Universal Composability en Indifferentiability

Rogaar, P.

P. Rogaar

Universal Composability en Indifferentiability

Bachelorscriptie, 24 augustus 2009

Scriptiebegeleider: dr. D. Hofheinz

Mathematisch Instituut, Universiteit Leiden

Inhoudsopgave

Hoofdstuk 1

Voorwoord

In de wereld van de cryptologie worden oplossingen bestudeerd voor uiteenlopende problemen die zich alle kenmerken door twee zaken. Ten eerste is daar het doel dat bereikt moet worden - denk aan een bancaire transactie, of een geheime berekening. Ten tweede is er een abstracte aanvallende instantie, die de adversary wordt genoemd. Deze aanvaller kan uit meerdere systemen bestaan, maar voor de zekerheid wordt aangenomen dat die allemaal samenwerken.

Het doel dat bereikt moet worden met de gezochte oplossing, kan zo eenvoudig zijn als twee instanties die samen een random bit willen genereren, of vele malen complexer - online veilingen en verkiezingen zijn maar twee voorbeelden. Vaak wordt voor het bereiken van de eenvoudige doelstellingen effectief gebruik gemaakt van bepaalde (mogelijk onbewezen) getaltheoretische eigenschappen. Voor de complexere doelstellingen is dat niet altijd doenlijk, en zodoende wordt gepoogd de doelstelling te bereiken door de kleinere cryptografische primitieven te combineren tot een groter geheel met de gewenste eigenschappen. Hierbij dient zich echter een probleem aan. Hoe weten we dat de eigenschappen die een bepaalde methode op zich heeft, ook in een groter geheel geplaatst nog gelden? Met andere woorden, we hebben een kader nodig waarin we kunnen beschrijven welke eigenschappen van een methode bewaard blijven, wat de omgeving ook voor invloed vormt.

Neem bijvoorbeeld het volgende scenario: RSA-versleuteling kan gebruikt worden om berichten te versleutelen met een publieke sleutel, zodat de ontvanger zijn geheime sleutel kan gebruiken om deze te ontsleutelen. Is deze encryptie geschikt om gebruikt te worden in een protocol voor een online veiling? Misschien hebben berichten uit dat protocol een bepaalde voorspelbare structuur (denk aan 'voorwerp, bod') waaruit de sleutel stiekem verkregen kan worden. Mogelijk kan een van de verkopers zijn rol misbruiken door als 'man-in-the-middle' op te treden. Het is zelfs denkbaar dat er aanvallen mogelijk zijn op de RSA-methode die we nu niet kunnen bedenken. Wat we nodig hebben, is een beschrijving van de eigenschappen die RSA-versleuteling heeft, *ongeacht de omgeving.*

Universal Composability is een raamwerk dat ontwikkeld is door Ran Canetti om deze beschrijving te kunnen leveren. Uit deze methode van beschrijven volgen ook enkele stellingen over de veiligheid van systemen die uit dergelijke componenten opgebouwd zijn.

In deze scriptie zal in worden gegaan op de verschillende aspecten van dergelijke beschrijvingen in componenten. Ook zal de compositiestelling besproken worden, waarmee eigenschappen van componenten kunnen worden gebruikt als basis voor bewijzen over de geldigheid van grotere protocollen. Verder zal, om een en ander breder te beschouwen, ook het raamwerk van indifferentiability worden besproken zoals dat in [Maurer et al., 2004] geïntroduceerd is. In dit raamwerk kan de compositiestelling ook geformuleerd worden, maar het overbrengen van het bewijs naar het andere raamwerk heeft wat voeten in de aarde. Enkele problemen die zich voordoen, worden in sectie 4.3 besproken.

Hoofdstuk 2

Definities

Omdat er veel begrippen in dit raamwerk besproken zullen worden die mogelijk nog niet bekend zijn bij de lezer, zal er voorafgaand aan een formelere bespreking, geschetst worden wat de ideeën achter de besproken concepten zijn. In sectie 2.6 zullen deze ideeën met formelere structuren in verband gebracht worden.

2.1 Protocol

In het veld van computercommunicatie is een protocol een bepaalde functionaliteit die gerealiseerd kan worden. De uitwerking van deze functionaliteit kan op verschillende niveaus beschreven zijn.

Op heel abstract niveau valt hierbij te denken aan een zwarte doos waarop een aantal gebruikers aangesloten is; Zij geven hun invoer door via een niet af te luisteren kanaal aan de doos, en krijgen even later het resultaat van het protocol terug. We noemen deze beschrijving, die sterk gebaseerd is op de wensen van de gebruikers, ook wel een *ideaal protocol*. Als deze abstractie bij een bezoek aan de groenteboer wordt ingezet, zal dat ongeveer als volgt verlopen.

Een klant loopt binnen bij de groenteboer en wil een komkommer kopen. In de winkel staat een grote zwarte doos, aan wie de klant zijn portemonnee en boodschappenlijstje geeft. De groenteboer geeft al zijn groenten aan de doos, waarna deze een komkommer, het boodschappenlijstje met het woord 'komkommer' doorgestreept en de portemonnee met iets minder geld teruggeeft aan de klant, en wat geld en de rest van de groenten aan de groenteboer.

Deze notie kan gebruikt worden bij allerlei doelstellingen. Bij een veiling geven alle partijen hun biedingen door, waarna de doos bekendmaakt aan iedereen wie er gewonnen heeft. Bij een identificatieproces geeft degene die zich legitimeert zijn geheime sleutel door en de controlerende partij de identiteit die bevestigd dient te worden, waarna de

doos doorgeeft aan de controleur of de gelegitimeerde persoon inderdaad de bedoelde identiteit heeft. Alle partijen vertrouwen de zwarte doos om de gestelde functionaliteit met het juiste resultaat uit te voeren. In die zin is het hier weergegeven protocol *ideaal*, daar het slechts een ideale versie schetst van de communicatie tussen de protocollen. Door gebruik te maken van de betrouwbare zwarte doos, kunnen problemen bij de transactie ook vermeden worden. Helaas leven we niet in een wereld met betrouwbare zwarte dozen, en zullen we een ander protocol voor onze transactie, een 'reeële' variant op het ideale protocol, moeten maken, die uiteindelijk toch hetzelfde bereikt. Het voorbeeld van een bezoek aan de groenteboer staat in een dergelijke formulering ook dichter bij de gebruikelijke ervaring in een winkel.

Een klant loopt binnen bij de groenteboer en wil een komkommer kopen. Hij stapt naar de balie toe en zegt gedag. De groenteboer vraagt wat hij wil kopen, de klant zegt: "Een komkommer". De groenteboer noemt een prijs, en de klant knikt. De groenteboer pakt een komkommer en legt deze op de toonbank. De klant pakt het bedrag uit zijn portemonnee en legt dit ook op de toonbank. Beide partijen pakken hun nieuwe bezit van de toonbank en bergen het weg.

In deze formulering komen meerdere stappen naar voren: Eerst is er een ontmoetingsfase, waarbij beide partijen hun wensen uiten. De afhandelingsfase volgt, waarna in een 'gelijk oversteken'-proces beide partijen eigenaar worden van de gewenste zaken.

Dit verschil tussen reële en ideale protocollen staat centraal in onze verdere bespreking. Steeds zoeken we een reëel protocol om de eigenschappen van een ideaal protocol zo goed mogelijk na te doen. In deze scriptie zullen we uitgaan van een abstractere formulering, omdat deze door het algemene karakter een grotere uitdrukkingskracht in onze onderwerpen representeert. In deze context zullen we protocollen vaak met π of ρ noteren, of, als we spreken over een ideaal protocol, met φ.

