat}{\leftrightarrow}$ s□⊀× : A.10
s□⊀⊗ → s□⊀× : A.9
s□⊀⊗ ↮ s◀↓ : 4.4,3.4,2.29,4.5,A.2,5.1
s□⊀⊗ $\overset{\flat}{\leftrightarrow}$ s◀↓ : 2.29,7.20,A.10,4.4
s□⊀⊗ ↚ s◀⊀ : 2.29,3.4
s□⊀⊗ $\overset{\flat}{\leftrightarrow}$ s◀⊀ : 2.29,7.20,A.10,4.4
s□⊀⊗ → s◀⊀ : A.9,4.4
s□⊀⊗ ↮ s◁↓ : 2.23,3.4,4.5,A.2,5.1
s□⊀⊗ $\overset{\flat}{\leftrightarrow}$ s◁↓ : A.10,7.20,2.23
s□⊀⊗ ↚ s◁⊀ : 2.23,3.4
s□⊀⊗ $\overset{\flat}{\leftrightarrow}$ s◁⊀ : A.10,2.23,7.20
s□⊀⊗ → s◁⊀ : A.9,2.23
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓× : 7.20,A.10,A.9,A.7
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 : A.1,7.20,A.10,A.9,A.7
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : 7.20,A.10,A.7
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■⊀× : 7.20,A.10,A.7
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■⊀×N : A.9,A.2,7.20,A.10,A.7
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■⊀×1 : A.10,A.1,7.20,A.7
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■⊀⊗ : 7.20,A.7
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k■⊀⊗N : 7.20,A.10,A.2,A.7
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× : A.10,7.20,A.9,A.7
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : 7.20,A.9,A.7,A.10
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k□⊀× : A.10,A.7,7.20
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k□⊀⊗ : A.7
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k◀↓ : 2.29,7.20,A.7,A.10,A.9
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k◀⊀ : 2.29,7.20,A.7,A.10
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k◁↓ : A.10,7.20,2.23,A.7
s□⊀⊗ $\overset{\aleph\flat}{\leftrightarrow}$ k◁⊀ : 2.23,A.10,A.7,7.20

B.5. Statistische allgemeine Komponierbarkeit (s◀)

s◀↓ $\overset{\flat}{\nrightarrow}$ *perfect* : A.10,4.4,A.6
s◀↓ ← *perfect* : A.6,4.4
s◀↓ ↔ s■↓× : 2.29,4.5
s◀↓ ↔ s■↓×1 : A.1,2.29,4.5
s◀↓ $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P : 2.29,A.9,A.3,7.20,A.10,A.7,4.4,A.6
s◀↓ ↛ s■↓⊗ : 4.4,3.4
s◀↓ ← s■↓⊗ : A.9,2.29

s◄↓ $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : A.10,2.29,7.20
s◄↓ ↔ s■↓× : A.2,2.29,4.5
s◄↓ ↔ s■↓×1 : A.1,A.2,2.29,4.5
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P :
2.29,A.10,A.3,7.20,A.7,4.4,A.6
s◄↓ ↛ s■↓⊗ : 4.4,3.4
s◄↓ ← s■↓⊗ : A.2,A.9,2.29
s◄↓ $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : 7.20,A.10,4.4,2.29
s◄↓ ↔ s□↓× : 4.4,2.29
s◄↓ ↛ s□↓⊗ : 4.4,3.4
s◄↓ ← s□↓⊗ : A.9,4.4
s◄↓ $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : A.10,4.4,2.29,7.20
s◄↓ ↚ s□↓× : 2.29,4.5,A.2,5.1
s◄↓ $\overset{\flat}{\leftrightarrow}$ s□↓× : 7.20,4.4,2.29
s◄↓ → s□↓× : 2.29,A.5
s◄↓ ↭ s□↓⊗ : 2.29,4.5,3.4,A.2,5.1,4.4
s◄↓ $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : A.10,7.20,4.4,2.29
s◄↓ ↚ s◄↓ : 2.29,4.5,4.4,A.2,5.1
s◄↓ $\overset{\flat}{\leftrightarrow}$ s◄↓ : 2.29,7.20,4.4
s◄↓ → s◄↓ : 2.29,A.5,4.4
s◄↓ ↔ s◁↓ : 2.29,2.23,4.4
s◄↓ ↚ s◁↓ : 2.29,4.5,A.2,2.23,5.1
s◄↓ $\overset{\flat}{\leftrightarrow}$ s◁↓ : 2.23,7.20,4.4,2.29
s◄↓ → s◁↓ : 2.29,A.5,2.23
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓× :
2.29,7.20,A.10,A.9,A.7,4.4,A.6
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 :
2.29,A.1,7.20,A.10,A.9,A.7,4.4,A.6
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : 2.29,7.20,A.10,A.7,4.4
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓↓× : 2.29,7.20,A.10,A.7,4.4,A.6
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×N :
2.29,7.20,A.9,A.2,A.10,A.7,4.4,A.6
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 :
2.29,7.20,A.10,A.1,A.7,4.4,A.6
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : 2.29,7.20,A.7,A.10,4.4,A.6
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗N : 2.29,7.20,A.10,A.2,A.7,4.4
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× : 2.29,A.7,A.10,4.4,7.20
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : 2.29,7.20,A.7,A.10,4.4
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× : 2.29,7.20,A.7,A.10,4.4,A.6
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : 2.29,7.20,A.7,A.10,4.4,A.6
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k◄↓ : 2.29,7.20,A.10,A.9,A.7,4.4
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k◄↓ : 2.29,7.20,A.10,A.7,4.4,A.6
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k◁↓ : 2.29,2.23,A.7,A.10,4.4,7.20
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k◁↓ :
2.29,2.23,7.20,A.7,A.10,4.4,A.6
s◄↓ $\overset{\flat}{\nrightarrow}$ *perfect* : 2.29,A.6
s◄↓ ← *perfect* : A.6,4.4

s◄↓ ↛ s■↓× : 4.4,A.2,5.1
s◄↓ ← s■↓× : A.2,2.29
s◄↓ $\overset{\flat}{\leftrightarrow}$ s■↓× : 7.20,2.29
s◄↓ ↛ s■↓×1 : A.1,4.4,A.2,5.1
s◄↓ ← s■↓×1 : A.1,A.2,2.29
s◄↓ $\overset{\flat}{\leftrightarrow}$ s■↓×1 : A.1,A.2,2.29,7.20,A.10
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P :
2.29,A.9,A.3,7.20,A.10,A.7,4.4
s◄↓ ↛ s■↓⊗ : 2.29,3.4
s◄↓ ← s■↓⊗ : A.9,A.2,2.29
s◄↓ $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : A.9,A.2,2.29,7.20
s◄↓ ↛ s■↓× : 4.4,5.1
s◄↓ ← s■↓× : 2.29
s◄↓ $\overset{\flat}{\leftrightarrow}$ s■↓× : 2.29,7.20
s◄↓ ↛ s■↓×1 : A.1,4.4,5.1
s◄↓ ← s■↓×1 : A.1,2.29
s◄↓ $\overset{\flat}{\leftrightarrow}$ s■↓×1 : A.1,2.29,7.20
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1P :
2.29,A.10,A.3,7.20,A.7,4.4,A.6
s◄↓ ↛ s■↓⊗ : 2.29,3.4
s◄↓ ← s■↓⊗ : A.9,2.29
s◄↓ $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : A.4,A.10,4.4,2.29,7.20
s◄↓ ↛ s□↓× : 4.5,4.4,A.2,5.1
s◄↓ ← s□↓× : A.5,4.4
s◄↓ $\overset{\flat}{\leftrightarrow}$ s□↓× : A.5,4.4,2.29,7.20
s◄↓ ↛ s□↓⊗ : 2.29,3.4
s◄↓ ← s□↓⊗ : A.5,A.9,4.4
s◄↓ $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : A.10,7.20,4.4,2.29
s◄↓ ↔ s□↓× : 4.4,2.29
s◄↓ ↛ s□↓⊗ : 2.29,3.4
s◄↓ ← s□↓⊗ : A.9,4.4
s◄↓ $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : A.9,4.4,2.29,7.20
s◄↓ ↛ s◄↓ : 2.29,4.5,4.4,A.2,5.1
s◄↓ ← s◄↓ : 2.29,A.5,4.4
s◄↓ $\overset{\flat}{\leftrightarrow}$ s◄↓ : 2.29,7.20,4.4
s◄↓ ↛ s◁↓ : 2.23,4.5,4.4,A.2,5.1
s◄↓ ← s◁↓ : 2.23,A.5,4.4
s◄↓ $\overset{\flat}{\leftrightarrow}$ s◁↓ : 2.29,7.20,2.23,4.4
s◄↓ ↔ s◁↓ : 2.29,2.23,4.4
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓× : 2.29,7.20,A.10,A.9,A.7,4.4
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 :
2.29,A.1,7.20,A.10,A.9,A.7,4.4
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : 2.29,7.20,A.10,A.7,4.4
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓↓× : 2.29,7.20,A.10,A.7,4.4,A.6
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×N :
2.29,A.9,A.2,7.20,A.10,A.7,4.4
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓×1 :
2.29,A.10,A.1,7.20,A.7,4.4,A.6
s◄↓ $\overset{\aleph\flat}{\leftrightarrow}$ k■↓⊗ : 2.29,7.20,A.7,A.10,4.4,A.6

s◀ $\not\downarrow$ $\overset{\not\flat}{\leftrightarrow}$ k■ $\not\downarrow\otimes$N : 2.29,7.20,A.10,A.2,A.7,4.4
s◀ $\not\downarrow$ $\overset{\not\flat}{\leftrightarrow}$ k□↓× : 2.29,7.20,A.10,A.9,A.7,4.4
s◀ $\not\downarrow$ $\overset{\not\flat}{\leftrightarrow}$ k□↓⊗ : 2.29,7.20,A.10,A.7,4.4,A.9
s◀ $\not\downarrow$ $\overset{\not\flat}{\leftrightarrow}$ k□ $\not\downarrow\times$: 2.29,A.7,7.20,A.10,4.4,A.6
s◀ $\not\downarrow$ $\overset{\not\flat}{\leftrightarrow}$ k□ $\not\downarrow\otimes$: 2.29,7.20,A.7,A.10,4.4,A.6
s◀ $\not\downarrow$ $\overset{\not\flat}{\leftrightarrow}$ k◀↓ : 2.29,7.20,A.10,A.9,A.7,4.4
s◀ $\not\downarrow$ $\overset{\not\flat}{\leftrightarrow}$ k◀ $\not\downarrow$: 2.29,7.20,A.10,A.7,4.4,A.6
s◀ $\not\downarrow$ $\overset{\not\flat}{\leftrightarrow}$ k◁↓ : 2.29,7.20,A.10,2.23,A.7,4.4
s◀ $\not\downarrow$ $\overset{\not\flat}{\leftrightarrow}$ k◁ $\not\downarrow$:
2.29,2.23,A.7,7.20,A.10,4.4,A.6

B.6. Statistische einfache Komponierbarkeit (s◁)

s◁↓ $\overset{\flat}{\rightarrow}$ *perfect* : 2.23,A.6
s◁↓ ← *perfect* : A.6,2.23
s◁↓ ↔ s■↓× : 4.5,2.23
s◁↓ ↔ s■↓×1 : A.1,4.5,2.23
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ s■↓×1P :
2.23,A.9,A.3,7.20,A.10,A.7,A.6
s◁↓ ↛ s■↓⊗ : 2.23,3.4
s◁↓ ← s■↓⊗ : A.4,A.9,2.23
s◁↓ $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : A.10,7.20,2.23
s◁↓ ↔ s■ $\not\downarrow\times$: A.2,4.5,2.23
s◁↓ ↔ s■ $\not\downarrow\times$1 : A.1,A.2,4.5,2.23
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ s■ $\not\downarrow\times$1P :
2.23,A.10,A.3,7.20,A.7,A.6
s◁↓ ↛ s■ $\not\downarrow\otimes$: 2.23,3.4
s◁↓ ← s■ $\not\downarrow\otimes$: A.2,A.4,A.9,2.23
s◁↓ $\overset{\flat}{\leftrightarrow}$ s■ $\not\downarrow\otimes$: 7.20,A.10,2.23
s◁↓ ↔ s□↓× : 2.23
s◁↓ ↛ s□↓⊗ : 2.23,3.4
s◁↓ ← s□↓⊗ : A.9,2.23
s◁↓ $\overset{\flat}{\leftrightarrow}$ s□↓⊗ : A.9,2.23,7.20
s◁↓ ↚ s□ $\not\downarrow\times$: 2.23,4.5,A.2,5.1
s◁↓ $\overset{\flat}{\leftrightarrow}$ s□ $\not\downarrow\times$: 7.20,2.23
s◁↓ → s□ $\not\downarrow\times$: 2.23,A.5
s◁↓ ↮ s□ $\not\downarrow\otimes$: 2.23,3.4,4.5,A.2,5.1
s◁↓ $\overset{\flat}{\leftrightarrow}$ s□ $\not\downarrow\otimes$: A.10,7.20,2.23
s◁↓ ↔ s◀↓ : 2.29,2.23,4.4
s◁↓ ↚ s◀ $\not\downarrow$: 2.23,4.5,4.4,A.2,5.1
s◁↓ $\overset{\flat}{\leftrightarrow}$ s◀ $\not\downarrow$: 2.29,7.20,2.23,4.4
s◁↓ → s◀ $\not\downarrow$: 2.23,A.5,4.4
s◁↓ ↚ s◁ $\not\downarrow$: 2.23,4.5,A.2,5.1
s◁↓ $\overset{\flat}{\leftrightarrow}$ s◁ $\not\downarrow$: 2.23,7.20
s◁↓ → s◁ $\not\downarrow$: 2.23,A.5
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k■↓× : 2.23,7.20,A.10,A.9,A.7,A.6
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k■↓×1 :
2.23,A.1,7.20,A.10,A.9,A.7,A.6
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k■↓⊗ : 2.23,7.20,A.10,A.7
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k■ $\not\downarrow\times$: 2.23,7.20,A.10,A.7,A.6
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k■ $\not\downarrow\times$N :
2.23,7.20,A.9,A.2,A.10,A.7,A.6
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k■ $\not\downarrow\times$1 : 2.23,7.20,A.10,A.1,A.7,A.6
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k■ $\not\downarrow\otimes$: A.10,2.23,A.6,A.7,7.20
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k■ $\not\downarrow\otimes$N : 2.23,7.20,A.10,A.2,A.7
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k□↓× : A.10,2.23,7.20,A.7
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k□↓⊗ : 2.23,7.20,A.7,A.10
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k□ $\not\downarrow\times$: A.10,2.23,A.6,A.7,7.20
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k□ $\not\downarrow\otimes$: 2.23,7.20,A.7,A.10,A.6
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k◀↓ : 2.23,2.29,A.7,7.20,A.10,A.9
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k◀ $\not\downarrow$: A.10,2.23,A.6,2.29,A.7,7.20
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k◁↓ : 2.23,A.10,7.20,A.7
s◁↓ $\overset{\not\flat}{\leftrightarrow}$ k◁ $\not\downarrow$: A.10,2.23,A.6,A.7,7.20
s◁ $\not\downarrow$ $\overset{\flat}{\rightarrow}$ *perfect* : 7.20,A.10,2.23,A.6
s◁ $\not\downarrow$ ← *perfect* : A.6,2.23
s◁ $\not\downarrow$ ↛ s■↓× : A.2,2.23,5.1
s◁ $\not\downarrow$ ← s■↓× : A.2,A.4,2.23
s◁ $\not\downarrow$ $\overset{\flat}{\leftrightarrow}$ s■↓× : 7.20,2.23
s◁ $\not\downarrow$ ↛ s■↓×1 : A.1,A.2,2.23,5.1
s◁ $\not\downarrow$ ← s■↓×1 : A.1,A.2,A.4,2.23
s◁ $\not\downarrow$ $\overset{\flat}{\leftrightarrow}$ s■↓×1 : A.1,7.20,2.23,A.10
s◁ $\not\downarrow$ $\overset{\not\flat}{\leftrightarrow}$ s■↓×1P :
2.23,A.9,A.3,7.20,A.10,A.7
s◁ $\not\downarrow$ ↛ s■↓⊗ : 2.23,3.4
s◁ $\not\downarrow$ ← s■↓⊗ : A.9,A.2,A.4,2.23
s◁ $\not\downarrow$ $\overset{\flat}{\leftrightarrow}$ s■↓⊗ : 7.20,A.10,2.23
s◁ $\not\downarrow$ ↛ s■ $\not\downarrow\times$: 2.23,5.1
s◁ $\not\downarrow$ ← s■ $\not\downarrow\times$: A.4,2.23
s◁ $\not\downarrow$ $\overset{\flat}{\leftrightarrow}$ s■ $\not\downarrow\times$: 7.20,2.23
s◁ $\not\downarrow$ ↛ s■ $\not\downarrow\times$1 : A.1,2.23,5.1
s◁ $\not\downarrow$ ← s■ $\not\downarrow\times$1 : A.1,A.4,2.23
s◁ $\not\downarrow$ $\overset{\flat}{\leftrightarrow}$ s■ $\not\downarrow\times$1 : A.1,A.4,2.23,7.20
s◁ $\not\downarrow$ $\overset{\not\flat}{\leftrightarrow}$ s■ $\not\downarrow\times$1P : 2.23,A.10,A.3,7.20,A.7
s◁ $\not\downarrow$ ↛ s■ $\not\downarrow\otimes$: 2.23,5.1
s◁ $\not\downarrow$ ← s■ $\not\downarrow\otimes$: A.9,A.4,2.23
s◁ $\not\downarrow$ $\overset{\flat}{\leftrightarrow}$ s■ $\not\downarrow\otimes$: 7.20,A.10,2.23
s◁ $\not\downarrow$ ↛ s□↓× : 4.5,A.2,2.23,5.1
s◁ $\not\downarrow$ ← s□↓× : A.5,2.23
s◁ $\not\downarrow$ $\overset{\flat}{\leftrightarrow}$ s□↓× : 7.20,2.23

$s\triangleleft\not\downarrow \nrightarrow s\square\downarrow\otimes$: 2.23,3.4
$s\triangleleft\not\downarrow \leftarrow s\square\downarrow\otimes$: A.5,A.9,2.23
$s\triangleleft\not\downarrow \overset{\flat}{\leftrightarrow} s\square\downarrow\otimes$: 7.20,A.10,2.23
$s\triangleleft\not\downarrow \leftrightarrow s\square\not\downarrow\times$: 2.23
$s\triangleleft\not\downarrow \nrightarrow s\square\not\downarrow\otimes$: 2.23,3.4
$s\triangleleft\not\downarrow \leftarrow s\square\not\downarrow\otimes$: A.9,2.23
$s\triangleleft\not\downarrow \overset{\flat}{\leftrightarrow} s\square\not\downarrow\otimes$: A.10,2.23,7.20
$s\triangleleft\not\downarrow \nrightarrow s\blacktriangleleft\downarrow$: 2.29,4.5,A.2,2.23,5.1
$s\triangleleft\not\downarrow \leftarrow s\blacktriangleleft\downarrow$: 2.29,A.5,2.23
$s\triangleleft\not\downarrow \overset{\flat}{\leftrightarrow} s\blacktriangleleft\downarrow$: 2.29,7.20,2.23,4.4
$s\triangleleft\not\downarrow \leftrightarrow s\blacktriangleleft\not\downarrow$: 2.29,2.23,4.4
$s\triangleleft\not\downarrow \nrightarrow s\triangleleft\downarrow$: 2.23,4.5,A.2,5.1
$s\triangleleft\not\downarrow \leftarrow s\triangleleft\downarrow$: 2.23,A.5
$s\triangleleft\not\downarrow \overset{\flat}{\leftrightarrow} s\triangleleft\downarrow$: 2.23,7.20
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\blacksquare\downarrow\times$: 2.23,7.20,A.10,A.9,A.7
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\blacksquare\downarrow\times1$: 2.23,A.1,7.20,A.10,A.9,A.7
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\blacksquare\downarrow\otimes$: 2.23,7.20,A.10,A.7
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\blacksquare\not\downarrow\times$: 2.23,7.20,A.7,A.10
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\blacksquare\not\downarrow\times N$: 2.23,A.9,A.2,7.20,A.10,A.7
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\blacksquare\not\downarrow\times1$: 2.23,A.10,A.1,7.20,A.7
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\blacksquare\not\downarrow\otimes$: 2.23,7.20,A.7,A.10
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\blacksquare\not\downarrow\otimes N$: 2.23,7.20,A.10,A.2,A.7
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\square\downarrow\times$: 2.23,7.20,A.7,A.10,A.9
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\square\downarrow\otimes$: 2.23,7.20,A.10,A.7,A.9
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\square\not\downarrow\times$: 2.23,A.7
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\square\not\downarrow\otimes$: 2.23,7.20,A.7,A.10
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\blacktriangleleft\downarrow$: 2.23,7.20,A.10,2.29,A.9,A.7
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\blacktriangleleft\not\downarrow$: 2.23,2.29,A.7,7.20,A.10
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\triangleleft\downarrow$: 2.23,7.20,A.10,A.7
$s\triangleleft\not\downarrow \overset{\aleph^\flat}{\leftrightarrow} k\triangleleft\not\downarrow$: 2.23,A.7

B.7. Komplexitätstheoretische allgemeine Sicherheit (k■)

$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow}$ *perfect* : A.7
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\blacksquare\downarrow\times$: A.7
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\blacksquare\downarrow\times1$: 7.20,A.10,A.1,A.7
$k\blacksquare\downarrow\times \leftrightarrow s\blacksquare\downarrow\times1P$: A.3
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\blacksquare\downarrow\otimes$: 7.20,A.7,A.9
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\blacksquare\not\downarrow\times$: 7.20,A.7,A.10,A.9
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\blacksquare\not\downarrow\times1$: A.10,A.1,7.20,A.7,A.9
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftarrow} s\blacksquare\not\downarrow\times1P$: A.10,A.3,A.8
$k\blacksquare\downarrow\times \rightarrow s\blacksquare\not\downarrow\times1P$: A.5,A.3
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\blacksquare\not\downarrow\otimes$: 7.20,A.7,A.10,A.9
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\square\downarrow\times$: 7.20,A.7,A.10,A.9
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\square\downarrow\otimes$: A.6,A.7,A.10,7.20,A.9
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\square\not\downarrow\times$: 7.20,A.10,A.7,A.9
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\square\not\downarrow\otimes$: 7.20,A.7,A.10,A.9
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\blacktriangleleft\downarrow$: A.10,4.4,A.6,A.7,2.29,7.20,A.9
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\blacktriangleleft\not\downarrow$: 7.20,4.4,A.7,2.29,A.10,A.9
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\triangleleft\downarrow$: A.10,2.23,A.6,A.7,7.20,A.9
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftrightarrow} s\triangleleft\not\downarrow$: 2.23,7.20,A.10,A.7,A.9
$k\blacksquare\downarrow\times \leftrightarrow k\blacksquare\downarrow\times1$: A.1
$k\blacksquare\downarrow\times \overset{\flat}{\leftrightarrow} k\blacksquare\downarrow\otimes$: A.10
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftarrow} k\blacksquare\not\downarrow\times$: A.8
$k\blacksquare\downarrow\times \rightarrow k\blacksquare\not\downarrow\times$: A.5
$k\blacksquare\downarrow\times \leftrightarrow k\blacksquare\not\downarrow\times N$: A.2
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftarrow} k\blacksquare\not\downarrow\times1$: A.10,A.1,A.8
$k\blacksquare\downarrow\times \rightarrow k\blacksquare\not\downarrow\times1$: A.5,A.1
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftarrow} k\blacksquare\not\downarrow\otimes$: A.8
$k\blacksquare\downarrow\times \overset{\flat}{\rightarrow} k\blacksquare\not\downarrow\otimes$: A.5,A.10
$k\blacksquare\downarrow\times \overset{\flat}{\leftrightarrow} k\blacksquare\not\downarrow\otimes N$: A.2,A.10
$k\blacksquare\downarrow\times \nleftarrow k\square\downarrow\times$: 5.2
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftarrow} k\square\downarrow\times$: 5.10
$k\blacksquare\downarrow\times \rightarrow k\square\downarrow\times$: A.4
$k\blacksquare\downarrow\times \nleftarrow k\square\downarrow\otimes$: 5.2
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftarrow} k\square\downarrow\otimes$: 5.10
$k\blacksquare\downarrow\times \overset{\flat}{\rightarrow} k\square\downarrow\otimes$: 2.29,A.10
$k\blacksquare\downarrow\times \nleftarrow k\square\not\downarrow\times$: 5.2
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftarrow} k\square\not\downarrow\times$: 5.10
$k\blacksquare\downarrow\times \rightarrow k\square\not\downarrow\times$: A.5,A.4
$k\blacksquare\downarrow\times \nleftarrow k\square\not\downarrow\otimes$: 5.2
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftarrow} k\square\not\downarrow\otimes$: 5.10
$k\blacksquare\downarrow\times \overset{\flat}{\rightarrow} k\square\not\downarrow\otimes$: 2.29,A.10,A.5
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftarrow} k\blacktriangleleft\downarrow$: 6.2
$k\blacksquare\downarrow\times \overset{\flat}{\rightarrow} k\blacktriangleleft\downarrow$: 2.29
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftarrow} k\blacktriangleleft\not\downarrow$: 6.2
$k\blacksquare\downarrow\times \overset{\flat}{\rightarrow} k\blacktriangleleft\not\downarrow$: A.5,2.29
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftarrow} k\triangleleft\downarrow$: 2.23,5.10
$k\blacksquare\downarrow\times \overset{\flat}{\rightarrow} k\triangleleft\downarrow$: 2.29,2.23
$k\blacksquare\downarrow\times \overset{\aleph^\flat}{\leftarrow} k\triangleleft\not\downarrow$: 2.23,5.10
$k\blacksquare\downarrow\times \overset{\flat}{\rightarrow} k\triangleleft\not\downarrow$: A.5,2.29,2.23

k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ *perfect* : A.1,A.7
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s■↓× : A.1,A.7
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s■↓×1 : A.1,7.20,A.10,A.7
k■↓×1 $\leftrightarrow$ s■↓×1P : A.3,A.1
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s■↓⊗ : A.1,A.9,A.7,7.20
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s■⊀× : A.1,7.20,A.10,A.9,A.7
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s■⊀×1 : A.1,A.10,7.20,A.7,A.9
k■↓×1 $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ s■⊀×1P : A.1,A.10,A.3,A.8
k■↓×1 $\rightarrow$ s■⊀×1P : A.1,A.5,A.3
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s■⊀⊗ : A.1,A.10,7.20,A.9,A.7
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s□↓× : A.1,7.20,A.10,A.9,A.7
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s□↓⊗ : A.1,A.6,A.7,A.10,7.20,A.9
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s□⊀× : A.1,7.20,A.10,A.9,A.7
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s□⊀⊗ : A.1,7.20,A.7,A.10,A.9
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s◀↓ :
A.1,A.10,4.4,A.6,A.7,2.29,7.20,A.9
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s◀⊀ :
2.29,A.1,7.20,A.10,A.9,A.7,4.4
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s◁↓ :
A.1,A.10,2.23,A.6,A.7,7.20,A.9
k■↓×1 $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s◁⊀ : A.1,2.23,7.20,A.10,A.7,A.9
k■↓×1 $\leftrightarrow$ k■↓× : A.1
k■↓×1 $\overset{\flat}{\leftrightarrow}$ k■↓⊗ : A.10,A.1
k■↓×1 $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k■⊀× : A.1,A.8
k■↓×1 $\rightarrow$ k■⊀× : A.1,A.5
k■↓×1 $\leftrightarrow$ k■⊀×N : A.2,A.1
k■↓×1 $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k■⊀×1 : A.1,A.10,A.8
k■↓×1 $\rightarrow$ k■⊀×1 : A.1,A.5
k■↓×1 $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k■⊀⊗ : A.1,A.8
k■↓×1 $\overset{\flat}{\rightarrow}$ k■⊀⊗ : A.1,A.5,A.10
k■↓×1 $\overset{\flat}{\leftrightarrow}$ k■⊀⊗N : A.2,A.10,A.1
k■↓×1 $\nleftarrow$ k□↓× : A.1,5.2
k■↓×1 $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k□↓× : A.1,5.10
k■↓×1 $\rightarrow$ k□↓× : A.1,A.4
k■↓×1 $\nleftarrow$ k□↓⊗ : A.1,5.2
k■↓×1 $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k□↓⊗ : A.1,5.10
k■↓×1 $\overset{\flat}{\rightarrow}$ k□↓⊗ : A.1,2.29,A.10
k■↓×1 $\nleftarrow$ k□⊀× : A.1,5.2
k■↓×1 $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k□⊀× : A.1,5.10
k■↓×1 $\rightarrow$ k□⊀× : A.1,A.5,A.4
k■↓×1 $\nleftarrow$ k□⊀⊗ : A.1,5.2
k■↓×1 $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k□⊀⊗ : A.1,5.10
k■↓×1 $\overset{\flat}{\rightarrow}$ k□⊀⊗ : A.1,2.29,A.10,A.5
k■↓×1 $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k◀↓ : A.1,6.2
k■↓×1 $\overset{\flat}{\rightarrow}$ k◀↓ : A.1,2.29
k■↓×1 $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k◀⊀ : A.1,6.2
k■↓×1 $\overset{\flat}{\rightarrow}$ k◀⊀ : A.1,A.5,2.29
k■↓×1 $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k◁↓ : A.1,2.23,5.10
k■↓×1 $\overset{\flat}{\rightarrow}$ k◁↓ : A.1,2.29,2.23
k■↓×1 $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k◁⊀ : A.1,2.23,5.10
k■↓×1 $\overset{\flat}{\rightarrow}$ k◁⊀ : A.1,A.5,2.29,2.23
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ *perfect* : A.7
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s■↓× : A.9,A.7,7.20
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s■↓×1 : 7.20,A.10,A.1,A.9,A.7
k■↓⊗ $\overset{\flat}{\leftrightarrow}$ s■↓×1P : A.10,A.3
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s■↓⊗ : A.7
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s■⊀× : 7.20,A.9,A.7,A.10
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s■⊀×1 : A.10,A.1,7.20,A.7,A.9
k■↓⊗ $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ s■⊀×1P : A.10,A.3,A.8
k■↓⊗ $\overset{\flat}{\rightarrow}$ s■⊀×1P : A.5,A.10,A.3
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s■⊀⊗ : 7.20,A.7,A.10
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s□↓× : A.10,7.20,A.7
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s□↓⊗ : 7.20,A.7,A.10
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s□⊀× : 7.20,A.10,A.7
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s□⊀⊗ : 7.20,A.10,A.7
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s◀↓ : A.10,4.4,7.20,A.7,2.29
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s◀⊀ : 7.20,4.4,A.10,A.7,2.29
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s◁↓ : A.10,2.23,7.20,A.7
k■↓⊗ $\overset{\mathbb{N}\flat}{\leftrightarrow}$ s◁⊀ : 2.23,7.20,A.10,A.7
k■↓⊗ $\overset{\flat}{\leftrightarrow}$ k■↓× : A.10
k■↓⊗ $\overset{\flat}{\leftrightarrow}$ k■↓×1 : A.1,A.10
k■↓⊗ $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k■⊀× : A.8
k■↓⊗ $\overset{\flat}{\rightarrow}$ k■⊀× : A.9,A.5
k■↓⊗ $\overset{\flat}{\leftrightarrow}$ k■⊀×N : A.2,A.10
k■↓⊗ $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k■⊀×1 : A.10,A.1,A.8
k■↓⊗ $\overset{\flat}{\rightarrow}$ k■⊀×1 : A.5,A.10,A.1
k■↓⊗ $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k■⊀⊗ : A.8
k■↓⊗ $\rightarrow$ k■⊀⊗ : A.5
k■↓⊗ $\leftrightarrow$ k■⊀⊗N : A.2
k■↓⊗ $\nleftarrow$ k□↓× : 5.2
k■↓⊗ $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k□↓× : 5.10
k■↓⊗ $\overset{\flat}{\rightarrow}$ k□↓× : A.10,2.29
k■↓⊗ $\nleftarrow$ k□↓⊗ : 5.2
k■↓⊗ $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k□↓⊗ : 5.10
k■↓⊗ $\rightarrow$ k□↓⊗ : A.4
k■↓⊗ $\nleftarrow$ k□⊀× : 5.2
k■↓⊗ $\overset{\mathbb{N}\flat}{\nleftrightarrow}$ k□⊀× : 5.10