2.2 Adversary

De twee beschrijvingen die in de vorige paragraaf gegeven zijn, zijn licht verwant. Uiteindelijk hebben ze het zelfde resultaat, *als* er geen rare dingen gebeuren. Immers, in een groentewinkel zijn meer klanten die zich met de transactie kunnen bemoeien, en ook de andere medewerkers van de groenteboer kunnen de transactie beïnvloeden. In het eerste geval maakt dit niet uit. Immers, de zwarte doos regelt de hele transactie voor de klant en de winkelier, waardoor anderen niet de kans krijgen het proces te verstoren. In het tweede geval zou dit bijvoorbeeld kunnen leiden tot diefstal van het geld en de komkommer terwijl ze op de toonbank liggen, of het stiekem vervangen van de komkommer door een mandarijn.

Net zo beschouwen we bij protocollen situaties waarbij een kwaadwillende derde partij de manier van het afwerken van het protocol wil beïnvloeden. Deze partij kan hierdoor doeleinden bereiken die niet wenselijk zijn voor de legitiem deelnemende partijen. In

concrete protocollen voor computercommunicatie kan dit bijvoorbeeld betekenen dat een buitenstaander in staat is om de geboden bedragen in een veiling te achterhalen en zo eenvoudig de veiling met een zo klein mogelijk hoger bod kan winnen, of zelfs de biedingen van anderen kan blokkeren. Soms zouden verschillende buitenstaanders met verschillende mogelijkheden kunnen samenwerken om één doel te bereiken; Daarom modelleren we deze tegenstanders als één partij die de beschikking heeft over al hun mogelijkheden. Deze partij, die we de *adversary* noemen, heeft hierdoor alle mogelijkheden die iedere groep samenwerkende buitenstaanders kan hebben.

De adversary, die vaak met $\mathcal{A}$ of $\mathcal{S}$ wordt aangeduide, kan niet alleen communicatie tussen verschillende partijen afluisteren en manipuleren, maar ook partijen die deelnemen aan het protocol aanwijzen om ze te corrumperen. Deze notie modelleert het idee dat er een aantal partijen van binnenuit samenwerken met de adversary. In de praktijk is dit realistisch; Bij het uitvoeren van het protocol kunnen bijvoorbeeld medewerkers van het bedrijf betrokken zijn die spioneren voor de concurrent, of de legitieme toegangsweg tot het protocol is openbaar toegankelijk zoals bijvoorbeeld bij het inloggen voor een website.

2.3 Securityparameter

We hebben het in de voorgaande secties gehad over partijen die protocollen uitvoeren en de adversary die daar op ongewenste wijze tegenin gaat. In sommige gevallen kunnen we een protocol maken dat een doelstelling zó realiseert, dat wat de adversary daar ook tegen kan doen, het nooit zal lukken om het protocol te compromitteren. Bij veel doelstellingen kunnen we echter bewijzen dat ze niet te realiseren zijn met een protocol dat zo veilig is. Een voorbeeld hiervan is publieke-sleutel encryptie. Hierbij wordt een paar sleutels gegenereerd, waarbij de ene de publieke sleutel en de andere de geheime sleutel wordt genoemd. Deze publieke sleutel wordt vervolgens aan iedereen bekendgemaakt. Deze sleutels zijn zo geconstrueerd, dat berichten die met de publieke sleutel versleuteld zijn, met de geheime sleutel te ontcijferen zijn. Echter, door alle mogelijke geheime sleutels af te gaan en de bijbehorende publieke sleutel te berekenen kunnen we eenvoudig een geheime sleutel vinden voor een bepaalde publieke sleutel. Aangezien de geheime sleutel dan bekend is, is het encryptieschema niet veilig meer.

Dit voorbeeld geeft aan dat we problemen willen kenmerken aan de hand van hun moeilijkheidsgraad. Immers, het genereren van de sleutels moet *gemakkelijk* zijn, net als het versleutelen en ontcijferen met gegeven sleutels, maar het vinden van de geheime sleutel als de publieke sleutel gegeven is, moet *moeilijk* zijn, net als het ontcijferen van een versleuteld bericht als de geheime sleutel niet beschikbaar is. Maar wat is moeilijk? Als het meer dan een zeker aantal, zeg een miljard, stappen op een computer vergt? Een paar jaar later is een moeilijk probleem dan misschien opeens helemaal niet zo moeilijk meer op te lossen, omdat de beschikbare computerkracht is toegenomen. Mogelijk komen er ook nieuwe algoritmen beschikbaar om zo'n probleem sneller op te kunnen lossen - dit

kan het oplossen van een voorheen als lastig ervaren probleem totaal triviaal maken.

Uit het voorgaande blijkt dat de moeilijkheidsgraad van een opgave uitgedrukt mag worden in hoe groot de invoer is – een moeilijke opgave mag best snel op te lossen zijn voor kleine invoerwaarden, als de benodigde tijd maar heel snel groter wordt als de invoer groter wordt. Blijft de berekening relatief eenvoudig, dan noemen we het probleem gemakkelijk.

Willen we dit formeler definiëren, dan hebben we het concept van een Turingmachine nodig. Een Turingmachine is een abstract logisch concept dat programma's kan draaien. Het heeft veel weg van een computer en voor al onze doeleinden kunnen we er van uitgaan dat waar we het over een Turingmachine hebben die een programma draait, we het over een computer hebben die deze berekening uitvoert. We gaan er hier verder van uit dat onze Turingmachines probabilistisch zijn, dus dat onze computer ook algoritmen uit kan voeren die een element van willekeurigheid vergen. Op de verschillen tussen deterministische en probabilistische Turingmachines gaan we hier niet verder in.

Stel dat er een Turingmachine bestaat die een bepaald soort probleem voor ons oplost. Beschouwen we nu het aantal rekenstappen dat nodig is voor het vinden van de oplossing, dan hangt dat aantal mogelijk af van de moeilijkheidsgraad van de gekozen instantie van het probleem – hierbij valt bijvoorbeeld te denken aan de lengte van een te factoriseren getal of het aantal decimalen dat we van π willen vinden. Als dit aantal stappen naar boven begrensd wordt door een functie die polynomiaal is in de securityparameter voor alle instanties, dan noemen we de Turingmachine PPT, hetwelk staat voor Probabilistisch Polynomiale Tijd. Dit correspondeert met onze eerdergenoemde notie van gemakkelijke problemen, oftewel problemen met een gemakkelijke oplossing. Is er geen PPT Turingmachine die ons probleem oplost, dan is het probleem moeilijk (i.e. er is dan geen gemakkelijke oplossing).

Als een cryptosysteem gebruik maakt van computationele aannames, dan wordt aangenoment dat er voor een bepaald probleem geen PPT Turingmachines bestaan die dat probleem oplossen. Dit betekent dat als we een moeilijke instantie van het probleem kiezen, zoals een groot te factoriseren getal, dat het oplossen dan heel snel moeilijker wordt. Omdat de veiligheid van ons cryptosysteem afhangt van de moeilijkheidsgraad van het corresponderende probleem, noemen we dit getal de securityparameter van het systeem.

2.4 Omgeving

Bij het beschouwen van samenstellingen van protocollen, waarbij een klein protocol optreedt als subprotocol van een ander, willen we dat eigenschappen van het grote protocol geen invloed hebben op de eigenschappen van het kleinere protocol. Met andere woorden, *wat er ook gebeurt* met het kleine protocol, de eigenschappen blijven overeind.

Dat een protocol op verscheidene manieren gebruikt kan worden door de partijen en de adversary wordt gemodelleerd met behulp van een *omgeving*. Deze omgeving, die we vaak met $\mathcal{Z}$ noteren, is in feite ook een PPT Turingmachine die het uitvoeren van het protocol en de verschillende partijen op zich neemt. Als we de omgeving uitvoeren, zal deze eerst de adversary aanroepen en vervolgens de verschillende partijen het protocol uit laten voeren. De uitvoer van de omgeving na het uitvoeren van alle partijen noemen we het resultaat van het uitvoeren van het protocol. Per protocol kunnen er vele omgevingen gedefinieerd worden. Er zijn immers veel verschillende manieren om met een protocol om te gaan.

Het idee van een omgeving heeft geen concrete parallel – er is in ons voorbeeld van het bezoek aan de groenteboer geen entiteit die de verschillende partijen aanstuurt om de aankoop van een komkommer te bewerkstelligen. In stellingen en beschrijvingen vervult de omgeving altijd dezelfde rol: Het is een omschrijving van de manier waarop het protocol uitgevoerd wordt door de deelnemers en de adversary.

2.5 Emulatie

Om veiligheidseigenschappen te kunnen definiëren op een algemene manier, kunnen we geen gebruik maken van conventionele cryptografische termen – immers, ook als er nieuwe aanvalsmethoden ontstaan, moet ons protocol daartegen bestand zijn. Als we denken aan aanvallen op een reële implementatie van een protocol, dan bedoelen we aanvallen die de functionaliteit beperken – bijvoorbeeld omdat een sleutel niet meer geheim is. De functionaliteit die de correcte werking van een protocol beschrijft, definieert een ideaal protocol dat correspondeert met de reeële implementatie – net zoals het zwartedoos-voorbeeld van de groenteboer een ideaal protocol is bij de reële implementatie van de uitwisseling bij de kassa.

Stel dat de omgeving de partijen bestuurt bij het uitvoeren van het protocol en dat alles goed loopt – i.e., de adverary doet niets en communiceert niet met de omgeving. Op geen enkele manier is nu voor de omgeving te onderscheiden of hij te maken heeft met het ideale protocol of een reële implementatie daarvan – immers, de boodschappen zijn gelijk, dus als we de omgeving vragen te raden of het om een ideaal of een reëel protocol gaat, dan is de kans dat deze goed raadt, niet hoger dan $\frac{1}{2}$. We eisen hier dat de partijen die deelnemen aan het protocol, zich in principe (dus zonder externe beïnvloeding door de adversary) netjes aan het protocol houden. Dit is geen beperking, omdat we iedere partij die afwijkt van het protocol, modelleren alsof deze daartoe wordt aangezet door de adversary. Door het samenvoegen van de kennis en de besturing van alle afwijkende elementen, ontstaat de sterkst mogelijke situatie voor de aanvallers, die daardoor hun acties naar believen kunnen afstemmen en synchroniseren.

De zaken staan er anders voor als de adversary zich bezig gaat houden met de uitvoering van het protocol. Immers, hij is in staat om communicatie af te luisteren en te veranderen

en partijen te corrumperen. Dit kan een flinke invloed hebben op de correcte uitvoering van het protocol, en omdat de adversary informatie door kan spelen aan de omgeving, kan ook de omgeving merken dat de uitvoering anders verloopt. Mocht een dergelijke aanval niet mogelijk zijn op het ideale protocol, dan betekent dat dat een dergelijke adversary de omgeving in staat stelt onderscheid te maken tussen het draaien van het ideale en het reële protocol. Merk op dat dit een noodzakelijke voorwaarde is om gebruik te kunnen maken van zwakheden in de reële implementatie van een protocol – immers, als er geen efficiënte manier is om onderscheid te maken tussen het reële en het ideale protocol, dan is er zeker geen manier om het reële protocol te misbruiken op een manier die niet correspondeert met een aanval op het ideale protocol.

2.6 Formele definities

2.6.1 Interactieve Turingmachine

Een Interactieve Turingmachine (ITM) is een formalisatie van het concept van een computerprogramma dat op een computer draait, waarbij het ook met andere programma's kan communiceren. Een Turingmachine bestaat uit een stel (papier)stroken[1] waarop gegevens bijgehouden kunnen worden en instructies om, eventueel na het invoeren van gegevens van de stroken en het uitvoeren van logische operaties op de gegevens, deze weg te schrijven naar deze of andere stroken. Een ITM heeft bovendien de mogelijkheid om sommige (extern beschrijfbare) stroken van *andere* ITM's te beschrijven en op sommige stroken gegevens van andere ITM's te ontvangen. Dit geeft een ITM de mogelijkheid om te communiceren met andere ITM's. Een ITM heeft de beschikking over de volgende stroken:

- Een extern beschrijfbare strook genaamd identiteit.
- Een extern beschrijfbare strook genaamd securityparameter.
- Een extern beschrijfbare strook genaamd invoer.
- Een extern beschrijfbare strook genaamd inkomendecommunicatie.
- Een extern beschrijfbare strook genaamd subroutine − uitvoer.
- Een strook genaamd uitvoer.
- Een strook genaamd random.
- Een één bit lange strook genaamd activering.
- Een strook genaamd werkstrook.

[1] Het woordgebruik rond Turingmachines, in het bijzonder de stroken waarmee gewerkt wordt, doet mogelijk vermoeden dat Turingmachines werkelijk gebouwd en gebruikt worden. Dit is niet het geval: het zijn zuiver logische hulpmiddelen

Het uitvoeren van een ITM is een proces dat van buitenaf in gang gezet wordt. Tijdens het uitvoeren gebeurt het volgende:
Het initiërende proces beschrijft enkele stroken van de ITM. De strook identiteit wordt gevuld met de gewenste identiteit van deze instantie van de ITM. De strook getiteld securityparameter wordt beschreven met een bitstring die bestaat uit een aantal enen[2]. Het initiërende proces schrijft verder een voldoende lange random string op random en tenslotte de invoer voor de code van het proces op invoer. De code van de Turingmachine gaat nu aan de slag om aan de hand van de invoer op invoer, inkomendecommunicatie (afkomstig van het netwerk tijdens het uitvoeren) en subroutine − uitvoer (afkomstig van subroutines van de ITM) berekeningen te verrichten. Gedurende deze berekeningen kan de ITM schrijven op de stroken werkstrook, bedoeld als intern geheugen, en de inkomendecommunicatie van andere ITM's. Tevens kan hij andere ITM's als subroutine aanspreken, waarbij hij nu zelf het initiërende proces is. Aan het einde van de berekening geeft de ITM een bepaalde uitvoer op zijn strook uitvoer. Deze uitvoer wordt teruggegeven aan de instantie die de ITM oorspronkelijk aangeroepen heeft. Onze ITM's zullen begrensd zijn in het aantal rekenstappen dat ze kunnen maken door de securityparameter. Het aantal stappen dat de ITM maakt, moet een polynomiale functie zijn in de securityparameter k. In dit geval spreken we van een Polynomiale tijd Probabilistische Turingmachine (PPT). We noemen de ITM gecombineerd met de gegevens die door het initiërende proces op de stroken geschreven worden, tezamen wel een *instantie* van de ITM, afgekort een ITI. In sectie 2.6.5 wordt verder ingegaan op het uitvoeren van een protocol als een systeem van ITM's. Voor een diepgaandere behandeling van het uitvoeren van een ITM, zie sectie 3.2 en 3.3 van [Canetti, 2005].

2.6.2 Adversary

Ook de adversary wordt geïnterpreteerd als een ITM. Als een protocol uitgevoerd wordt, is de adversary, die we met $\mathcal{A}$ noteren, een ITM die door de omgeving aangeroepen wordt, en deze heeft op zijn beurt meerdere mogelijkheden. Ten eerste kan $\mathcal{A}$ uitvoer aan het initiërende proces geven door deze te schrijven naar zijn uitvoer. Ten tweede kan $\mathcal{A}$ berichten afleveren bij de verschillende andere ITM's op hun inkomendecommunicatie. Ten derde kan $\mathcal{A}$ ook partijen corrumperen door hun een speciaal bericht te sturen, maar dit laatste is alleen mogelijk indien $\mathcal{A}$ daarom gevraagd wordt door zijn initiërende partij.