k■↓⊗ $\xrightarrow{\flat}$ k□⊀× : A.5,A.10,A.4
k■↓⊗ $\nleftarrow$ k□⊀⊗ : 5.2
k■↓⊗ $\overset{\aleph\flat}{\nleftarrow}$ k□⊀⊗ : 5.10
k■↓⊗ $\rightarrow$ k□⊀⊗ : A.5,A.4
k■↓⊗ $\overset{\aleph\flat}{\nleftarrow}$ k◀↓ : 6.2
k■↓⊗ $\xrightarrow{\flat}$ k◀↓ : A.9,2.29
k■↓⊗ $\overset{\aleph\flat}{\nleftarrow}$ k◀⊀ : 6.2
k■↓⊗ $\xrightarrow{\flat}$ k◀⊀ : A.5,A.10,2.29
k■↓⊗ $\overset{\aleph\flat}{\nleftarrow}$ k◁↓ : 2.23,5.10
k■↓⊗ $\xrightarrow{\flat}$ k◁↓ : A.10,2.29,2.23
k■↓⊗ $\overset{\aleph\flat}{\nleftarrow}$ k◁⊀ : 2.23,5.10
k■↓⊗ $\xrightarrow{\flat}$ k◁⊀ : A.5,A.9,2.29,2.23
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ *perfect* : A.7
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s■↓× : 7.20,A.7,A.9,A.10
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1 : 7.20,A.10,A.1,A.9,A.7
k■⊀× $\overset{\aleph\flat}{\rightarrow}$ s■↓×1P : A.3,A.8
k■⊀× $\leftarrow$ s■↓×1P : A.3,A.5
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s■↓⊗ : A.10,7.20,A.7
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s■⊀× : A.7
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s■⊀×1 : A.10,A.1,7.20,A.7
k■⊀× $\leftrightarrow$ s■⊀×1P : A.3
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s■⊀⊗ : 7.20,A.7,A.10
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s□↓× : 7.20,A.9,A.10,A.7
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s□↓⊗ : A.6,A.7,A.10,7.20
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s□⊀× : 7.20,A.7,A.10
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s□⊀⊗ : 7.20,A.7,A.10
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s◀↓ : A.10,4.4,A.6,A.7,2.29,7.20
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s◀⊀ : 7.20,A.10,4.4,A.6,A.7,2.29
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s◁↓ : A.10,2.23,A.6,A.7,7.20
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ s◁⊀ : 2.23,7.20,A.7,A.10
k■⊀× $\overset{\aleph\flat}{\rightarrow}$ k■↓× : A.8
k■⊀× $\leftarrow$ k■↓× : A.5
k■⊀× $\overset{\aleph\flat}{\rightarrow}$ k■↓×1 : A.1,A.8
k■⊀× $\leftarrow$ k■↓×1 : A.1,A.5
k■⊀× $\overset{\aleph\flat}{\rightarrow}$ k■↓⊗ : A.8
k■⊀× $\xleftarrow{\flat}$ k■↓⊗ : A.9,A.5
k■⊀× $\overset{\aleph\flat}{\rightarrow}$ k■⊀×N : A.2,A.8
k■⊀× $\leftarrow$ k■⊀×N : A.2,A.5
k■⊀× $\leftrightarrow$ k■⊀×1 : A.1
k■⊀× $\overset{\flat}{\leftrightarrow}$ k■⊀⊗ : A.10
k■⊀× $\overset{\aleph\flat}{\rightarrow}$ k■⊀⊗N : A.2,A.8
k■⊀× $\xleftarrow{\flat}$ k■⊀⊗N : A.2,A.9,A.5
k■⊀× $\nleftarrow$ k□↓× : 5.2
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k□↓× : 5.10,A.8
k■⊀× $\nleftarrow$ k□↓⊗ : 5.2
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k□↓⊗ : 5.10,A.8
k■⊀× $\nleftarrow$ k□⊀× : 5.2
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k□⊀× : 5.10
k■⊀× $\rightarrow$ k□⊀× : A.4
k■⊀× $\nleftarrow$ k□⊀⊗ : 5.2
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k□⊀⊗ : 5.10
k■⊀× $\xrightarrow{\flat}$ k□⊀⊗ : A.10,A.4
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k◀↓ : 6.2,2.29,5.10,A.8
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k◀⊀ : 6.2
k■⊀× $\xrightarrow{\flat}$ k◀⊀ : 2.29
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k◁↓ : 2.23,5.10,A.8
k■⊀× $\overset{\aleph\flat}{\leftrightarrow}$ k◁⊀ : 2.23,5.10
k■⊀× $\xrightarrow{\flat}$ k◁⊀ : 2.29,2.23
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ *perfect* : A.9,A.2,A.7
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s■↓× : A.9,A.2,7.20,A.7
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s■↓×1 : A.2,7.20,A.10,A.1,A.7,A.9
k■⊀×N $\leftrightarrow$ s■↓×1P : A.3,A.2
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s■↓⊗ : A.9,A.2,A.7,7.20
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s■⊀× : 7.20,A.9,A.2,A.10,A.7
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s■⊀×1 : A.2,A.10,A.1,7.20,A.7,A.9
k■⊀×N $\overset{\aleph\flat}{\nleftarrow}$ s■⊀×1P : A.2,A.10,A.3,A.8
k■⊀×N $\rightarrow$ s■⊀×1P : A.2,A.5,A.3
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s■⊀⊗ : A.2,7.20,A.7,A.9,A.10
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s□↓× : 7.20,A.9,A.2,A.10,A.7
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s□↓⊗ : A.10,7.20,A.9,A.2,A.7,A.6
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s□⊀× : A.9,A.2,7.20,A.10,A.7
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s□⊀⊗ : A.2,7.20,A.7,A.9,A.10
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s◀↓ : 2.29,7.20,A.9,A.2,A.10,A.7,4.4,A.6
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s◀⊀ : 2.29,A.9,A.2,7.20,A.10,A.7,4.4
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s◁↓ : A.2,A.10,2.23,A.6,A.7,7.20,A.9
k■⊀×N $\overset{\aleph\flat}{\leftrightarrow}$ s◁⊀ : A.2,2.23,7.20,A.10,A.7,A.9
k■⊀×N $\leftrightarrow$ k■↓× : A.2
k■⊀×N $\leftrightarrow$ k■↓×1 : A.1,A.2
k■⊀×N $\overset{\flat}{\leftrightarrow}$ k■↓⊗ : A.10,A.2
k■⊀×N $\overset{\aleph\flat}{\nleftarrow}$ k■⊀× : A.2,A.8
k■⊀×N $\rightarrow$ k■⊀× : A.2,A.5

k■⊀×N $\overset{\aleph_0^\flat}{\leftrightarrow}$ k■⊀×1 : A.2,A.10,A.1,A.8
k■⊀×N $\rightarrow$ k■⊀×1 : A.2,A.5,A.1
k■⊀×N $\overset{\aleph_0^\flat}{\leftrightarrow}$ k■⊀⊗ : A.2,A.8
k■⊀×N $\overset{\flat}{\rightarrow}$ k■⊀⊗ : A.2,A.5,A.10
k■⊀×N $\overset{\flat}{\leftrightarrow}$ k■⊀⊗N : A.2,A.10
k■⊀×N $\nleftarrow$ k□↓× : A.2,5.2
k■⊀×N $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□↓× : A.2,5.10
k■⊀×N $\rightarrow$ k□↓× : A.2,A.4
k■⊀×N $\nleftarrow$ k□↓⊗ : A.2,5.2
k■⊀×N $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□↓⊗ : A.2,5.10
k■⊀×N $\overset{\flat}{\rightarrow}$ k□↓⊗ : A.2,2.29,A.10
k■⊀×N $\nleftarrow$ k□⊀× : A.2,5.2
k■⊀×N $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□⊀× : A.2,5.10
k■⊀×N $\rightarrow$ k□⊀× : A.2,A.5,A.4
k■⊀×N $\nleftarrow$ k□⊀⊗ : A.2,5.2
k■⊀×N $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□⊀⊗ : A.2,5.10
k■⊀×N $\overset{\flat}{\rightarrow}$ k□⊀⊗ : A.2,2.29,A.10,A.5
k■⊀×N $\overset{\aleph_0^\flat}{\leftrightarrow}$ k◄↓ : A.2,6.2
k■⊀×N $\overset{\flat}{\rightarrow}$ k◄↓ : A.2,2.29
k■⊀×N $\overset{\aleph_0^\flat}{\leftrightarrow}$ k◄⊀ : A.2,6.2
k■⊀×N $\overset{\flat}{\rightarrow}$ k◄⊀ : A.2,A.5,2.29
k■⊀×N $\overset{\aleph_0^\flat}{\leftrightarrow}$ k◁↓ : A.2,2.23,5.10
k■⊀×N $\overset{\flat}{\rightarrow}$ k◁↓ : A.2,2.29,2.23
k■⊀×N $\overset{\aleph_0^\flat}{\leftrightarrow}$ k◁⊀ : A.2,2.23,5.10
k■⊀×N $\overset{\flat}{\rightarrow}$ k◁⊀ : A.2,A.5,2.29,2.23
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ *perfect* : A.1,A.7,A.10
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↓× : A.10,A.1,7.20,A.7,A.9
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↓×1 : A.1,7.20,A.10,A.9,A.7
k■⊀×1 $\overset{\aleph_0^\flat}{\rightarrow}$ s■↓×1P : A.3,A.10,A.1,A.8
k■⊀×1 $\leftarrow$ s■↓×1P : A.3,A.5,A.1
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↓⊗ : A.10,A.1,7.20,A.7
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■⊀× : A.10,A.1,7.20,A.7
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■⊀×1 : A.1,A.10,7.20,A.7
k■⊀×1 $\leftrightarrow$ s■⊀×1P : A.3,A.1
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■⊀⊗ : A.10,A.1,A.7,7.20
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s□↓× : A.1,7.20,A.9,A.10,A.7
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s□↓⊗ : A.10,7.20,A.1,A.7,A.6
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s□⊀× : A.1,7.20,A.7,A.10
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s□⊀⊗ : A.10,A.1,7.20,A.7
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s◄↓ :
2.29,7.20,A.10,A.1,A.7,4.4,A.6
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s◄⊀ :
2.29,A.10,A.1,7.20,A.7,4.4,A.6
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s◁↓ : A.1,A.10,2.23,A.6,A.7,7.20
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ s◁⊀ : A.1,2.23,7.20,A.7,A.10
k■⊀×1 $\overset{\aleph_0^\flat}{\rightarrow}$ k■↓× : A.10,A.1,A.8
k■⊀×1 $\leftarrow$ k■↓× : A.5,A.1
k■⊀×1 $\overset{\aleph_0^\flat}{\rightarrow}$ k■↓×1 : A.1,A.10,A.8
k■⊀×1 $\leftarrow$ k■↓×1 : A.1,A.5
k■⊀×1 $\overset{\aleph_0^\flat}{\rightarrow}$ k■↓⊗ : A.10,A.1,A.8
k■⊀×1 $\overset{\flat}{\leftarrow}$ k■↓⊗ : A.5,A.10,A.1
k■⊀×1 $\leftrightarrow$ k■⊀× : A.1
k■⊀×1 $\overset{\aleph_0^\flat}{\rightarrow}$ k■⊀×N : A.2,A.10,A.1,A.8
k■⊀×1 $\leftarrow$ k■⊀×N : A.2,A.5,A.1
k■⊀×1 $\overset{\flat}{\leftrightarrow}$ k■⊀⊗ : A.10,A.1
k■⊀×1 $\overset{\aleph_0^\flat}{\rightarrow}$ k■⊀⊗N : A.2,A.10,A.1,A.8
k■⊀×1 $\overset{\flat}{\leftarrow}$ k■⊀⊗N : A.2,A.5,A.10,A.1
k■⊀×1 $\nleftarrow$ k□↓× : A.1,5.2
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□↓× : A.1,5.10,A.10,A.8
k■⊀×1 $\nleftarrow$ k□↓⊗ : A.1,5.2
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□↓⊗ : A.1,A.8,5.10,A.10
k■⊀×1 $\nleftarrow$ k□⊀× : A.1,5.2
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□⊀× : A.1,5.10
k■⊀×1 $\rightarrow$ k□⊀× : A.1,A.4
k■⊀×1 $\nleftarrow$ k□⊀⊗ : A.1,5.2
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□⊀⊗ : A.1,5.10
k■⊀×1 $\overset{\flat}{\rightarrow}$ k□⊀⊗ : A.1,A.10,A.4
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ k◄↓ : A.1,6.2,A.10,2.29,A.8
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ k◄⊀ : A.1,6.2
k■⊀×1 $\overset{\flat}{\rightarrow}$ k◄⊀ : A.1,2.29
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ k◁↓ : 2.23,A.10,A.1,A.8,5.10
k■⊀×1 $\overset{\aleph_0^\flat}{\leftrightarrow}$ k◁⊀ : A.1,2.23,5.10
k■⊀×1 $\overset{\flat}{\rightarrow}$ k◁⊀ : A.1,2.29,2.23
k■⊀⊗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ *perfect* : A.7
k■⊀⊗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↓× : A.9,A.10,7.20,A.7
k■⊀⊗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↓×1 : 7.20,A.10,A.1,A.9,A.7
k■⊀⊗ $\overset{\aleph_0^\flat}{\rightarrow}$ s■↓×1P : A.3,A.8
k■⊀⊗ $\overset{\flat}{\leftarrow}$ s■↓×1P : A.3,A.5,A.10
k■⊀⊗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↓⊗ : 7.20,A.7,A.10
k■⊀⊗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■⊀× : A.10,A.7,7.20
k■⊀⊗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■⊀×1 : A.10,A.1,A.7,7.20
k■⊀⊗ $\overset{\flat}{\leftrightarrow}$ s■⊀×1P : A.3,A.10
k■⊀⊗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■⊀⊗ : A.7
k■⊀⊗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s□↓× : 7.20,A.9,A.10,A.7
k■⊀⊗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s□↓⊗ : A.6,A.7,A.10,7.20
k■⊀⊗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s□⊀× : A.10,7.20,A.7
k■⊀⊗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s□⊀⊗ : 7.20,A.7