2.6.3 Omgeving

In een systeem van ITM's is er sprake van een initiële ITM, die als eerste gestart wordt en vervolgens de rest van de ITM's aanroept. In ons kader noemen we deze ITM de

[2]Omdat we de complexiteit van ITM's bepalen aan de hand van de lengte van de input en niet de waarde ervan, wordt de securityparameter doorgegeven als een bitstring van een bepaalde lengte, niet als een bepaald getal.

omgeving $\mathcal{Z}$, vanwege de interpretatie die mogelijk is als 'het grotere protocol' waarbinnen dit systeem uitgevoerd wordt. Verder definieert men een controlefunctie die aangeeft of bepaalde schrijfoperaties toegestaan zijn. Zo mag een ITM niet zomaar op inkomendecommunicatie van een andere ITM schrijven, aangezien alle communicatie door $\mathcal{A}$ gecontroleerd wordt. Deze controlefunctie staat dus communicatie vanaf alle ITM's naar inkomendecommunicatie van $\mathcal{A}$ toe. $\mathcal{A}$ op zijn beurt mag, zoals in 2.6.2 besproken, zelf berichten bedenken en afleveren bij de verschillende ITM's. In het bijzonder kan $\mathcal{A}$ dus kiezen een bericht dat hij binnenkrijgt van een ITM door te sturen (na eventuele aanpassingen) aan andere ITM's. Alle ITM's mogen wel communiceren met $\mathcal{Z}$.

2.6.4 Computationeel ononderscheidbaar

In de complexiteitsleer wordt vaak uitgedrukt hoe groot een functie is, uitgedrukt in een zekere parameter k (in het kader van cryptografie en protocollen: de securityparameter).

Definitie 2.1. *We noemen een functie $f : \mathbb{N} \to \mathbb{R}$ verwaarloosbaar in k (negligible, notatie $f = \mathsf{negl}(k)$) indien $\forall c \in \mathbb{N}\ \exists N_0 \in \mathbb{N}\ \forall k > N_0$ geldt dat*

$$|f(k)| < \frac{1}{k^c}.$$

Met andere woorden, als men f met een reciproke van een polynomiale functie vergelijkt, zal f op de lange duur dichter bij nul komen.

Definitie 2.2. *Zij $X = (X_n)_{n\in\mathbb{N}}$ en $Y = (Y_n)_{n\in\mathbb{N}}$ twee families stochasten. Als we de uitvoer y van een Turingmachine $\mathcal{A}$ op invoer x aangeven met $\mathcal{A}(x) = y$, dan noemen we deze families stochasten* computationeel ononderscheidbaar *indien geldt dat voor alle PPT $\mathcal{A}$*

$$\left| \Pr_{x\leftarrow X_n}[\mathcal{A}(x) = 1] - \Pr_{y\leftarrow Y_n}[\mathcal{A}(y) = 1] \right| = \mathsf{negl}(n).$$

In dat geval noteren we $X \approx_C Y$.

Met andere woorden, er is geen efficiënte methode om een significante voorsprong te hebben bij het raden uit welk van de twee families een bepaalde gerealiseerde waarde komt. De relatie $\approx_C$ is transitief.

Lemma 2.1 (Transitiviteit van $\approx_C$)**.** *Zij $X = (X_n)_{n\in\mathbb{N}}$, $Y = (Y_n)_{n\in\mathbb{N}}$ en $Z = (Z_n)_{n\in\mathbb{N}}$ drie families stochasten, zodanig dat $X \approx_C Y$ en $Y \approx_C Z$. Dan geldt $X \approx_C Z$.*

Bewijs. Zij $X = (X_n)_{n\in\mathbb{N}}$, $Y = (Y_n)_{n\in\mathbb{N}}$ en $Z = (Z_n)_{n\in\mathbb{N}}$ drie families stochasten, willekeurig gegeven. Stel dat $X \approx_C Y$ en $Y \approx_C Z$, dan geldt per definitie van $\approx_C$ dat voor alle PPT $\mathcal{A}$:

$$\left| \Pr_{x\leftarrow X_n}[\mathcal{A}(x) = 1] - \Pr_{y\leftarrow Y_n}[\mathcal{A}(y) = 1] \right| = \mathsf{negl}(n)$$

en

$$\left| \Pr_{y \leftarrow Y_n}[\mathcal{A}(y) = 1] - \Pr_{z \leftarrow Z_n}[\mathcal{A}(z) = 1] \right| = \mathsf{negl}(n).$$

Nu geldt, via de driehoeksongelijkheid, dat voor alle PPT $\mathcal{A}$:

$$\left| \Pr_{x \leftarrow X_n}[\mathcal{A}(x) = 1] - \Pr_{z \leftarrow Z_n}[\mathcal{A}(z) = 1] \right| \leq$$

$$\left| \Pr_{x \leftarrow X_n}[\mathcal{A}(x) = 1] - \Pr_{y \leftarrow Y_n}[\mathcal{A}(y) = 1] \right| +$$

$$\left| \Pr_{y \leftarrow Y_n}[\mathcal{A}(y) = 1] - \Pr_{z \leftarrow Z_n}[\mathcal{A}(z) = 1] \right| = \mathsf{negl}(n).$$

Derhalve is dan ook $X \approx_C Z$. □

Het concept van computationele ononderscheidbaarheid heeft een concrete interpretatie bij de emulatie van protocollen zoals eerder beschreven. Hierbij moet de omgeving onderscheid maken: Is, met alle uitvoer van de verschillende ITM's, er efficiënt onderscheid te maken tussen het resultaat van het uitvoeren van het ene protocol of het andere? In dat licht moet de definitie zoals gegeven in sectie 2.5 gezien worden: Of we nu het ene of het andere protocol nemen, bestaat er een PPT $\mathcal{Z}$ die onderscheid kan maken, dus is er een denkbaar kader waarbinnen het uitmaakt welke je gebruikt?

2.6.5 Protocollen uitvoeren

Het uitvoeren van een protocol komt neer op het verenigen van alle bovengenoemde concepten. Van een verzameling PPT's wordt als eerste de omgeving $\mathcal{Z}$ geactiveerd, met een bepaalde securityparameter. In de uitvoering van $\mathcal{Z}$ is de adversary $\mathcal{A}$ de eerstvolgende PPT die geactiveerd wordt. Deze PPT's kunnen nu beide nieuwe PPT's aanroepen, zoals instanties van het protocol dat uitgevoerd wordt. De controlefunctie van het systeem PPT's regelt de communicatiemogelijkheden. Als een instantie van het protocol een andere instantie activeert, dan ontvangt deze de uitvoer van deze instantie direct. Echter, als twee protocolinstanties die naast elkaar bestaan, met elkaar willen communiceren, dan dient de verzender zijn bericht naar $\mathcal{A}$ te versturen, waarna $\mathcal{A}$ de mogelijkheid heeft om een bericht naar de oorspronkelijke ontvanger te sturen. Merk op dat als $\mathcal{A}$ alleen afluistert (we spreken dan van een *passieve* adversary), dat $\mathcal{A}$ de berichten dan slechts direct doorspeelt naar de ontvanger.

De omgeving houdt bij welke partijen er door de adversary gecorrumpeerd worden. De toestemming om een partij te corrumperen moet van de omgeving $\mathcal{Z}$ komen: Zodra $\mathcal{A}$ door $\mathcal{Z}$ geïnstrueerd wordt een partij μ te corrumperen, stuurt $\mathcal{A}$ een bericht naar μ om de corruptie tot stand te brengen. Vanaf het ontvangstmoment van het bericht door μ laat μ alle informatie en veranderingen in toestand direct aan $\mathcal{A}$ weten, en alle communicatie van μ verloopt nu via $\mathcal{A}$.

Het idee uit sectie 2.5 is te formaliseren tot een notie die we UC-emulatie noemen, waarbij UC staat voor Universal Composability. We noteren met $EXEC_{\pi,\mathcal{A},\mathcal{Z}}$ de uitvoer van $\mathcal{Z}$ na het draaien van het protocol π met adversary $\mathcal{A}$. We zeggen dat een protocol π een ander protocol φ UC-emuleert indien voor iedere PPT adversary $\mathcal{A}$ er een PPT adversary $\mathcal{S}$ bestaat zodanig dat voor alle omgevingen $\mathcal{Z}$ er geldt dat

$$EXEC_{\varphi,\mathcal{S},\mathcal{Z}} \approx_C EXEC_{\pi,\mathcal{A},\mathcal{Z}}.$$

De $\approx_C$ staat hierbij voor 'is computationeel ononderscheidbaar van'. Het UC-emuleren van protocollen onderling induceert een transitieve en reflexieve relatie op de verzameling van protocollen. De reflexiviteit is triviaal – kies $\mathcal{S} = \mathcal{A}$. De transitiviteit van deze relatie volgt uit de volgende redenering.

Lemma 2.2. *Zij π_1, π_2 en π_3 protocollen. Stel dat protocol π_1 protocol π_2 UC-emuleert, en dat protocol π_2 protocol π_3 UC-emuleert. Dan zal protocol π_1 protocol π_3 UC-emuleren.*

Bewijs. Neem aan dat we een adversary $\mathcal{A}_1$ hebben tegen π_1. Per aanname is er nu een PPT $\mathcal{S}_2$ tegen π_2 zodanig dat de uitvoer van $\mathcal{Z}$ bij gebruik van π_2 computationeel ononderscheidbaar is van $\mathcal{Z}$ met π_1. Per aanname is er nu ook een $\mathcal{S}_3$ voor het uitvoeren van $\mathcal{Z}$ met π_3, zodanig dat $EXEC_{\mathcal{S}_2,\pi_2,\mathcal{Z}} \approx_C EXEC_{\mathcal{S}_3,\pi_3,\mathcal{Z}}$. We weten nu dat $EXEC_{\mathcal{A}_1,\pi_1,\mathcal{Z}} \approx_C EXEC_{\mathcal{S}_2,\pi_2,\mathcal{Z}}$ en $EXEC_{\mathcal{S}_2,\pi_2,\mathcal{Z}} \approx_C EXEC_{\mathcal{S}_3,\pi_3,\mathcal{Z}}$. Lemma 2.1 geeft nu het gewenste resultaat, dus dat $EXEC_{\mathcal{A}_1,\pi_1,\mathcal{Z}} \approx_C EXEC_{\mathcal{S}_3,\pi_3,\mathcal{Z}}$. □

Zonder beperking der algemeenheid kunnen we aannemen dat de omgeving en de adversary proberen onderscheid te maken tussen π en φ, dus dat de uitvoer van de omgeving een bit is die de beste gok geeft over de aard van het protocol waarmee gewerkt wordt. De $\mathcal{S}$ die gegeven moet worden, heet $\mathcal{S}$ voor simulator. Immers, als gewerkt wordt met het ideale protocol, dan heeft de adversary veel minder mogelijkheden dan bij het reële protocol. Als er dan toch invoer nodig is om bepaalde berekeningen te doen, dan zullen deze gegokt moeten worden – de adversary $\mathcal{S}$ moet doen *alsof* hij dezelfde mogelijkheden heeft als $\mathcal{A}$, zonder deze te hebben. Indien $\mathcal{S}$ een bepaalde handeling van $\mathcal{A}$ na kan doen op het ideale protocol, dan is deze handeling mogelijk bij iedere implementatie die dit ideale protocol UC-emuleert, niet alleen degene waarop $\mathcal{A}$ werkt.

Hoofdstuk 3

Universal Composition

3.1 Samenstelling van protocollen

Om onze ideeën over emulatie effectief toe te passen, is het belangrijk protocollen te kunnen beschrijven die gebruik maken van andere protocollen als subroutines. Nemen we het voorbeeld van het bezoek aan de groenteboer nogmaals ter hand, dan merken we dat de betaling veel lijkt op de betaling bij een slager, of bij een bakker. We hopen een klein protocol te vinden dat de ideale functionaliteit die bij betaling hoort, emuleert. Als bewezen kan worden dat dit protocol de functionaliteit UC-emuleert, dan weten we zeker dat in welke winkel we ook betalen, als we de functionaliteit van betalen nodig hebben, dit protocol voldoet.

Het samenstellen van protocollen noteren we als volgt. Zij π, ρ, φ protocollen. Als π één of meerdere keren gebruik maakt van φ als subroutine, dan noteren we dat met π^{φ}. Vervangen we nu φ door ρ, dan noteren we het nieuwe protocol met π^{ρ} of, als we duidelijker willen maken dat we φ door ρ vervangen hebben, met $\pi^{\rho/\varphi}$.

Stel dat π niet één aanroep van φ doet, maar meerdere. De ITM van φ wordt dan meerdere keren aangeroepen, waardoor er op bepaalde momenten meerdere instanties van φ uitgevoerd worden. Als we nu instanties van φ één voor één vervangen door ρ, dan levert dat tussenprotocollen op waarbij π gebruik maakt van een aantal instanties van ϕ en een aantal van ρ. Zo'n tussenprotocol noemen we een *hybride protocol*. We zullen zien dat hybride protocollen een centrale rol spelen in het bewijs van de compositiestelling.

3.2 Dummy adversary

In bewijzen die te maken hebben met Universal Composition, zullen we uit een bestaande adversary $\mathcal{A}$ een simulator $\mathcal{S}$ moeten construeren die eenzelfde gedrag van een willekeurige omgeving $\mathcal{Z}$ tot gevolg heeft. Nu is het kwantificeren over alle mogelijke

adversary's niet alleen een vrij ingrijpende bezigheid, het is ook niet nodig. Het blijkt dat het beschouwen van een eenvoudig gedefinieerde adversary, de dummy adversary, genoeg is om een equivalent begrip van emulatie te verkrijgen.

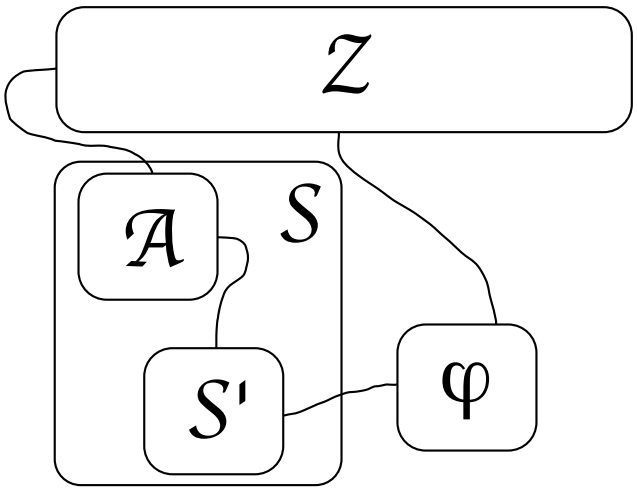
De dummy adversary, die we aan zullen duiden met $\mathcal{D}$, gedraagt zich als volgt. Als er een bericht binnenkomt, dan speelt hij dit direct door aan de omgeving $\mathcal{Z}$. Als $\mathcal{Z}$ een bericht stuurt, dan speelt hij het direct door aan het protocol. Verder bemoeit hij zich niet met de uitvoer van het protocol. In essentie is de dummy adversary de *moeilijkst te simuleren* adversary. Roep immers in herinnering dat de simulator vaak niet de mogelijkheden zal hebben van de adversary, en dat hij deze daarom zal moeten simuleren. Hoe meer gegevens de simulator aan de omgeving af moet staan, hoe lastiger het zal zijn deze op een overtuigende manier te simuleren. We maken dit exact in het volgende lemma:

Lemma 3.1. *Zij ρ, φ protocollen. Dan UC-emuleert ρ φ dan en slechts dan als ρ φ UC-emuleert voor de dummy adversary $\mathcal{D}$.*