k■ $\not{\otimes}$ $\overset{\aleph_0}{\leftrightarrow}$ s◀↓ : A.10,4.4,A.6,A.7,2.29,7.20
k■ $\not{\otimes}$ $\overset{\aleph_0}{\leftrightarrow}$ s◀$\not{\downarrow}$: 7.20,A.10,4.4,A.6,A.7,2.29
k■ $\not{\otimes}$ $\overset{\aleph_0}{\leftrightarrow}$ s◁↓ : A.10,2.23,A.6,A.7,7.20
k■ $\not{\otimes}$ $\overset{\aleph_0}{\leftrightarrow}$ s◁$\not{\downarrow}$: 2.23,A.10,7.20,A.7
k■ $\not{\otimes}$ $\overset{\aleph_0}{\rightarrow}$ k■↓× : A.8
k■ $\not{\otimes}$ $\overset{\flat}{\leftarrow}$ k■↓× : A.5,A.10
k■ $\not{\otimes}$ $\overset{\aleph_0}{\rightarrow}$ k■↓×1 : A.1,A.8
k■ $\not{\otimes}$ $\overset{\flat}{\leftarrow}$ k■↓×1 : A.1,A.5,A.10
k■ $\not{\otimes}$ $\overset{\aleph_0}{\rightarrow}$ k■↓⊗ : A.8
k■ $\not{\otimes}$ $\leftarrow$ k■↓⊗ : A.5
k■ $\not{\otimes}$ $\overset{\flat}{\hookrightarrow}$ k■$\not{\downarrow}$× : A.10
k■ $\not{\otimes}$ $\overset{\aleph_0}{\rightarrow}$ k■$\not{\downarrow}$×N : A.2,A.8
k■ $\not{\otimes}$ $\overset{\flat}{\leftarrow}$ k■$\not{\downarrow}$×N : A.2,A.5,A.10
k■ $\not{\otimes}$ $\overset{\flat}{\hookrightarrow}$ k■$\not{\downarrow}$×1 : A.10,A.1
k■ $\not{\otimes}$ $\overset{\aleph_0}{\rightarrow}$ k■$\not{\downarrow}$⊗N : A.2,A.8
k■ $\not{\otimes}$ $\leftarrow$ k■$\not{\downarrow}$⊗N : A.2,A.5
k■ $\not{\otimes}$ $\nleftarrow$ k□↓× : 5.2
k■ $\not{\otimes}$ $\overset{\aleph_0}{\leftrightarrow}$ k□↓× : 5.10,A.8
k■ $\not{\otimes}$ $\nleftarrow$ k□↓⊗ : 5.2
k■ $\not{\otimes}$ $\overset{\aleph_0}{\leftrightarrow}$ k□↓⊗ : 5.10,A.8
k■ $\not{\otimes}$ $\nleftarrow$ k□$\not{\downarrow}$× : 5.2
k■ $\not{\otimes}$ $\overset{\aleph_0}{\nleftarrow}$ k□$\not{\downarrow}$× : 5.10
k■ $\not{\otimes}$ $\overset{\flat}{\rightarrow}$ k□$\not{\downarrow}$× : A.10,A.4
k■ $\not{\otimes}$ $\nleftarrow$ k□$\not{\downarrow}$⊗ : 5.2
k■ $\not{\otimes}$ $\overset{\aleph_0}{\nleftarrow}$ k□$\not{\downarrow}$⊗ : 5.10
k■ $\not{\otimes}$ $\rightarrow$ k□$\not{\downarrow}$⊗ : A.4
k■ $\not{\otimes}$ $\overset{\aleph_0}{\leftrightarrow}$ k◀↓ : 6.2,2.29,A.8
k■ $\not{\otimes}$ $\overset{\aleph_0}{\nleftarrow}$ k◀$\not{\downarrow}$: 6.2
k■ $\not{\otimes}$ $\overset{\flat}{\rightarrow}$ k◀$\not{\downarrow}$: A.10,2.29
k■ $\not{\otimes}$ $\overset{\aleph_0}{\leftrightarrow}$ k◁↓ : 2.23,5.10,A.8
k■ $\not{\otimes}$ $\overset{\aleph_0}{\nleftarrow}$ k◁$\not{\downarrow}$: 2.23,5.10
k■ $\not{\otimes}$ $\overset{\flat}{\rightarrow}$ k◁$\not{\downarrow}$: A.9,2.29,2.23
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ *perfect* : A.2,A.7
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s■↓× : A.2,A.9,A.7,7.20
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s■↓×1 : A.2,7.20,A.10,A.1,A.9,A.7
k■ $\not{\otimes}$N $\overset{\flat}{\hookrightarrow}$ s■↓×1P : A.3,A.10,A.2
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s■↓⊗ : A.2,A.7
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s■$\not{\downarrow}$× : A.2,7.20,A.9,A.7,A.10
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s■$\not{\downarrow}$×1 : A.2,A.10,A.1,7.20,A.7
k■ $\not{\otimes}$N $\overset{\aleph_0}{\nleftarrow}$ s■$\not{\downarrow}$×1P : A.2,A.10,A.3,A.8
k■ $\not{\otimes}$N $\overset{\flat}{\rightarrow}$ s■$\not{\downarrow}$×1P : A.2,A.5,A.10,A.3
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s■$\not{\downarrow}$⊗ : A.10,7.20,A.2,A.7
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s□↓× : 7.20,A.10,A.2,A.7
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s□↓⊗ : A.10,7.20,A.2,A.7
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s□$\not{\downarrow}$× : 7.20,A.10,A.2,A.7
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s□$\not{\downarrow}$⊗ : 7.20,A.10,A.2,A.7
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s◀↓ : 2.29,7.20,A.10,A.2,A.7,4.4
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s◀$\not{\downarrow}$: 2.29,7.20,A.10,A.2,A.7,4.4
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s◁↓ : A.2,A.10,2.23,7.20,A.7
k■ $\not{\otimes}$N $\overset{\aleph_0}{\leftrightarrow}$ s◁$\not{\downarrow}$: A.2,2.23,7.20,A.10,A.7
k■ $\not{\otimes}$N $\overset{\flat}{\leftrightarrow}$ k■↓× : A.10,A.2
k■ $\not{\otimes}$N $\overset{\flat}{\leftrightarrow}$ k■↓×1 : A.1,A.10,A.2
k■ $\not{\otimes}$N $\leftrightarrow$ k■↓⊗ : A.2
k■ $\not{\otimes}$N $\overset{\aleph_0}{\nleftarrow}$ k■$\not{\downarrow}$× : A.2,A.8
k■ $\not{\otimes}$N $\overset{\flat}{\rightarrow}$ k■$\not{\downarrow}$× : A.2,A.9,A.5
k■ $\not{\otimes}$N $\overset{\flat}{\leftrightarrow}$ k■$\not{\downarrow}$×N : A.2,A.10
k■ $\not{\otimes}$N $\overset{\aleph_0}{\nleftarrow}$ k■$\not{\downarrow}$×1 : A.2,A.10,A.1,A.8
k■ $\not{\otimes}$N $\overset{\flat}{\rightarrow}$ k■$\not{\downarrow}$×1 : A.2,A.5,A.10,A.1
k■ $\not{\otimes}$N $\overset{\aleph_0}{\nleftarrow}$ k■$\not{\downarrow}$⊗ : A.2,A.8
k■ $\not{\otimes}$N $\rightarrow$ k■$\not{\downarrow}$⊗ : A.2,A.5
k■ $\not{\otimes}$N $\nleftarrow$ k□↓× : A.2,5.2
k■ $\not{\otimes}$N $\overset{\aleph_0}{\nleftarrow}$ k□↓× : A.2,5.10
k■ $\not{\otimes}$N $\overset{\flat}{\rightarrow}$ k□↓× : A.2,A.10,2.29
k■ $\not{\otimes}$N $\nleftarrow$ k□↓⊗ : A.2,5.2
k■ $\not{\otimes}$N $\overset{\aleph_0}{\nleftarrow}$ k□↓⊗ : A.2,5.10
k■ $\not{\otimes}$N $\rightarrow$ k□↓⊗ : A.2,A.4
k■ $\not{\otimes}$N $\nleftarrow$ k□$\not{\downarrow}$× : A.2,5.2
k■ $\not{\otimes}$N $\overset{\aleph_0}{\nleftarrow}$ k□$\not{\downarrow}$× : A.2,5.10
k■ $\not{\otimes}$N $\overset{\flat}{\rightarrow}$ k□$\not{\downarrow}$× : A.2,A.5,A.10,A.4
k■ $\not{\otimes}$N $\nleftarrow$ k□$\not{\downarrow}$⊗ : A.2,5.2
k■ $\not{\otimes}$N $\overset{\aleph_0}{\nleftarrow}$ k□$\not{\downarrow}$⊗ : A.2,5.10
k■ $\not{\otimes}$N $\rightarrow$ k□$\not{\downarrow}$⊗ : A.2,A.5,A.4
k■ $\not{\otimes}$N $\overset{\aleph_0}{\nleftarrow}$ k◀↓ : A.2,6.2
k■ $\not{\otimes}$N $\overset{\flat}{\rightarrow}$ k◀↓ : A.2,A.9,2.29
k■ $\not{\otimes}$N $\overset{\aleph_0}{\nleftarrow}$ k◀$\not{\downarrow}$: A.2,6.2
k■ $\not{\otimes}$N $\overset{\flat}{\rightarrow}$ k◀$\not{\downarrow}$: A.2,A.5,A.10,2.29
k■ $\not{\otimes}$N $\overset{\aleph_0}{\nleftarrow}$ k◁↓ : A.2,2.23,5.10
k■ $\not{\otimes}$N $\overset{\flat}{\rightarrow}$ k◁↓ : A.2,A.10,2.29,2.23
k■ $\not{\otimes}$N $\overset{\aleph_0}{\nleftarrow}$ k◁$\not{\downarrow}$: A.2,2.23,5.10
k■ $\not{\otimes}$N $\overset{\flat}{\rightarrow}$ k◁$\not{\downarrow}$: A.2,A.5,A.9,2.29,2.23

B.8. Komplexitätstheoretische spezielle Sicherheit (k□)

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ *perfect* : A.7

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s■↓× : 7.20,A.10,A.7

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s■↓×1 : A.1,7.20,A.10,A.7

k□↓× $\overset{\text{NP}}{\rightarrow}$ s■↓×1P : A.3,5.2

k□↓× $\overset{\text{NP}}{\rightarrow}$ s■↓×1P : A.3,5.10

k□↓× $\overset{\text{NP}}{\leftarrow}$ s■↓×1P : A.3,A.4

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s■↓⊗ : A.6,A.7,A.9,7.20

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s■↓̸× : A.10,7.20,A.7,A.9

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s■↓̸×1 : 7.20,A.10,A.1,A.6,A.7,A.9

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s■↓̸×1P : A.10,A.3,A.8,5.10

k□↓× $\overset{\text{NP}}{\rightarrow}$ s■↓̸×1P : A.3,5.2

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s■↓̸⊗ : 7.20,A.7,A.10,A.9

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s□↓× : A.7

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s□↓⊗ : 7.20,A.7,A.9

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s□↓̸× : 7.20,A.7,A.10,A.9

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s□↓̸⊗ : 7.20,A.7,A.10,A.9

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s◀↓ : A.10,4.4,7.20,A.7,2.29

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s◀↓̸ : 7.20,4.4,A.7,2.29,A.10,A.9

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s◁↓ : A.10,2.23,7.20,A.7

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ s◁↓̸ : 2.23,7.20,A.7,A.10,A.9

k□↓× $\overset{\text{NP}}{\rightarrow}$ k■↓× : 5.2

k□↓× $\overset{\text{NP}}{\rightarrow}$ k■↓× : 5.10

k□↓× $\overset{\text{NP}}{\leftarrow}$ k■↓× : A.4

k□↓× $\overset{\text{NP}}{\rightarrow}$ k■↓×1 : A.1,5.2

k□↓× $\overset{\text{NP}}{\rightarrow}$ k■↓×1 : A.1,5.10

k□↓× $\overset{\text{NP}}{\leftarrow}$ k■↓×1 : A.1,A.4

k□↓× $\overset{\text{NP}}{\rightarrow}$ k■↓⊗ : 5.2

k□↓× $\overset{\text{NP}}{\rightarrow}$ k■↓⊗ : 5.10

k□↓× $\overset{\text{P}}{\leftarrow}$ k■↓⊗ : A.10,2.29

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ k■↓̸× : A.8,5.10

k□↓× $\overset{\text{NP}}{\rightarrow}$ k■↓̸× : 5.2

k□↓× $\overset{\text{NP}}{\rightarrow}$ k■↓̸×N : A.2,5.2

k□↓× $\overset{\text{NP}}{\rightarrow}$ k■↓̸×N : A.2,5.10

k□↓× $\overset{\text{NP}}{\leftarrow}$ k■↓̸×N : A.2,A.4

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ k■↓̸×1 : A.10,A.1,A.8,5.10

k□↓× $\overset{\text{NP}}{\rightarrow}$ k■↓̸×1 : A.1,5.2

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ k■↓̸⊗ : A.8,5.10

k□↓× $\overset{\text{NP}}{\rightarrow}$ k■↓̸⊗ : 5.2

k□↓× $\overset{\text{NP}}{\rightarrow}$ k■↓̸⊗N : A.2,5.2

k□↓× $\overset{\text{NP}}{\rightarrow}$ k■↓̸⊗N : A.2,5.10

k□↓× $\overset{\text{P}}{\leftarrow}$ k■↓̸⊗N : A.2,A.10,2.29

k□↓× $\overset{\text{P}}{\leftrightarrow}$ k□↓⊗ : A.10

k□↓× $\overset{\text{NP}}{\leftarrow}$ k□↓̸× : A.8

k□↓× $\rightarrow$ k□↓̸× : A.5

k□↓× $\overset{\text{NP}}{\leftarrow}$ k□↓̸⊗ : A.8

k□↓× $\overset{\text{P}}{\rightarrow}$ k□↓̸⊗ : A.10,A.5

k□↓× $\overset{\text{NP}}{\rightarrow}$ k◀↓ : 6.16

k□↓× $\overset{\text{P}}{\leftarrow}$ k◀↓ : 2.29

k□↓× $\overset{\text{NP}}{\leftrightarrow}$ k◀↓̸ : 6.16,2.29,5.10,A.8

k□↓× $\overset{\text{P}}{\leftrightarrow}$ k◁↓ : 2.23

k□↓× $\overset{\text{NP}}{\leftarrow}$ k◁↓̸ : 2.23,A.8

k□↓× $\overset{\text{P}}{\rightarrow}$ k◁↓̸ : A.10,A.5,2.23

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ *perfect* : A.7

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s■↓× : 7.20,A.7,A.10,A.9

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s■↓×1 : A.1,7.20,A.10,A.9,A.7

k□↓⊗ $\overset{\text{NP}}{\rightarrow}$ s■↓×1P : A.3,5.2

k□↓⊗ $\overset{\text{NP}}{\rightarrow}$ s■↓×1P : A.3,5.10

k□↓⊗ $\overset{\text{P}}{\leftarrow}$ s■↓×1P : A.3,2.29,A.10

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s■↓⊗ : 7.20,A.7,A.6

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s■↓̸× : A.10,7.20,A.9,A.7

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s■↓̸×1 : 7.20,A.10,A.1,A.6,A.7

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s■↓̸×1P : A.10,A.3,A.8,5.10

k□↓⊗ $\overset{\text{NP}}{\rightarrow}$ s■↓̸×1P : A.3,5.2

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s■↓̸⊗ : 7.20,A.9,A.7,A.10

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s□↓× : A.9,A.7,7.20

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s□↓⊗ : A.7

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s□↓̸× : 7.20,A.9,A.7,A.10

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s□↓̸⊗ : 7.20,A.9,A.7,A.10

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s◀↓ : A.10,4.4,A.7,2.29,7.20

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s◀↓̸ : 7.20,4.4,A.9,A.7,2.29,A.10