Bewijs. Van links naar rechts is de mededeling triviaal. Immers, als φ door ρ UC-geëmuleerd wordt, dan geldt dat voor alle adversary's, dus in het bijzonder ook voor $\mathcal{D}$. Stel nu dat φ UC-geëmuleerd wordt door ρ voor de dummy adversary $\mathcal{D}$. Zij $\mathcal{S}'$ de door de definitie gegeven simulator voor $\mathcal{D}$, zodat $EXEC_{\mathcal{D},\rho,\mathcal{Z}} \approx_C EXEC_{\mathcal{S}',\varphi,\mathcal{Z}}$. Zij $\mathcal{A}$ een gegeven PPT adversary tegen ρ. We zoeken nu een $\mathcal{S}$, zodanig dat $EXEC_{\mathcal{A},\rho,\mathcal{Z}} \approx_C EXEC_{\mathcal{S},\varphi,\mathcal{Z}}$, aangezien dan volgt dat ρ φ UC-emuleert. We construeren $\mathcal{S}$ tegen φ als volgt. Als $\mathcal{S}$ aangeroepen wordt door de omgeving, dan roept deze een instantie van $\mathcal{A}$ en een instantie van $\mathcal{S}$' aan. Alle berichten die $\mathcal{S}$ ontvangt van $\mathcal{Z}$, stuurt hij direct door aan $\mathcal{A}$. Berichten van $\mathcal{A}$ die voor ρ bedoeld zijn, worden bij $\mathcal{S}$' afgeleverd. Berichten van $\mathcal{A}$ die voor $\mathcal{Z}$ bedoeld zijn, stuurt $\mathcal{S}$ meteen door aan $\mathcal{Z}$. Berichten van $\mathcal{S}$' die voor φ bedoeld zijn, stuurt $\mathcal{S}$ door aan φ. Berichten van $\mathcal{S}$' die voor $\mathcal{Z}$ bedoeld zijn, stuurt $\mathcal{S}$ door aan $\mathcal{A}$ alsof een instantie van ρ dit net geantwoord heeft.
De gedachte hierachter is dat $\mathcal{S}'$ per definitie iedere omgeving kan overtuigen met $\mathcal{D}$ en ρ van doen te hebben. Stel nu dat er een omgeving $\mathcal{Z}$ bestaat die onderscheid kan maken tussen $\varphi, \mathcal{S}$ en $\rho, \mathcal{A}$. We laten onze constructie intact, maar vatten de verschillende onderdelen nu anders op. Zij $\mathcal{Z}'$ de samenvoeging van $\mathcal{Z}$ en $\mathcal{A}$, waarbij de invoer van $\mathcal{Z}$' direct aan $\mathcal{Z}$ wordt doorgespeeld, en resultaten van $\mathcal{Z}$ komen direct als resultaat van $\mathcal{Z}$' naar buiten. $\mathcal{Z}$ zal een instantie van $\mathcal{A}$ aanroepen, en we sturen de communicatie van $\mathcal{A}$ die voor ρ bedoeld is, door aan $\mathcal{S}$'. We zullen $\mathcal{Z}$' opvatten als omgeving. Merk op dat $EXEC_{\varphi,\mathcal{S},\mathcal{Z}} = EXEC_{\varphi,\mathcal{S}',\mathcal{Z}'}$, met andere woorden: we hebben alleen de onderdelen opnieuw gegroepeerd (zie ook figuur 3.2). Deze zelfde hergroepering kan op het drietal $\rho, \mathcal{A}, \mathcal{Z}$ toegepast worden om $\rho, \mathcal{D}, \mathcal{Z}'$ te krijgen. Ook deze verandering van namen heeft geen effect op de uitvoer van de omgeving. Echter, we zien dat

$$EXEC_{\rho,\mathcal{D},\mathcal{Z}'} = EXEC_{\rho,\mathcal{A},\mathcal{Z}} \not\approx_C EXEC_{\varphi,\mathcal{S},\mathcal{Z}} = EXEC_{\varphi,\mathcal{S}',\mathcal{Z}'}.$$

Zodoende concluderen we dat $\mathcal{Z}'$ een omgeving is die de UC-emulatie voor $\mathcal{D}$ van φ door ρ tegenspreekt, en dat $\mathcal{Z}$ dus niet kan bestaan, dus dat ρ φ voor alle adversary's UC-emuleert. □



Figuur 3.1: Constructie van een simulator tegen φ

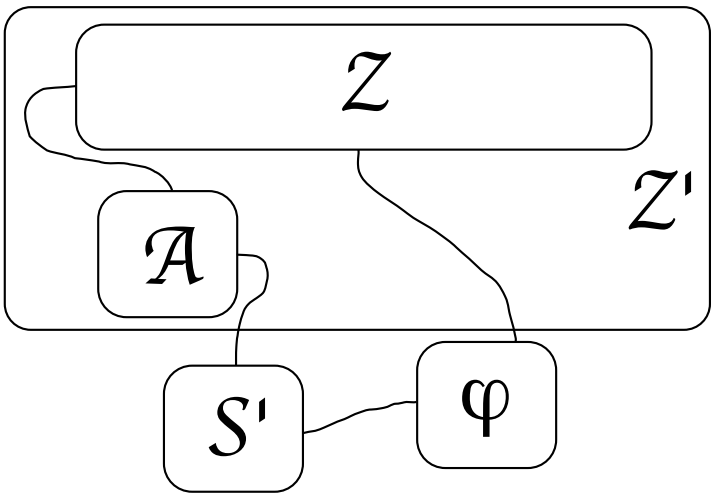
Zie voor een uitgebreidere behandeling van dit bewijs subsectie 4.3.1 van [Canetti, 2005].

3.3 Compositiestelling

De bruikbaarheid van kleine protocollen als bouwblokken voor het maken van grotere, complexere protocollen zoals we die in de vorige sectie bespraken, is geen uitspraak die slechts een intuïtieve rechtvaardiging heeft. Met behulp van de ideeën die we in de voorgaande paragrafen hebben behandeld, kunnen we de compositiestelling formuleren. De compositiestelling is een centrale stelling in het raamwerk van Universal Composability en is in feite de rechtvaardiging van de set gereedschap die we ontwikkeld hebben. De compositie van protocollen is een graag gebruikte techniek om complexe protocollen op te bouwen. Immers, 'iemand betaalt een bedrag aan iemand anders' is een functionaliteit die in meerdere omgevingen nuttig kan zijn: Denk aan een internetbankieromgeving, een online casino of een internetveiling. Het is wenselijk dat de component die de betaling regelt, zich op een bepaalde manier gedraagt die niet afhangt van de omgeving waar deze in gebruikt wordt. Van dergelijke componenten zijn veel meer voorbeelden te geven: denk bijvoorbeeld aan 'communiceren zonder afgeluisterd te worden', 'een groep mensen laten stemmen', enzovoorts. De stelling luidt als volgt:

Stelling 3.2 (Compositiestelling)**.** *Zij φ, ρ protocollen. Zij π een protocol dat instanties van ρ of φ aanroept. Stel: ρ UC-emuleert φ. Dan geldt: π^ρ UC-emuleert π^φ.*

Deze stelling belichaamt inderdaad de manier waarop we de bouwstenen graag gebruikt zien. Als niemand onderscheid kan maken tussen het ene subprotocol en het andere, dan kan het grote protocol dat ook niet, en als het grote protocol niet essentieel anders draait met het ene subprotocol dan met het andere, dan zal ook aan de uitvoer van het



Figuur 3.2: Reductie van deze simulator tot een simulator van $\mathcal{D}$

grote protocol weinig veranderen door deze wijziging.

3.4 Bewijs van de compositiestelling

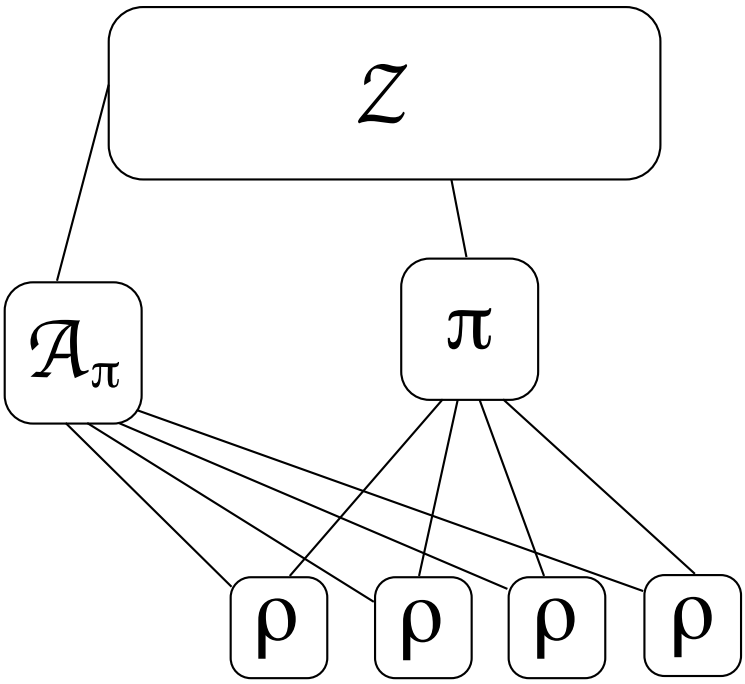
Het bewijs van de compositiestelling is gedetailleerd terug te vinden in [Canetti, 2005], sectie 5.2. Een weergave hiervan waarbij niet in wordt gegaan op deze details, is hieronder te vinden.

Bewijs van de compositiestelling. We zullen lemma 3.1 gebruiken om het bewijs te vereenvoudigen, waardoor we alleen nog rekening hoeven te houden met de dummy adversary $\mathcal{D}$ [1]. Zij π, ρ en φ protocollen zodanig dat π gebruik maakt van φ en ρ φ UC-emuleert. Zij $\pi^\rho = \pi^{\rho/\varphi}$ het protocol π waarin aanroepen van instanties φ vervangen zijn door instanties ρ (zie figuur 3.3). We gebruiken lemma 3.1 op de definitie van UC-emulatie en hoeven dan niet meer te kwantificeren over alle adversary's $\mathcal{A}$. Nu dient een PPT simulator $\mathcal{A}_\pi$ voor $\mathcal{D}$ geconstrueerd te worden die voor alle PPT omgevingen $\mathcal{Z}$ aan het volgende voldoet:

$$EXEC_{\pi^\rho,\mathcal{D},\mathcal{Z}} \approx_C EXEC_{\pi,\mathcal{A}_\pi,\mathcal{Z}}.$$

Zij $\mathcal{S}$ de door de definitie van UC-emulatie bestaande simulator voor de dummy adversary tegen ρ. We construeren simulator $\mathcal{A}_\pi$ als volgt. Roep in herinnering dat $\mathcal{A}_\pi$ zich moet gedragen alsof de omgeving met $\mathcal{D}$ van doen heeft, en dus moet doen alsof hij data doorspeelt die door π en instanties van ρ gegenereerd worden. Hiertoe communiceert $\mathcal{A}_\pi$ direct met π, net als de dummy adversary, en draait $\mathcal{A}_\pi$ een instantie van $\mathcal{S}$ voor iedere

[1] Merk op dat we de definitie van UC-emulatie ook ten opzichte van de dummy adversary hadden kunnen formuleren, en lemma 3.1 opnemen om de algemeenheid te ondersteunen. De hier gebruikte notie geeft een intuïtievere definitie.



Figuur 3.3: Weergave van de aanroepen die π doet naar ρ en de plaats van $\mathcal{A}_\pi$

instantie van φ die aangeroepen wordt door π. Immers, $\mathcal{S}$ is een PPT die de uitvoer van ρ kan simuleren als hij met φ communiceert. We vatten $\mathcal{A}_\pi$ op als simulator tegen π na vervanging van een aantal (mogelijk niet alle) instanties van φ door ρ. Communicatie met instanties van ρ doet $\mathcal{A}_\pi$ ook direct, net als de dummy adversary. We gebruiken een hybride argument om de geldigheid van $\mathcal{A}_\pi$ te duiden. Zij k de securityparameter van het systeem (die door de omgeving wordt doorgegeven). Zij $m(k)$ een bovengrens aan het aantal instanties van φ dat aangeroepen wordt door π. We definiëren nu de volgende tussenprotocollen. Zij $0 \leq l \leq m(k)$, dan is π_l het protocol dat in de eerste l instanties van het subprotocol φ gebruikt, maar de rest vervangt door aanroepen naar ρ. In het bijzonder is $\pi_0 = \pi^\rho$ en $\pi_{m(k)} = \pi$ (zie ook figuur 3.4 voor een voorbeeld). Neem nu aan dat er een omgeving $\mathcal{Z}$ is waarvoor geldt dat

$$EXEC_{\pi,\mathcal{A}_\pi,\mathcal{Z}} \not\approx_C EXEC_{\pi^{\rho/\varphi},\mathcal{D},\mathcal{Z}}.$$

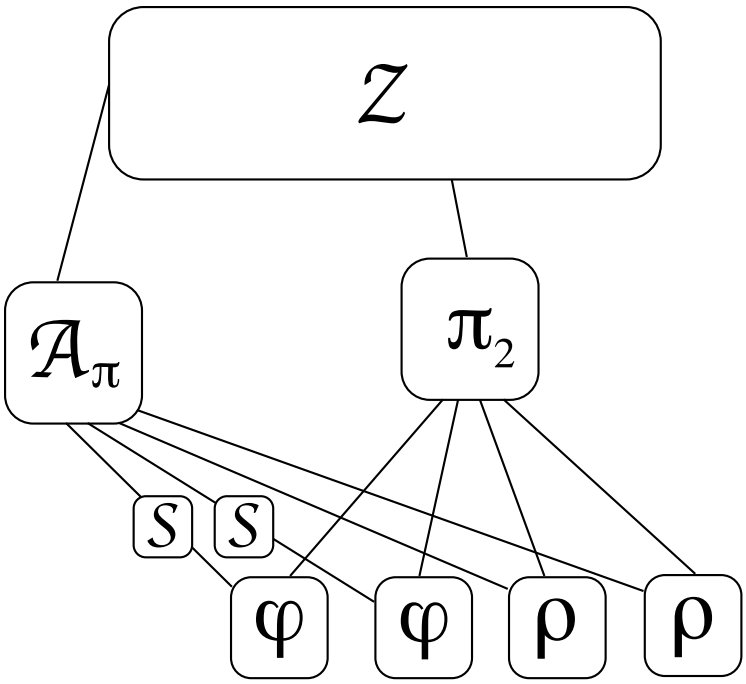
Zij nu

$$\varepsilon = EXEC_{\pi,\mathcal{A}_\pi,\mathcal{Z}} - EXEC_{\pi^{\rho/\varphi},\mathcal{D},\mathcal{Z}} \neq \mathsf{negl}(n).$$

Op basis van het duiventilprincipe moet er een l zijn, $0 \leq l \leq m(k)$, zodanig dat

$$EXEC_{\pi_l,\mathcal{A}_\pi,\mathcal{Z}} - EXEC_{\pi_{l+1},\mathcal{A}_\pi,\mathcal{Z}} \geq \frac{\varepsilon}{m} \neq \mathsf{negl}(n).$$

We construeren nu een nieuwe omgeving $\mathcal{Z}_l$ die onderscheid zal gaan maken tussen φ en ρ. $\mathcal{Z}_l$ draait intern de vorige omgeving $\mathcal{Z}$, net als π en alle aangeroepen instanties van