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s◁↓ : A.10,2.23,A.7,7.20

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ s◁↓̸ : 2.23,7.20,A.9,A.7,A.10

k□↓⊗ $\overset{\text{NP}}{\rightarrow}$ k■↓× : 5.2

k□↓⊗ $\overset{\text{NP}}{\rightarrow}$ k■↓× : 5.10

k□↓⊗ $\overset{\text{P}}{\leftarrow}$ k■↓× : 2.29,A.10

k□↓⊗ $\overset{\text{NP}}{\rightarrow}$ k■↓×1 : A.1,5.2

k□↓⊗ $\overset{\text{NP}}{\rightarrow}$ k■↓×1 : A.1,5.10

k□↓⊗ $\overset{\text{P}}{\leftarrow}$ k■↓×1 : A.1,2.29,A.10

k□↓⊗ $\overset{\text{NP}}{\rightarrow}$ k■↓⊗ : 5.2

k□↓⊗ $\overset{\text{NP}}{\rightarrow}$ k■↓⊗ : 5.10

k□↓⊗ $\overset{\text{NP}}{\leftarrow}$ k■↓⊗ : A.4

k□↓⊗ $\overset{\text{NP}}{\leftrightarrow}$ k■↓̸× : A.8,5.10

k□↓⊗ $\overset{\text{NP}}{\rightarrow}$ k■↓̸× : 5.2

k□↓⊗ $\nrightarrow$ k■⊀×N : A.2,5.2
k□↓⊗ $\overset{\aleph_0}{\nrightarrow}$ k■⊀×N : A.2,5.10
k□↓⊗ $\overset{\flat}{\nleftarrow}$ k■⊀×N : A.2,2.29,A.10
k□↓⊗ $\overset{\aleph_0}{\nleftrightarrow}$ k■⊀×1 : A.10,A.1,A.8,5.10
k□↓⊗ $\nrightarrow$ k■⊀×1 : A.1,5.2
k□↓⊗ $\overset{\aleph_0}{\nleftrightarrow}$ k■⊀⊗ : A.8,5.10
k□↓⊗ $\nrightarrow$ k■⊀⊗ : 5.2
k□↓⊗ $\nrightarrow$ k■⊀⊗N : A.2,5.2
k□↓⊗ $\overset{\aleph_0}{\nrightarrow}$ k■⊀⊗N : A.2,5.10
k□↓⊗ $\leftarrow$ k■⊀⊗N : A.2,A.4
k□↓⊗ $\overset{\flat}{\leftrightarrow}$ k□↓× : A.10
k□↓⊗ $\overset{\aleph_0}{\nleftarrow}$ k□⊀× : A.8
k□↓⊗ $\overset{\flat}{\rightarrow}$ k□⊀× : A.9,A.5
k□↓⊗ $\overset{\aleph_0}{\nleftarrow}$ k□⊀⊗ : A.8
k□↓⊗ $\rightarrow$ k□⊀⊗ : A.5
k□↓⊗ $\overset{\aleph_0}{\nrightarrow}$ k◄↓ : 6.16
k□↓⊗ $\overset{\flat}{\leftarrow}$ k◄↓ : 2.29,A.10
k□↓⊗ $\overset{\aleph_0}{\nleftrightarrow}$ k◄⊀ : 6.16,2.29,5.10,A.8
k□↓⊗ $\overset{\flat}{\leftrightarrow}$ k◁↓ : 2.23,A.10,A.9
k□↓⊗ $\overset{\aleph_0}{\nleftarrow}$ k◁⊀ : 2.23,A.8
k□↓⊗ $\overset{\flat}{\rightarrow}$ k◁⊀ : A.5,A.10,2.23
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ *perfect* : A.7
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s■↓× : 7.20,A.7,A.10
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s■↓×1 : A.1,7.20,A.10,A.7
k□⊀× $\nrightarrow$ s■↓×1P : A.3,5.2
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ s■↓×1P : A.3,5.10
k□⊀× $\leftarrow$ s■↓×1P : A.3,A.5,A.4
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s■↓⊗ : A.6,A.7,7.20,A.10
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s■⊀× : 7.20,A.7,A.10
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s■⊀×1 : 7.20,A.10,A.1,A.6,A.7
k□⊀× $\nrightarrow$ s■⊀×1P : A.3,5.2
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ s■⊀×1P : A.3,5.10
k□⊀× $\leftarrow$ s■⊀×1P : A.3,A.4
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s■⊀⊗ : 7.20,A.7,A.10
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s□↓× : 7.20,A.7,A.10,A.6
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s□↓⊗ : A.6,A.7,7.20,A.10
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s□⊀× : A.7
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s□⊀⊗ : 7.20,A.7,A.10
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s◄↓ : A.10,4.4,A.6,A.7,2.29,7.20
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s◄⊀ : 7.20,A.10,4.4,A.6,A.7,2.29
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s◁↓ : A.10,2.23,A.6,A.7,7.20
k□⊀× $\overset{\aleph_0}{\nleftrightarrow}$ s◁⊀ : 2.23,A.7
k□⊀× $\nrightarrow$ k■↓× : 5.2
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ k■↓× : 5.10
k□⊀× $\leftarrow$ k■↓× : A.5,A.4
k□⊀× $\nrightarrow$ k■↓×1 : A.1,5.2
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ k■↓×1 : A.1,5.10
k□⊀× $\leftarrow$ k■↓×1 : A.1,A.5,A.4
k□⊀× $\nrightarrow$ k■↓⊗ : 5.2
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ k■↓⊗ : 5.10
k□⊀× $\overset{\flat}{\leftarrow}$ k■↓⊗ : A.5,A.10,A.4
k□⊀× $\nrightarrow$ k■⊀× : 5.2
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ k■⊀× : 5.10
k□⊀× $\leftarrow$ k■⊀× : A.4
k□⊀× $\nrightarrow$ k■⊀×N : A.2,5.2
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ k■⊀×N : A.2,5.10
k□⊀× $\leftarrow$ k■⊀×N : A.2,A.5,A.4
k□⊀× $\nrightarrow$ k■⊀×1 : A.1,5.2
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ k■⊀×1 : A.1,5.10
k□⊀× $\leftarrow$ k■⊀×1 : A.1,A.4
k□⊀× $\nrightarrow$ k■⊀⊗ : 5.2
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ k■⊀⊗ : 5.10
k□⊀× $\overset{\flat}{\leftarrow}$ k■⊀⊗ : A.10,A.4
k□⊀× $\nrightarrow$ k■⊀⊗N : A.2,5.2
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ k■⊀⊗N : A.2,5.10
k□⊀× $\overset{\flat}{\leftarrow}$ k■⊀⊗N : A.2,A.5,A.10,A.4
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ k□↓× : A.8
k□⊀× $\leftarrow$ k□↓× : A.5
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ k□↓⊗ : A.8
k□⊀× $\overset{\flat}{\leftarrow}$ k□↓⊗ : A.9,A.5
k□⊀× $\overset{\flat}{\leftrightarrow}$ k□⊀⊗ : A.10
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ k◄↓ : 6.16
k□⊀× $\overset{\flat}{\leftarrow}$ k◄↓ : 2.29,A.5
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ k◄⊀ : 6.16
k□⊀× $\overset{\flat}{\leftarrow}$ k◄⊀ : 2.29
k□⊀× $\overset{\aleph_0}{\nrightarrow}$ k◁↓ : 2.23,A.8
k□⊀× $\overset{\flat}{\leftarrow}$ k◁↓ : 2.23,A.5
k□⊀× $\overset{\flat}{\leftrightarrow}$ k◁⊀ : 2.23
k□⊀⊗ $\overset{\aleph_0}{\nleftrightarrow}$ *perfect* : A.7
k□⊀⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s■↓× : 7.20,A.10,A.7
k□⊀⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s■↓×1 : A.1,7.20,A.10,A.7
k□⊀⊗ $\nrightarrow$ s■↓×1P : A.3,5.2
k□⊀⊗ $\overset{\aleph_0}{\nrightarrow}$ s■↓×1P : A.3,5.10
k□⊀⊗ $\overset{\flat}{\leftarrow}$ s■↓×1P : A.3,2.29,A.10,A.5
k□⊀⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s■↓⊗ : 7.20,A.7,A.6
k□⊀⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s■⊀× : A.10,7.20,A.7
k□⊀⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s■⊀×1 : 7.20,A.10,A.1,A.6,A.7
k□⊀⊗ $\nrightarrow$ s■⊀×1P : A.3,5.2

k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ s■⊬×1P : A.3,5.10
k□⊬⊗ $\overset{\flat}{\leftarrow}$ s■⊬×1P : A.3,A.10,A.4
k□⊬⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s■⊬⊗ : 7.20,A.7
k□⊬⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s□↓× : 7.20,A.10,A.6,A.7
k□⊬⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s□↓⊗ : 7.20,A.7,A.6
k□⊬⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s□⊬× : A.10,A.7,7.20
k□⊬⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s□⊬⊗ : A.7
k□⊬⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s◄↓ : A.10,4.4,A.6,A.7,2.29,7.20
k□⊬⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s◄⊬ : 7.20,A.10,4.4,A.6,A.7,2.29
k□⊬⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s◁↓ : A.10,2.23,A.6,A.7,7.20
k□⊬⊗ $\overset{\aleph_0}{\nleftrightarrow}$ s◁⊬ : 2.23,A.10,A.7,7.20
k□⊬⊗ ↛ k■↓× : 5.2
k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ k■↓× : 5.10
k□⊬⊗ $\overset{\flat}{\leftarrow}$ k■↓× : 2.29,A.10,A.5
k□⊬⊗ ↛ k■↓×1 : A.1,5.2
k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ k■↓×1 : A.1,5.10
k□⊬⊗ $\overset{\flat}{\leftarrow}$ k■↓×1 : A.1,2.29,A.10,A.5
k□⊬⊗ ↛ k■↓⊗ : 5.2
k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ k■↓⊗ : 5.10
k□⊬⊗ ← k■↓⊗ : A.5,A.4
k□⊬⊗ ↛ k■⊬× : 5.2
k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ k■⊬× : 5.10
k□⊬⊗ $\overset{\flat}{\leftarrow}$ k■⊬× : A.10,A.4
k□⊬⊗ ↛ k■⊬×N : A.2,5.2
k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ k■⊬×N : A.2,5.10
k□⊬⊗ $\overset{\flat}{\leftarrow}$ k■⊬×N : A.2,2.29,A.10,A.5
k□⊬⊗ ↛ k■⊬×1 : A.1,5.2
k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ k■⊬×1 : A.1,5.10
k□⊬⊗ $\overset{\flat}{\leftarrow}$ k■⊬×1 : A.1,A.10,A.4
k□⊬⊗ ↛ k■⊬⊗ : 5.2
k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ k■⊬⊗ : 5.10
k□⊬⊗ ← k■⊬⊗ : A.4
k□⊬⊗ ↛ k■⊬⊗N : A.2,5.2
k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ k■⊬⊗N : A.2,5.10
k□⊬⊗ ← k■⊬⊗N : A.2,A.5,A.4
k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ k□↓× : A.8
k□⊬⊗ $\overset{\flat}{\leftarrow}$ k□↓× : A.10,A.5
k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ k□↓⊗ : A.8
k□⊬⊗ ← k□↓⊗ : A.5
k□⊬⊗ $\overset{\flat}{\hookrightarrow}$ k□⊬× : A.10
k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ k◄↓ : 6.16
k□⊬⊗ $\overset{\flat}{\leftarrow}$ k◄↓ : 2.29,A.10,A.5
k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ k◄⊬ : 6.16
k□⊬⊗ $\overset{\flat}{\leftarrow}$ k◄⊬ : 2.29,A.10
k□⊬⊗ $\overset{\aleph_0}{\nrightarrow}$ k◁↓ : 2.23,A.8
k□⊬⊗ $\overset{\flat}{\leftarrow}$ k◁↓ : 2.23,A.10,A.5
k□⊬⊗ $\overset{\flat}{\hookrightarrow}$ k◁⊬ : 2.23,A.10

B.9. Komplexitätstheoretische allgemeine Komponierbarkeit (k◄)

k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ *perfect* : 2.29,A.7
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s■↓× : 7.20,A.10,2.29,A.7
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s■↓×1 : A.1,7.20,A.10,2.29,A.7
k◄↓ $\overset{\aleph_0}{\nrightarrow}$ s■↓×1P : A.3,6.2
k◄↓ $\overset{\flat}{\leftarrow}$ s■↓×1P : A.3,2.29
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s■↓⊗ : 2.29,A.9,A.7,A.6
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s■⊬× : A.10,7.20,2.29,A.7,A.9
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s■⊬×1 : 7.20,A.10,A.1,A.6,2.29,A.7,A.9
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s■⊬×1P : A.10,A.3,6.2,2.29,A.8
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s■⊬⊗ : 7.20,2.29,A.7,A.10,A.9
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s□↓× : 2.29,A.7,7.20,A.10,A.9
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s□↓⊗ : 2.29,7.20,A.7,A.10,A.9
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s□⊬× : 2.29,7.20,A.7,A.10,A.9
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s□⊬⊗ : 2.29,7.20,A.7,A.10,A.9
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s◄↓ : A.10,4.4,2.29,7.20,A.7,A.9
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s◄⊬ : 7.20,4.4,2.29,A.7,A.10,A.9
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s◁↓ : A.10,2.23,2.29,7.20,A.7,A.9
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ s◁⊬ : 2.23,2.29,7.20,A.7,A.10,A.9
k◄↓ $\overset{\aleph_0}{\nrightarrow}$ k■↓× : 6.2
k◄↓ $\overset{\flat}{\leftarrow}$ k■↓× : 2.29
k◄↓ $\overset{\aleph_0}{\nrightarrow}$ k■↓×1 : A.1,6.2
k◄↓ $\overset{\flat}{\leftarrow}$ k■↓×1 : A.1,2.29
k◄↓ $\overset{\aleph_0}{\nrightarrow}$ k■↓⊗ : 6.2
k◄↓ $\overset{\flat}{\leftarrow}$ k■↓⊗ : A.9,2.29
k◄↓ $\overset{\aleph_0}{\nleftrightarrow}$ k■⊬× : 2.29,5.10,A.8,6.2
k◄↓ $\overset{\aleph_0}{\nrightarrow}$ k■⊬×N : A.2,6.2
k◄↓ $\overset{\flat}{\leftarrow}$ k■⊬×N : A.2,2.29

k◀↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k■↗×1 : A.10,A.1,6.2,2.29,A.8
k◀↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k■↗⊗ : 6.2,2.29,A.8
k◀↓ $\overset{\aleph_0^\flat}{\twoheadrightarrow}$ k■↗⊗N : A.2,6.2
k◀↓ $\overset{\flat}{\leftarrow}$ k■↗⊗N : A.2,A.9,2.29
k◀↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□↓× : 6.16
k◀↓ $\overset{\flat}{\rightarrow}$ k□↓× : 2.29
k◀↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□↓⊗ : 6.16
k◀↓ $\overset{\flat}{\rightarrow}$ k□↓⊗ : 2.29,A.10
k◀↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□↗× : 6.16
k◀↓ $\overset{\flat}{\rightarrow}$ k□↗× : 2.29,A.5
k◀↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□↗⊗ : 6.16
k◀↓ $\overset{\flat}{\rightarrow}$ k□↗⊗ : 2.29,A.10,A.5
k◀↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k◀↗ : 2.29,5.10,A.8
k◀↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k◁↓ : 2.23,6.16
k◀↓ $\overset{\flat}{\rightarrow}$ k◁↓ : 2.29,2.23
k◀↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k◁↗ : 2.23,6.16
k◀↓ $\overset{\flat}{\rightarrow}$ k◁↗ : 2.29,A.10,A.5,2.23
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ *perfect* : 2.29,A.7
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↓× : 7.20,A.10,2.29,A.7
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↓×1 : A.1,7.20,A.10,2.29,A.7
k◀↗ $\overset{\aleph_0^\flat}{\twoheadrightarrow}$ s■↓×1P : A.3,6.2
k◀↗ $\overset{\flat}{\leftarrow}$ s■↓×1P : A.3,A.5,2.29
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↓⊗ : A.6,2.29,A.7,A.10,7.20
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↗× : A.10,7.20,2.29,A.7
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↗×1 : 7.20,A.10,A.1,A.6,2.29,A.7
k◀↗ $\overset{\aleph_0^\flat}{\twoheadrightarrow}$ s■↗×1P : A.3,6.2
k◀↗ $\overset{\flat}{\leftarrow}$ s■↗×1P : A.3,2.29
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↗⊗ : 2.29,A.10,A.7,7.20
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s□↓× : 7.20,A.10,A.6,2.29,A.7
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s□↓⊗ : A.6,2.29,A.7,A.10,7.20
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s□↗× : 2.29,A.7,7.20,A.10
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s□↗⊗ : 2.29,7.20,A.7,A.10
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s◀↓ : A.10,4.4,A.6,2.29,A.7,7.20
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s◀↗ : 7.20,A.10,4.4,A.6,2.29,A.7
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s◁↓ : A.10,2.23,A.6,2.29,A.7,7.20
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s◁↗ : 2.23,2.29,A.7,7.20,A.10
k◀↗ $\overset{\aleph_0^\flat}{\twoheadrightarrow}$ k■↓× : 6.2
k◀↗ $\overset{\flat}{\leftarrow}$ k■↓× : A.5,2.29
k◀↗ $\overset{\aleph_0^\flat}{\twoheadrightarrow}$ k■↓×1 : A.1,6.2
k◀↗ $\overset{\flat}{\leftarrow}$ k■↓×1 : A.1,A.5,2.29
k◀↗ $\overset{\aleph_0^\flat}{\twoheadrightarrow}$ k■↓⊗ : 6.2
k◀↗ $\overset{\flat}{\leftarrow}$ k■↓⊗ : A.5,A.10,2.29
k◀↗ $\overset{\aleph_0^\flat}{\twoheadrightarrow}$ k■↗× : 6.2
k◀↗ $\overset{\flat}{\leftarrow}$ k■↗× : 2.29
k◀↗ $\overset{\aleph_0^\flat}{\twoheadrightarrow}$ k■↗×N : A.2,6.2
k◀↗ $\overset{\flat}{\leftarrow}$ k■↗×N : A.2,A.5,2.29
k◀↗ $\overset{\aleph_0^\flat}{\twoheadrightarrow}$ k■↗×1 : A.1,6.2
k◀↗ $\overset{\flat}{\leftarrow}$ k■↗×1 : A.1,2.29
k◀↗ $\overset{\aleph_0^\flat}{\twoheadrightarrow}$ k■↗⊗ : 6.2
k◀↗ $\overset{\flat}{\leftarrow}$ k■↗⊗ : A.10,2.29
k◀↗ $\overset{\aleph_0^\flat}{\twoheadrightarrow}$ k■↗⊗N : A.2,6.2
k◀↗ $\overset{\flat}{\leftarrow}$ k■↗⊗N : A.2,A.5,A.10,2.29
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□↓× : 6.16,2.29,5.10,A.8
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□↓⊗ : 6.16,2.29,5.10,A.8
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□↗× : 6.16
k◀↗ $\overset{\flat}{\rightarrow}$ k□↗× : 2.29
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k□↗⊗ : 6.16
k◀↗ $\overset{\flat}{\rightarrow}$ k□↗⊗ : 2.29,A.10
k◀↗ $\overset{\aleph_0^\flat}{\twoheadrightarrow}$ k◀↓ : 2.29,5.10,A.8
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k◁↓ : 2.23,6.16,2.29,5.10,A.8
k◀↗ $\overset{\aleph_0^\flat}{\leftrightarrow}$ k◁↗ : 2.23,6.16
k◀↗ $\overset{\flat}{\rightarrow}$ k◁↗ : 2.29,2.23

B.10. Komplexitätstheoretische einfache Komponierbarkeit (k◁)

k◁↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ *perfect* : 2.23,A.7
k◁↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↓× : 2.23,7.20,A.10,A.7
k◁↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↓×1 : 2.23,A.1,7.20,A.10,A.7
k◁↓ $\overset{\aleph_0^\flat}{\twoheadrightarrow}$ s■↓×1P : A.3,2.23,5.10
k◁↓ $\overset{\flat}{\leftarrow}$ s■↓×1P : A.3,2.29,2.23
k◁↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↓⊗ : 2.23,A.6,A.7,A.10,7.20
k◁↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↗× : 2.23,A.10,7.20,A.7
k◁↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↗×1 : 2.23,7.20,A.10,A.1,A.6,A.7
k◁↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↗×1P : 2.23,A.10,A.3,A.8,5.10
k◁↓ $\overset{\aleph_0^\flat}{\leftrightarrow}$ s■↗⊗ : 2.23,7.20,A.7,A.10

$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} s\Box\downarrow\times$: 2.23,A.7
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} s\Box\downarrow\otimes$: A.10,2.23,A.7,7.20
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} s\Box\not\downarrow\times$: 2.23,7.20,A.7,A.10
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} s\Box\not\downarrow\otimes$: 2.23,7.20,A.7,A.10
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} s\blacktriangleleft\downarrow$: 2.23,A.10,4.4,7.20,A.7,2.29
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} s\blacktriangleleft\not\downarrow$: 2.23,7.20,4.4,A.7,2.29,A.10
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} s\lhd\downarrow$: 2.23,A.10,7.20,A.7
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} s\lhd\not\downarrow$: 2.23,7.20,A.7,A.10
$k\lhd\downarrow \overset{\aleph_b}{\rightarrow} k\blacksquare\downarrow\times$: 2.23,5.10
$k\lhd\downarrow \overset{\flat}{\leftarrow} k\blacksquare\downarrow\times$: 2.29,2.23
$k\lhd\downarrow \overset{\aleph_b}{\rightarrow} k\blacksquare\downarrow\times 1$: A.1,2.23,5.10
$k\lhd\downarrow \overset{\flat}{\leftarrow} k\blacksquare\downarrow\times 1$: A.1,2.29,2.23
$k\lhd\downarrow \overset{\aleph_b}{\rightarrow} k\blacksquare\downarrow\otimes$: 2.23,5.10
$k\lhd\downarrow \overset{\flat}{\leftarrow} k\blacksquare\downarrow\otimes$: A.10,2.29,2.23
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} k\blacksquare\not\downarrow\times$: 2.23,5.10,A.8
$k\lhd\downarrow \overset{\aleph_b}{\rightarrow} k\blacksquare\not\downarrow\times N$: A.2,2.23,5.10
$k\lhd\downarrow \overset{\flat}{\leftarrow} k\blacksquare\not\downarrow\times N$: A.2,2.29,2.23
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} k\blacksquare\not\downarrow\times 1$: 2.23,A.10,A.1,A.8,5.10
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} k\blacksquare\not\downarrow\otimes$: 2.23,5.10,A.8
$k\lhd\downarrow \overset{\aleph_b}{\rightarrow} k\blacksquare\not\downarrow\otimes N$: A.2,2.23,5.10
$k\lhd\downarrow \overset{\flat}{\leftarrow} k\blacksquare\not\downarrow\otimes N$: A.2,A.10,2.29,2.23
$k\lhd\downarrow \overset{\flat}{\leftrightarrow} k\Box\downarrow\times$: 2.23
$k\lhd\downarrow \overset{\flat}{\leftrightarrow} k\Box\downarrow\otimes$: A.9,2.23,A.10
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} k\Box\not\downarrow\times$: 2.23,A.8
$k\lhd\downarrow \overset{\flat}{\rightarrow} k\Box\not\downarrow\times$: 2.23,A.5
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} k\Box\not\downarrow\otimes$: 2.23,A.8
$k\lhd\downarrow \overset{\flat}{\rightarrow} k\Box\not\downarrow\otimes$: 2.23,A.10,A.5
$k\lhd\downarrow \overset{\aleph_b}{\rightarrow} k\blacktriangleleft\downarrow$: 2.23,6.16
$k\lhd\downarrow \overset{\flat}{\leftarrow} k\blacktriangleleft\downarrow$: 2.29,2.23
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} k\blacktriangleleft\not\downarrow$: 2.23,2.29,5.10,A.8,6.16
$k\lhd\downarrow \overset{\aleph_b}{\leftrightarrow} k\lhd\not\downarrow$: 2.23,A.8
$k\lhd\downarrow \overset{\flat}{\rightarrow} k\lhd\not\downarrow$: 2.23,A.10,A.5
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow}$ *perfect* : 2.23,A.7,A.10
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\blacksquare\downarrow\times$: 7.20,A.10,2.23,A.7
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\blacksquare\downarrow\times 1$: A.1,7.20,A.10,2.23,A.7
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} s\blacksquare\downarrow\times 1P$: A.3,2.23,5.10
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} s\blacksquare\downarrow\times 1P$: A.3,A.5,2.29,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\blacksquare\downarrow\otimes$: 2.23,7.20,A.10,A.7,A.6
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\blacksquare\not\downarrow\times$: 2.23,A.10,7.20,A.7
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\blacksquare\not\downarrow\times 1$: 2.23,7.20,A.10,A.1,A.6,A.7
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} s\blacksquare\not\downarrow\times 1P$: A.3,2.23,5.10
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} s\blacksquare\not\downarrow\times 1P$: A.3,2.29,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\blacksquare\not\downarrow\otimes$: 2.23,7.20,A.7,A.10
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\Box\downarrow\times$: 2.23,7.20,A.7,A.10,A.6
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\Box\downarrow\otimes$: 2.23,A.6,A.7,7.20,A.10
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\Box\not\downarrow\times$: 2.23,A.7
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\Box\not\downarrow\otimes$: 2.23,A.10,A.7,7.20
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\blacktriangleleft\downarrow$: 2.23,A.10,4.4,A.6,A.7,2.29,7.20
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\blacktriangleleft\not\downarrow$: 7.20,A.10,4.4,2.23,A.6,A.7,2.29
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\lhd\downarrow$: A.10,2.23,A.6,A.7,7.20
$k\lhd\not\downarrow \overset{\aleph_b}{\leftrightarrow} s\lhd\not\downarrow$: 2.23,A.7
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} k\blacksquare\downarrow\times$: 2.23,5.10
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} k\blacksquare\downarrow\times$: A.5,2.29,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} k\blacksquare\downarrow\times 1$: A.1,2.23,5.10
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} k\blacksquare\downarrow\times 1$: A.1,A.5,2.29,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} k\blacksquare\downarrow\otimes$: 2.23,5.10
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} k\blacksquare\downarrow\otimes$: A.5,A.9,2.29,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} k\blacksquare\not\downarrow\times$: 2.23,5.10
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} k\blacksquare\not\downarrow\times$: 2.29,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} k\blacksquare\not\downarrow\times N$: A.2,2.23,5.10
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} k\blacksquare\not\downarrow\times N$: A.2,A.5,2.29,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} k\blacksquare\not\downarrow\times 1$: A.1,2.23,5.10
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} k\blacksquare\not\downarrow\times 1$: A.1,2.29,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} k\blacksquare\not\downarrow\otimes$: 2.23,5.10
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} k\blacksquare\not\downarrow\otimes$: A.9,2.29,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} k\blacksquare\not\downarrow\otimes N$: A.2,2.23,5.10
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} k\blacksquare\not\downarrow\otimes N$: A.2,A.5,A.9,2.29,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} k\Box\downarrow\times$: 2.23,A.8
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} k\Box\downarrow\times$: A.10,A.5,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} k\Box\downarrow\otimes$: 2.23,A.8
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} k\Box\downarrow\otimes$: A.5,A.10,2.23
$k\lhd\not\downarrow \overset{\flat}{\leftrightarrow} k\Box\not\downarrow\times$: 2.23
$k\lhd\not\downarrow \overset{\flat}{\leftrightarrow} k\Box\not\downarrow\otimes$: A.10,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} k\blacktriangleleft\downarrow$: 2.23,6.16
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} k\blacktriangleleft\downarrow$: 2.29,A.10,A.5,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} k\blacktriangleleft\not\downarrow$: 2.23,6.16
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} k\blacktriangleleft\not\downarrow$: 2.29,2.23
$k\lhd\not\downarrow \overset{\aleph_b}{\rightarrow} k\lhd\downarrow$: 2.23,A.8
$k\lhd\not\downarrow \overset{\flat}{\leftarrow} k\lhd\downarrow$: 2.23,A.10,A.5

C. Eigene Arbeiten

Eingeladene Vorträge

[Unr04b] Dominique Unruh. Relating formal security for classical and quantum protocols. Isaac Newton Institute for Mathematical Sciences, eingeladener Vortrag, September 2004. Online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/unruh04relating.html`.

[Unr06a] Dominique Unruh. Long-term universal composability. RISC Seminar, CWI, Amsterdam, eingeladener Vortrag, März 2006. Volle Fassung ist [MQU07].

Aufsätze in Zeitschriften

[HU06a] Dennis Hofheinz und Dominique Unruh. An attack on a group-based cryptographic scheme, 2006. Erscheint im J. of Contemporary Mathematics, AMS.

[Unr06b] Dominique Unruh. Quantum programming languages. Teil einer Studie im Rahmen des BMBF-Projekts MARQUIS. *Informatik – Forschung und Entwicklung*, 21(1):55–63, 2006.

Konferenzbeiträge

[HU05a] Dennis Hofheinz und Dominique Unruh. Comparing two notions of simulatability. In Joe Kilian (Hrsg.), *Theory of Cryptography, Proceedings of TCC 2005*, Lecture Notes in Computer Science, Seiten 86–103. Springer-Verlag, 2005. Online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/hofheinz05comparing.html`.

[HMQU05a] Dennis Hofheinz, Jörn Müller-Quade und Dominique Unruh. Polynomial runtime in simulatability definitions. In *18th IEEE Computer Security Foundations Workshop, Proceedings of CSFW 2005*, Seiten 156–169. IEEE Computer Society, 2005. Online verfügbar

unter http://www.infsec.cs.uni-sb.de/~unruh/publications/hofheinz05polynomial.html.

[HMQU05b] Dennis Hofheinz, Jörn Müller-Quade und Dominique Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. In *Proceedings of the 5th Central European Conference on Cryptology, MoraviaCrypt '05*, 2005. Online verfügbar unter http://www.infsec.cs.uni-sb.de/~unruh/publications/hofheinz05universally.html.

[Unr05] Dominique Unruh. Quantum programs with classical output streams. *Electronic Notes in Theoretical Computer Science*, 2005. 3rd International Workshop on Quantum Programming Languages, online verfügbar unter http://www.infsec.cs.uni-sb.de/~unruh/publications/unruh05quantum.html.

[MQRU05] Jörn Müller-Quade, Stefan Röhrich und Dominique Unruh. Oblivious transfer is incomplete for deniable protocols. Workshop on The Past, Present and Future of Oblivious Transfer, Haifa, Mai 2005. Online verfügbar unter http://www.infsec.cs.uni-sb.de/~unruh/publications/muellerquade05oblivious.html.

[HU05b] Dennis Hofheinz und Dominique Unruh. On the notion of statistical security in simulatability definitions. In Jianying Zhou, Javier Lopez, Robert H. Deng und Feng Bao (Hrsg.), *Information Security, Proceedings of ISC '05*, Lecture Notes in Computer Science, Seiten 118–133. Springer-Verlag, 2005. Online verfügbar unter http://eprint.iacr.org/2005/032.

[BHMQU05] Michael Backes, Dennis Hofheinz, Jörn Müller-Quade und Dominique Unruh. On fairness in simulatability-based cryptographic systems. In Ralf Küsters und John Mitchell (Hrsg.), *Proceedings of the 2005 ACM Workshop on Formal Methods in Security Engineering*, Seiten 13–22. ACM Press, 2005. Volle Fassung online verfügbar unter http://eprint.iacr.org/2005/294.

[HU06b] Dennis Hofheinz und Dominique Unruh. Simulatable security and polynomially bounded concurrent composition. In *IEEE Symposium on Security and Privacy, Proceedings of SSP '06*, Seiten 169–182. IEEE Computer Society, 2006. Volle Fassung online verfügbar unter http://eprint.iacr.org/2006/130.ps.

[HMQU06a] Dennis Hofheinz, Jörn Müller-Quade und Dominique Unruh. On the (im)possibility of extending coin toss. In Serge Vaudenay

(Hrsg.), *Advances in Cryptology, Proceedings of EUROCRYPT '06*, Band 4004 von *Lecture Notes in Computer Science*, Seiten 504–521. Springer-Verlag, 2006. Volle Fassung online verfügbar unter `http://eprint.iacr.org/2006/177`.

[HMQU06b] Dennis Hofheinz, Jörn Müller-Quade und Dominique Unruh. A simple model of polynomial time UC (abstract). Workshop on Models for Cryptographic Protocols (MCP 2006), Århus, Juli 2006. Online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/hofheinz06simple.html`.

[MQU06] Jörn Müller-Quade und Dominique Unruh. Composable deniability (abstract). Workshop on Models for Cryptographic Protocols (MCP 2006), århus, Juli 2006. Online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/muellerquade06composable.html`.

[MQU07] Jörn Müller-Quade und Dominique Unruh. Long-term security and universal composability, 2007. Erscheint auf der TCC 2007, volle Fassung online verfügbar unter `http://eprint.iacr.org/2006/422`.

Verschiedenes

[Unr02] Dominique Unruh. Formal security in quantum cryptology. Studienarbeit, Institut für Algorithmen und Kognitive Systeme, Universität Karlsruhe, Dezember 2002. Online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/unruh02formal.html`.

[Unr03] Dominique Unruh. Zufallsextraktoren für Quellen variierender Qualität. Diplomarbeit, Institut für Algorithmen und Kognitive Systeme, Universität Karlsruhe, Juli 2003. Online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/unruh03zufallsextraktoren.html`.

[Unr04a] Dominique Unruh. Classical control in quantum programs. 5th European QIPC Workshop, Roma, September 2004. Poster, online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/unruh04classical.html`.

[Unr04c] Dominique Unruh. Simulatable security for quantum protocols, September 2004. Online verfügbar unter `http://arxiv.org/ps/quant-ph/0409125`.

[Unr06c] Dominique Unruh. Relations among statistical security notions or why exponential adversaries are unlimited, 2006. Online verfügbar unter http://eprint.iacr.org/2005/406.

D. Lebenslauf

Name:	Dominique Unruh
Familienstand:	verheiratet
Geburt:	17. März 1979
Mutter:	Dr. med. Catherine Bonneau
Vater:	Karl-Heinz Unruh

Schule

1985–1989	Bonifatiusschule, Hannover (Grundschule)
1989–1991	Orientierungsstufe Edenstraße, Hannover
1991–1993	Kaiser-Wilhelm-Gymnasium, Hannover
1993–1997	Hochbegabtenzweig der Christophorusschule Braunschweig (Internatsgymnasium)
1997	Abitur mit 1,7

Interludium

1997–1998	Grundwehrdienst bei der Marine (Bremerhaven, Flensburg)
1998	Dreimonatige Beschäftigung in der EDV-Abteilung des Vincentz-Verlags Hannover

Studium

1998–2003	Studium der Informatik an der Universität Karlsruhe
2000	Vordiplom der Informatik mit 1,0
2001	Vordiplom der Mathematik mit 1,2
2003	Diplom der Informatik mit Auszeichnung (1,0)

Beruf

2003–2006	Beschäftigung als wissenschaftlicher Mitarbeiter am Institut für Algorithmen und Kognitive Systeme, Fakultät für Informatik, Universität Karlsruhe (TH)
ab 2006	Beschäftigung als wissenschaftlicher Mitarbeiter an der Information Security and Cryptography Group, Fakultät für Informatik, Universität des Saarlandes

E. Abbildungsverzeichnis

F. Englische Fachausdrücke

allgemeine Komponierbarkeit	universal composability polynomially-bounded general composability
allgemeine Sicherheit	universal simulatability universal composability
Angreifer	adversary
Ausgang	outcome (Spieltheorie)
Dummy-Angreifer	dummy adversary
Einweg-Funktion	one-way function
Extensivform	extensive form (Spieltheorie)
Funktionalität	functionality
Geheimhaltung	privacy secrecy
gemischte Strategie	mixed strategy (Spieltheorie)
Gewinnfunktion	payoff function (Spieltheorie)
Informationsmenge	information set (Spieltheorie)
kollisionsresistente Hashfunktion	collision-resistant hash(ing) function collision-free hash(ing) function
komplexitätstheoretisch ununterscheidbar	computationally indistinguishable
komplexitätstheoretische Sicherheit	computational security
Korrektheit	correctness
nebenläufige Komposition	concurrent composition
Normalform	normal form (Spieltheorie)
Nullsummenspiel	zero-sum game (Spieltheorie)
parallele Komposition	parallel composition
perfekte Erinnerung	perfect recall (Spieltheorie)
perfekte Sicherheit	perfect security
reine Strategie	pure strategy (Spieltheorie)
sequentielle Komposition	sequential composition
sichere Funktionsauswertung	secure function evaluation secure multiparty computation
Sicherheit mit beschränktem Risiko	security with bounded risk

Sicht	view
Simulator	simulator
spezielle Komponierbarkeit	etwa: $O(1)$-bounded general composability
spezielle Sicherheit	general simulatability specialised-simulator UC
Spielbaum	game tree (Spieltheorie)
statistisch ununterscheidbar	statistically indistinguishable
statistische Sicherheit	statistical security information-theoretical security
statistischer Abstand	statistical distance total variation distance
überwältigend	overwhelming
Umgebung	environment honest user
Unterscheider	distinguisher
Verhaltensstrategie	behaviour strategy (Spieltheorie)
vernachlässigbar	negligible
Vollständigkeit	completeness
Zero-Knowledge-Beweis	zero-knowledge proof
Zufallsknoten	chance node nature's choice (Spieltheorie)

G. Satz- und Definitionsverzeichnis

H. Symbolverzeichnis

$\mathbb{N}$	Natürliche Zahlen ohne 0	15
$\mathbb{N}_0$	Natürliche Zahlen mit 0	15
Σ	Ein (in dieser Arbeit festes) Alphabet	15
Σ^*	Wörter über Σ	15
λ	Das leere Wort	15
Σ^+	Nichtleere Wörter über Σ	15
$POLY$	Polynomiell-beschränkte Funktionen	15
EXP	Exponentiell-beschränkte Funktionen	15
$\langle A, B \rangle$	Ausgabe der ITM B nach Interaktion mit ITM A	15
Δ	Statistischer Abstand	16
π	Meist: das reale Protokoll	40
ρ	Meist: das ideale Protokoll	40
$\mathcal{A}$	Meist: der Angreifer	40
$\mathcal{S}$	Meist: der Simulator	41
$\mathcal{Z}$	Meist: die Umgebung	41
$\mathcal{F}$	Meist: die ideale Funktionalität	45
`clk`	Der Spezialport des Schedulers, über den dieser aktiviert wird	49
`output`	Spezialport, über den Ausgabe am Protokollende ausgegeben wird	50
`input`	Spezialport, über den eine Maschine zu Protokollbeginn Eingabe erhält	50
run	Folge aller Aktivierungen während eines Protokollaufs	51
OUTPUT	Ausgabe der Umgebung nach einem Protokollauf	54
VIEW	Sicht der Umgebung	54
CVIEW	Komplexitätstheoretische Sicht der Umgebung	62
σ^π	Einfache Komposition der Protokolle σ und π	73
$\geq$	Oft: Kurzschreibweise für „so sicher wie“	74
$f \cdot \pi$	f-fache nebenläufige Komposition des Protokolls π	85
$\mathcal{Z}_?$	Zufällige Umgebung	99
pfx_n	Präfix der Länge n	103
$\mathcal{Z}_{out}$	Zufällig ausgebende Umgebung	114
F	In Kapitel 6: die reaktive Funktion aus Definition 6.4	185
P	In Kapitel 6: die sichere Funktionsauswertung für F	190

$\mathfrak{S}_i$	Menge der reinen Strategien von Spieler i	211
$\mathfrak{M}_i$	Menge der gemischten Strategien von Spieler i	215
$\mathfrak{B}_i$	Menge der Verhaltensstrategien von Spieler i	224
G^I	Das ideale Spiel	228
G^R	Das reale Spiel	229
G^C	Das kombinierte Spiel	232

I. Literaturverzeichnis

[Bac02a] Adam Back. Hashcash – a denial of service counter-measure, August 2002. Online verfügbar unter `http://www.cypherspace.org/hashcash/hashcash.pdf`.

[Bac02b] Michael Backes. *Cryptographically Sound Analysis of Security Protocols*. Dissertation, Universität des Saarlandes, 2002. Online verfügbar unter `http://www.infsec.cs.uni-sb.de/~backes/papers/PhDthesis.ps.gz`.

[Bar02] Boaz Barak. A probabilistic-time hierarchy theorem for "slightly non-uniform" algorithms. In José D.P. Rolim und Salil P. Vadhan (Hrsg.), *Randomization and Approximation Techniques, Proceedings of RANDOM 2002*, Nummer 2483 in Lecture Notes in Computer Science, Seiten 194–208. Springer-Verlag, 2002. Online verfügbar unter `http://www.cs.princeton.edu/~boaz/Papers/bptime.ps`.

[Bau92] Heinz Bauer. *Maß- und Integrationstheorie*. Walter de Gruyter, Berlin, zweite Auflage, 1992.

[Bau02] Heinz Bauer. *Wahrscheinlichkeitstheorie*. Walter de Gruyter, Berlin, fünfte Auflage, 2002.

[BB84] Charles H. Bennett und Gilles Brassard. Quantum cryptography: Public-key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing 1984*, Seiten 175–179. IEEE Computer Society, 1984.

[BDHK06] Michael Backes, Markus Dürmuth, Dennis Hofheinz und Ralf Küsters. Conditional reactive simulatability. In Dieter Gollmann, Jan Meier und Andrei Sabelfeld (Hrsg.), *Computer Security, Proceedings of ESORICS 2006*, Band 4189 von *Lecture Notes in Computer Science*. Springer-Verlag, 2006. Online verfügbar unter `http://eprint.iacr.org/2006/132`.

[BDPR98] Mihir Bellare, Amit Desai, David Pointcheval und Phillip Rogaway. Relations among notions of security for public-key encryption sche-

mes. In Hugo Krawczyk (Hrsg.), *Advances in Cryptology, Proceedings of CRYPTO '98*, Band 1462 von *Lecture Notes in Computer Science*, Seiten 26–45. Springer-Verlag, 1998. Erweiterte Fassung online verfügbar unter `http://eprint.iacr.org/1998/021.ps`.

[Bea92] Donald Beaver. Foundations of secure interactive computing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, Seiten 377–391. Springer-Verlag, 1992.

[BG02] Boaz Barak und Oded Goldreich. Universal arguments and their applications. In *17th Annual IEEE Conference on Computational Complexity, Proceedings of CCC'02*, Seiten 194–203. IEEE Computer Society, 2002. Online verfügbar unter `http://www.cs.princeton.edu/~boaz/Papers/uargs.ps`.

[BH04] Michael Backes und Dennis Hofheinz. How to break and repair a universally composable signature functionality. In Kan Zhang und Yuliang Zheng (Hrsg.), *Information Security, Proceedings of ISC '04*, Lecture Notes in Computer Science, Seiten 61–72. Springer-Verlag, 2004. Online verfügbar unter `http://eprint.iacr.org/2003/240.ps`.

[BHMQU05] Michael Backes, Dennis Hofheinz, Jörn Müller-Quade und Dominique Unruh. On fairness in simulatability-based cryptographic systems. In Ralf Küsters und John Mitchell (Hrsg.), *Proceedings of the 2005 ACM Workshop on Formal Methods in Security Engineering*, Seiten 13–22. ACM Press, 2005. Volle Fassung online verfügbar unter `http://eprint.iacr.org/2005/294`.

[BOGG+90] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali und P. Rogaway. Everything provable ist provable in zero-knowledge. In Shafi Goldwasser (Hrsg.), *Advances in Cryptology, Proceedings of CRYPTO '88*, Band 293 von *Lecture Notes in Computer Science*, Seiten 37–56. Springer-Verlag, 1990.

[BOGW88] Michael Ben-Or, Shafi Goldwasser und Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Twentieth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1988*, Seiten 1–10. ACM Press, 1988.

[BOM04] M. Ben-Or und D. Mayers. General security definition and composability for quantum & classical protocols, September 2004. Online verfügbar unter `http://xxx.lanl.gov/abs/quant-ph/0409062`.

[BPW04a] Michael Backes, Birgit Pfitzmann und Michael Waidner. A general composition theorem for secure reactive systems. In Moni Naor (Hrsg.), *Theory of Cryptography, Proceedings of TCC 2004*, Band 2951 von *Lecture Notes in Computer Science*, Seiten 336–354. Springer-Verlag, 2004. Online verfügbar unter `http://www.zurich.ibm.com/security/publications/2004/BaPfWa2004MoreGeneralComposition.pdf`.

[BPW04b] Michael Backes, Birgit Pfitzmann und Michael Waidner. Secure asynchronous reactive systems. IACR ePrint Archive, März 2004. Online verfügbar unter `http://eprint.iacr.org/2004/082.ps`.

[BR95] Mihir Bellare und Phillip Rogaway. Optimal asymmetric encryption—how to encrypt with RSA. In Alfredo de Santis (Hrsg.), *Advances in Cryptology, Proceedings of EUROCRYPT '94*, Band 950 von *Lecture Notes in Computer Science*, Seiten 92–111. Springer-Verlag, 1995. Erweiterte Fassung online verfügbar unter `http://www.cs.ucsd.edu/users/mihir/papers/oae.ps`.

[BS05] Boaz Barak und Amit Sahai. How to play almost any mental game over the net – concurrent composition via super-polynomial simulation. In *46th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2005*, Seiten 543–552. IEEE Computer Society, 2005. Erweiterte Fassung online verfügbar unter `http://eprint.iacr.org/2005/106`.

[Can95] Ran Canetti. *Studies in Secure Multiparty Computation and Applications*. Dissertation, Feinberg Graduate School of the Weizmann Institute of Science, Rehovot, Juni 1995. Online verfügbar unter `http://www.wisdom.weizmann.ac.il/~oded/ran-phd.html`.

[Can00] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000. Volle Fassung online verfügbar unter `http://eprint.iacr.org/1998/018.ps`.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, Seiten 136–145. IEEE Computer Society, 2001. Volle Fassung online verfügbar unter `http://www.eccc.uni-trier.de/eccc-reports/2001/TR01-016/revisn01.ps`.

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. IACR ePrint Archive, Dezember 2005. Volle und überarbeitete Fassung von [Can01], online verfügbar unter `http://eprint.iacr.org/2000/067.ps`.

[CCD88] David Chaum, Claude Crépeau und Ivan Damgård. Multiparty unconditionally secure protocols. In *Twentieth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1988*, Seiten 11–19. ACM Press, 1988. Online verfügbar unter `http://www.cs.mcgill.ca/~crepeau/GZIP/CCD88.ps.gz`.

[CF01] Ran Canetti und Marc Fischlin. Universally composable commitments. In Joe Kilian (Hrsg.), *Advances in Cryptology, Proceedings of CRYPTO '01*, Band 2139 von *Lecture Notes in Computer Science*, Seiten 19–40. Springer-Verlag, 2001. Volle Fassung online verfügbar unter `http://eprint.iacr.org/2001/055.ps`.

[CGH98] Ran Canetti, Oded Goldreich und Shai Halevi. The random oracle methodology, revisited. In *Thirtieth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1998*, Seiten 209–218. ACM Press, 1998. Vorläufige Version, erweiterte Fassung online verfügbar unter `http://eprint.iacr.org/1998/011.ps`.

[CKL03] Ran Canetti, Eyal Kushilevitz und Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham (Hrsg.), *Advances in Cryptology, Proceedings of EUROCRYPT '03*, Band 2656 von *Lecture Notes in Computer Science*, Seiten 68–86. Springer-Verlag, 2003. Volle Fassung online verfügbar unter `http://eprint.iacr.org/2004/116.ps`.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky und Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2002*, Seiten 494–503. ACM Press, 2002. Erweitertes Abstract, volle Fassung online verfügbar unter `http://eprint.iacr.org/2002/140.ps`.

[CS99] Jin-yi Cai und Alan L. Selman. Fine separation of average-time complexity classes. *SIAM Journal on Computing*, 28(4):1310–1325, 1999.

[DKMR05] Anupam Datta, Ralf Küsters, John C. Mitchell und Ajith Ramanathan. On the relationships between notions of simulation-based

security. In Joe Kilian (Hrsg.), *Theory of Cryptography, Proceedings of TCC 2005*, Lecture Notes in Computer Science, Seiten 476–494. Springer-Verlag, 2005. Online verfügbar unter `http://www.ti.informatik.uni-kiel.de/~kuesters/publications_html/DattaKuestersMitchellRamanathan-TCC-2005.ps.gz`.

[DN93] C. Dwork und M. Naor. Pricing via processing, or, combatting junk mail. In Ernest F. Brickell (Hrsg.), *Advances in Cryptology, Proceedings of CRYPTO '92*, Band 740 von *Lecture Notes in Computer Science*, Seiten 139–147. Springer-Verlag, 1993. Online verfügbar unter `http://dsns.csie.nctu.edu.tw/research/crypto/HTML/PDF/C92/139.PDF`.

[DNS98] Cynthia Dwork, Moni Naor und Amit Sahai. Concurrent zero-knowledge. In *Thirtieth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1998*, Seiten 409–418. ACM Press, 1998. Volle Fassung online verfügbar unter `http://eprint.iacr.org/1999/023`.

[DNW05] Cynthia Dwork, Moni Naor und Hoeteck Wee. Pebbling and proofs of work. In Victor Shoup (Hrsg.), *Advances in Cryptology, Proceedings of CRYPTO '05*, Band 3621 von *Lecture Notes in Computer Science*, Seiten 37–54. Springer-Verlag, 2005. Online verfügbar unter `http://www.wisdom.weizmann.ac.il/~naor/PAPERS/peb_abs.html`.

[FOPS04] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval und Jacques Stern. RSA-OAEP is secure under the RSA assumption. *Journal of Cryptology*, 17(2):81–104, 2004. Online verfügbar unter `http://www.di.ens.fr/~pointche/Documents/Papers/2004_joc.pdf`.

[FS90] Uriel Feige und Adi Shamir. Zero knowledge proofs of knowledge in two rounds. In Gilles Brassard (Hrsg.), *Advances in Cryptology, Proceedings of CRYPTO '89*, Band 435 von *Lecture Notes in Computer Science*, Seiten 526–544. Springer-Verlag, 1990.

[FS04] Lance Fortnow und Rahul Santhanam. Hierarchy theorems for probabilistic polynomial time. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, Seiten 316–324, 2004. Online verfügbar unter `http://people.cs.uchicago.edu/~fortnow/papers/probhier.ps`.

[GK96a] Oded Goldreich und Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(2):167–189, 1996. Online verfügbar unter `http://www.wisdom.weizmann.ac.il/~oded/PS/zkAK.ps`.

[GK96b] Oded Goldreich und Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, 1996. Online verfügbar unter `http://citeseer.ist.psu.edu/goldreich90composition.html`.

[GL91] Shafi Goldwasser und Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In Alfred Menezes und Scott A. Vanstone (Hrsg.), *Advances in Cryptology, Proceedings of CRYPTO '90*, Band 537 von *Lecture Notes in Computer Science*, Seiten 77–93. Springer-Verlag, 1991.

[GLS88] Martin Grötschel, László Lovász und Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, Band 2 von *Algorithms and Combinatorics*. Springer-Verlag, 1988. Online verfügbar unter `http://www-gdz.sub.uni-goettingen.de/cgi-bin/digbib.cgi?PPN330894919`.

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

[GMW86] Oded Goldreich, Silvio Micali und Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. In *27th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 1986*, Seiten 174–187. IEEE Computer Society, 1986. Online verfügbar unter `http://www.wisdom.weizmann.ac.il/~oded/X/gmw1c.pdf`, volle Fassung ist [GMW91].

[GMW87] Oded Goldreich, Silvio Micali und Avi Wigderson. How to play any mental game—a completeness theorem for protocols with honest majority. In *Nineteenth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1987*, Seiten 218–229. ACM Press, 1987. Erweitertes Abstract.

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

1991. Online verfügbar unter http://www.wisdom.weizmann.ac.il/~oded/X/gmw1j.pdf.

[GO94] Oded Goldreich und Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 1994. Online verfügbar unter http://citeseer.ist.psu.edu/goldreich94definitions.html.

[Gol01] Oded Goldreich. *Foundations of Cryptography – Volume 1 (Basic Tools)*. Cambridge University Press, August 2001. Vorläufige Fassung online verfügbar unter http://www.wisdom.weizmann.ac.il/~oded/frag.html.

[Gol02] Oded Goldreich. Zero-knowledge twenty years after its invention, 2002. Online verfügbar unter http://eprint.iacr.org/2002/186.

[Gol04] Oded Goldreich. *Foundations of Cryptography – Volume 2 (Basic Applications)*. Cambridge University Press, Mai 2004. Vorläufige Fassung online verfügbar unter http://www.wisdom.weizmann.ac.il/~oded/frag.html.

[HMQS04] Dennis Hofheinz, Jörn Müller-Quade und Rainer Steinwandt. On modeling IND-CCA security in cryptographic protocols, 2004. Erweitertes Abstract, erweiterte Fassung online verfügbar unter http://eprint.iacr.org/2003/024.ps.

[HMQU05a] Dennis Hofheinz, Jörn Müller-Quade und Dominique Unruh. Polynomial runtime in simulatability definitions. In *18th IEEE Computer Security Foundations Workshop, Proceedings of CSFW 2005*, Seiten 156–169. IEEE Computer Society, 2005. Online verfügbar unter http://www.infsec.cs.uni-sb.de/~unruh/publications/hofheinz05polynomial.html.

[HMQU05b] Dennis Hofheinz, Jörn Müller-Quade und Dominique Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. In *Proceedings of the 5th Central European Conference on Cryptology, MoraviaCrypt '05*, 2005. Online verfügbar unter http://www.infsec.cs.uni-sb.de/~unruh/publications/hofheinz05universally.html.

[HMQU06a] Dennis Hofheinz, Jörn Müller-Quade und Dominique Unruh. On the (im)possibility of extending coin toss. In Serge Vaudenay (Hrsg.), *Advances in Cryptology, Proceedings of EUROCRYPT '06*,

Band 4004 von *Lecture Notes in Computer Science*, Seiten 504–521. Springer-Verlag, 2006. Volle Fassung online verfügbar unter `http://eprint.iacr.org/2006/177`.

[HMQU06b] Dennis Hofheinz, Jörn Müller-Quade und Dominique Unruh. A simple model of polynomial time UC (abstract). Workshop on Models for Cryptographic Protocols (MCP 2006), Århus, Juli 2006. Online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/hofheinz06simple.html`.

[HS65] Juris Hartmanis und Richard Edwin Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117(5):285–306, Mai 1965.

[HU05a] Dennis Hofheinz und Dominique Unruh. Comparing two notions of simulatability. In Joe Kilian (Hrsg.), *Theory of Cryptography, Proceedings of TCC 2005*, Lecture Notes in Computer Science, Seiten 86–103. Springer-Verlag, 2005. Online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/hofheinz05comparing.html`.

[HU05b] Dennis Hofheinz und Dominique Unruh. On the notion of statistical security in simulatability definitions. In Jianying Zhou, Javier Lopez, Robert H. Deng und Feng Bao (Hrsg.), *Information Security, Proceedings of ISC '05*, Lecture Notes in Computer Science, Seiten 118–133. Springer-Verlag, 2005. Online verfügbar unter `http://eprint.iacr.org/2005/032`.

[HU06a] Dennis Hofheinz und Dominique Unruh. An attack on a group-based cryptographic scheme, 2006. Erscheint im J. of Contemporary Mathematics, AMS.

[HU06b] Dennis Hofheinz und Dominique Unruh. Simulatable security and polynomially bounded concurrent composition. In *IEEE Symposium on Security and Privacy, Proceedings of SSP '06*, Seiten 169–182. IEEE Computer Society, 2006. Volle Fassung online verfügbar unter `http://eprint.iacr.org/2006/130.ps`.

[KM92] Daphne Koller und Nimrod Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior*, 4(4):528–552, Oktober 1992. Online verfügbar unter `http://theory.stanford.edu/~megiddo/pdf/recall.pdf` (ohne Abbildungen).

[KPR98] Joe Kilian, Erez Petrank und Charles Rackoff. Lower bounds for zero knowledge on the internet. In *39th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 1998*, Seiten 484–492. IEEE Computer Society, 1998. Volle Fassung online verfügbar unter `http://arxiv.org/abs/cs.CR/0107003`.

[Kuh56] Harold William Kuhn. Extensive games and the problem of information. In Kenneth J. Arrow und Harold William Kuhn (Hrsg.), *Contributions to the theory of games*, Band 2, Seiten 193–216. Princeton University Press, 2 Auflage, 1956.

[Küs06] Ralf Küsters. Simulation-based security with inexhaustible interactive turing machines. IACR eprint 2006/151, 2006. Online verfügbar unter `http://eprint.iacr.org/2006/151`.

[Lin03] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *44th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2003*, Seiten 394–403. IEEE Computer Society, 2003. Online verfügbar unter `http://eprint.iacr.org/2003/141`.

[May93] Timothy C. May. Timed-release crypto. Cypherpunks Mailing List (cypherpunks@toad.com), Februar 1993. Online verfügbar unter `http://cypherpunks.venona.com/date/1993/02/msg00129.html`.

[MMS03] P. Mateus, J. Mitchell und A. Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time process calculus. In Roberto Amadio und Denis Lugiez (Hrsg.), *Concurrency Theory, Proceedings of CONCUR '03*, Band 2761 von *Lecture Notes in Computer Science*, Seiten 327–349. Springer-Verlag, 2003. Online verfügbar unter `http://wslc.math.ist.utl.pt/ftp/pub/MateusP/03-MMS-seccomp.pdf`.

[MQ02] Jörn Müller-Quade. Quantum pseudosignatures. *Journal of Modern Optics*, 49(8):1269–1276, Juli 2002.

[MQRU05] Jörn Müller-Quade, Stefan Röhrich und Dominique Unruh. Oblivious transfer is incomplete for deniable protocols. Workshop on The Past, Present and Future of Oblivious Transfer, Haifa, Mai 2005. Online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/muellerquade05oblivious.html`.

[MQS03] Jörn Müller-Quade und Rainer Steinwandt. On the problem of authentication in a quantum protocol to detect traffic analysis. *Quantum Information and Computation*, 3(1):48–54, 2003.

[MQU06] Jörn Müller-Quade und Dominique Unruh. Composable deniability (abstract). Workshop on Models for Cryptographic Protocols (MCP 2006), århus, Juli 2006. Online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/muellerquade06composable.html`.

[MQU07] Jörn Müller-Quade und Dominique Unruh. Long-term security and universal composability, 2007. Erscheint auf der TCC 2007, volle Fassung online verfügbar unter `http://eprint.iacr.org/2006/422`.

[MR91] Silvio Micali und Phillip Rogaway. Secure computation. Unveröffentlichtes Manuskript, erweitertes Abstract ist [MR92], 1991.

[MR92] Silvio Micali und Phillip Rogaway. Secure computation (abstract). In Joan Feigenbaum (Hrsg.), *Advances in Cryptology, Proceedings of CRYPTO '91*, Band 576 von *Lecture Notes in Computer Science*, Seiten 392–404. Springer-Verlag, 1992.

[Nas50] John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of the USA*, 36(1):48–49, 1950. Online verfügbar unter `http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1063129`.

[PKC02] RSA Laboratories. *PKCS #1: RSA Cryptography Standard, Version 2.1*, 2002. Online verfügbar unter `ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf`.

[PS04] Manoj Prabhakaran und Amit Sahai. New notions of security: achieving universal composability without trusted setup. In László Babai (Hrsg.), *36th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2004*, Seiten 242–251. ACM Press, 2004. Volle Fassung online verfügbar unter `http://eprint.iacr.org/2004/139`.

[PW01] Birgit Pfitzmann und Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy, Proceedings of SSP '01*, Seiten 184–200. IEEE Computer Society, 2001. Volle Fassung online verfügbar unter `http://eprint.iacr.org/2000/066.ps`.

[Ras89] Eric Rasmusen. *Games and Information*. Basil Blackwell, 1989.

[RK99] Ransom Richardson und Joe Kilian. On the concurrent composition of zero-knowledge proofs. In Jacques Stern (Hrsg.), *Advances in Cryptology, Proceedings of EUROCRYPT '99*, Band 1592 von *Lecture Notes in Computer Science*, Seiten 415–431. Springer-Verlag, 1999.

[RSW96] Ronald L. Rivest, Adi Shamir und David A. Wagner. Timelock puzzles and timed-release crypto. Technischer Bericht MIT/LCS/TR-684, Massachusetts Institute of Technology, Februar 1996. Online verfügbar unter `http://theory.lcs.mit.edu/~rivest/RivestShamirWagner-timelock.ps`.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 1994*, Seiten 124–134. IEEE Computer Society, 1994.

[SHS02] Federal Information Processing Standards Publications. *FIPS PUB 180-2: Secure Hash Standard (SHS)*, August 2002. Online verfügbar unter `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf`.

[Unr02] Dominique Unruh. Formal security in quantum cryptology. Studienarbeit, Institut für Algorithmen und Kognitive Systeme, Universität Karlsruhe, Dezember 2002. Online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/unruh02formal.html`.

[Unr03] Dominique Unruh. Zufallsextraktoren für Quellen variierender Qualität. Diplomarbeit, Institut für Algorithmen und Kognitive Systeme, Universität Karlsruhe, Juli 2003. Online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/unruh03zufallsextraktoren.html`.

[Unr04a] Dominique Unruh. Classical control in quantum programs. 5th European QIPC Workshop, Roma, September 2004. Poster, online verfügbar unter `http://www.infsec.cs.uni-sb.de/~unruh/publications/unruh04classical.html`.

[Unr04b] Dominique Unruh. Relating formal security for classical and quantum protocols. Isaac Newton Institute for Mathematical Sciences, eingeladener Vortrag, September 2004. Online verfügbar

unter http://www.infsec.cs.uni-sb.de/~unruh/publications/unruh04relating.html.

[Unr04c] Dominique Unruh. Simulatable security for quantum protocols, September 2004. Online verfügbar unter http://arxiv.org/ps/quant-ph/0409125.

[Unr05] Dominique Unruh. Quantum programs with classical output streams. *Electronic Notes in Theoretical Computer Science*, 2005. 3rd International Workshop on Quantum Programming Languages, online verfügbar unter http://www.infsec.cs.uni-sb.de/~unruh/publications/unruh05quantum.html.

[Unr06a] Dominique Unruh. Long-term universal composability. RISC Seminar, CWI, Amsterdam, eingeladener Vortrag, März 2006. Volle Fassung ist [MQU07].

[Unr06b] Dominique Unruh. Quantum programming languages. Teil einer Studie im Rahmen des BMBF-Projekts MARQUIS. *Informatik – Forschung und Entwicklung*, 21(1):55–63, 2006.

[Unr06c] Dominique Unruh. Relations among statistical security notions or why exponential adversaries are unlimited, 2006. Online verfügbar unter http://eprint.iacr.org/2005/406.

[vEB90] Peter van Emde Boas. Machine models and simulations. In Jan van Leeuwen (Hrsg.), *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, Seiten 1–66. Elsevier, 1990.

[vNM44] John von Neumann und Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton Univ. Press, 1944.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computation. In *23th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 1982*, Seiten 160–164. IEEE Computer Society, 1982. Online verfügbar unter http://www.cs.wisc.edu/areas/sec/yao1982-ocr.pdf.

[Yao95] Andrew Chi-Chih Yao. Security of quantum protocols against coherent measurements. In *Twenty-Seventh Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1995*, Seiten 67–75. ACM Press, 1995.

J. Index