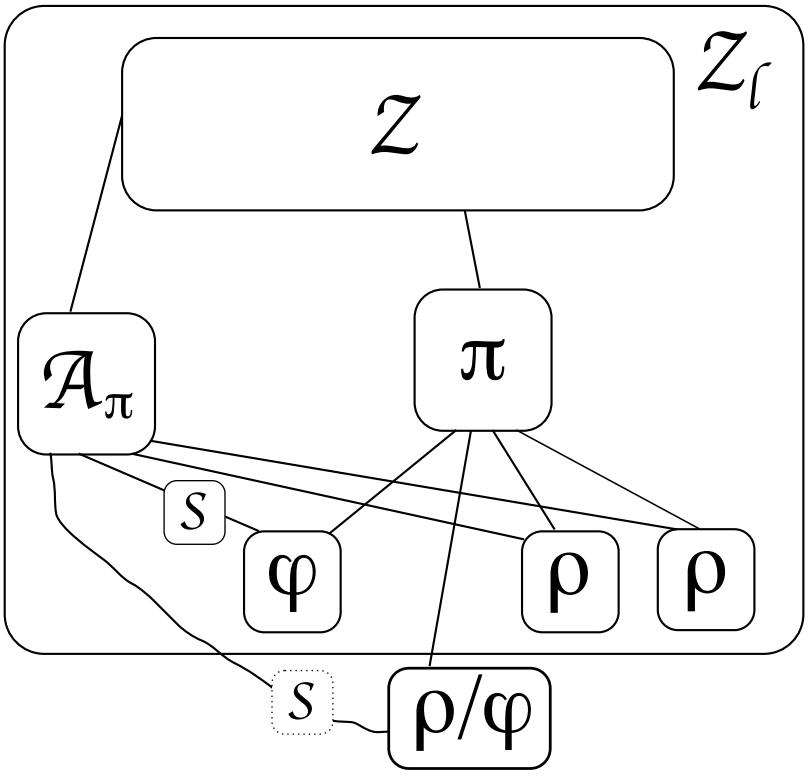


Figuur 3.4: π_2, één van de hybride tussenprotocollen

ρ en φ, behalve één, namelijk de l-de instantie. Omdat l af kan hangen van de keuze van de securityparameter k en we $\mathcal{Z}_l$ niet voor iedere securityparameter opnieuw kunnen kiezen, wordt deze bij de input meegegeven, dus als de input van $\mathcal{Z}$ gelijk was aan z, dan is de input van $\mathcal{Z}_l$ gelijk aan (z, l), zodat $\mathcal{Z}_l$ weet welke variant van het hybride model gedraaid moet worden. De uitvoer van $\mathcal{Z}_l$ is de uitvoer van $\mathcal{Z}$ als deze klaar is. Aanroepen naar de l-de instantie worden doorgespeeld aan een werkelijke φ of ρ, en aanroepen naar de adversary worden doorgespeeld aan respectievelijk $\mathcal{S}$ of $\mathcal{D}$. Zie ook figuur 3.5. Aangezien de uitvoer van $\mathcal{Z}$ gelijk is aan de uitvoer van $\mathcal{Z}_l$, geldt

$$EXEC_{\varphi,\mathcal{S},\mathcal{Z}_l} = EXEC_{\varphi,\mathcal{A}_\pi,\mathcal{Z}} \not\approx_C EXEC_{\rho,\mathcal{A}_\pi,\mathcal{Z}} = EXEC_{\rho,\mathcal{D},\mathcal{Z}_l}.$$

Dit is echter in tegenspraak met onze aanname dat ρ φ UC-emuleert, dus UC-emuleert $\pi^{\rho/\varphi}$ π. □



Figuur 3.5: Het samennemen van componenten geeft de nieuwe omgeving $\mathcal{Z}_l$

Hoofdstuk 4

Indifferentiability

In de voorgaande hoofdstukken zijn we bij het introduceren van terminologie en concepten uitgegaan van het raamwerk zoals dat door Ran Canetti in [Canetti, 2005] geïntroduceerd wordt. Echter, dergelijke analyses van bouwstenen in de constructie van complexe protocollen zijn door anderen op andere manieren gedaan. Zo hebben Ueli Maurer, Renato Renner en Clemens Holenstein in [Maurer et al., 2004] het begrip Indifferentiability gedefinieerd en gebruikt. Dit hoofdstuk zal de voorgaande definities slechts als beginpunt en vergelijkingsmateriaal gebruiken ten opzichte van de concepten die door deze heren gebruikt worden.

4.1 Begrippen

In de voorgaande secties was de gebruikte primitieve in de analyse van protocollen een ITM, een interactieve Turingmachine. Deze interpretatie heeft veel voordelen, omdat deze ons een inzichtelijke manier geeft om de volgorde van gebeurtenissen te duiden. Echter, het is eenvoudig de weg kwijt te raken in het bos van aanroepen, proces- en sessie-identiteiten en communicatiestroken. In het kader van Indifferentiability spreken we over systemen.

Definitie 4.1. *Noteren we* $X^i = [X_1, \ldots, X_i]$ *en* $Y^i = [Y_1, \ldots, Y_i]$ *voor natuurlijke* i*, dan is een* $(\mathcal{X}, \mathcal{Y})$-systeem *een rij conditionele kansverdelingen* $P_{Y_i|X^iY^{i-1}}$ *(met* $i \in \mathbb{N}$*), waarbij* X_i*, de* i*-de invoer, en* Y_i*, de* i*-de uitvoer, stochasten zijn met respectievelijk* $\mathcal{X}$ *en* $\mathcal{Y}$ *als uitslagenruimte.*

Intuïtief gezien correspondeert dit met het gevoel dat een systeem gedefinieerd wordt door het (eventueel stochastische) gedrag na een bepaalde invoer en voorgaande uitvoer. Merk op dat ITM's op een natuurlijke wijze als systemen op te vatten zijn. Immers, een ITM ontvangt invoer van buitenaf en verwerkt deze (via een algoritme, maar dat is voor een systeem niet van belang) tot een eventueel stochastisch bepaalde uitvoer. In de bespreking van computationele mogelijkheden van protocollen kwam de term 'PPT'

langs, waarbij 'polynomiaal' refereerde aan het aantal rekenstappen als functie van de securityparameter en de lengte van de invoer. Bij de notie van systemen hebben we helaas geen eenheid van rekentijd, omdat we geen expliciete berekenmethoden beschrijven. Daarom wordt een zogeheten kostenfunctie gedefinieerd die bij een systeem met een bepaalde invoer de kosten van het produceren van de uitvoer beschrijft. Deze functie moet zich ook daadwerkelijk als kostenfunctie gedragen, dus als een systeem andere systemen als component gebruikt, dan zal het totaal van de kosten voor dat systeem ten minste gelijk zijn aan de som van de kosten van de aangeroepen instanties.

Definitie 4.2. *Zij $\mathcal{S}_{k\in\mathbb{N}}$ een familie systemen die geïndexeerd wordt door de securityparameter $k \in \mathbb{N}$. Is er een polynoom in de lengte van de invoer en de securityparameter k dat de kostenfunctie voor alle invoer en securityparameters begrenst, dan zeggen we dat* $\mathcal{S}$ kostenefficiënt *is.*

Aangezien we in dit raamwerk werken met kansverdelingen, is de notie van entropie gemakkelijk toepasbaar, waardoor onmogelijkheidsresultaten af te leiden zijn over bepaalde systemen met zekere kostenfuncties. Bedenk bijvoorbeeld dat als een component tegen bepaalde kosten bepaalde informatie oplevert met een zekere entropie, dat dan een bovengrens gegeven kan worden voor hoeveel informatie kan worden verwerkt door systemen die deze component gebruiken. Wij gaan hier niet verder op in: in [Maurer et al., 2004] wordt dit uitgebreider toegelicht en toegepast.

4.2 Indistinguishable en indifferentiable

Om een systeem als cryptosysteem op te vatten, splitsen we de invoerzijde van het systeem in tweeën, zodat er een publieke interface en een private interface beschikbaar is. Zo'n systeem noemen we een *resource*. Een cryptosysteem vatten we op als een resource, dus met een publieke en een private interface. Deze interfaces modelleren respectievelijk de toegang van de adversary en van de eerlijke partijen. Net als eerder beschouwen we de veiligheid van een cryptosysteem slechts ten opzichte van een ideaal cryptosysteem, dat per aanname bepaalde eigenschappen heeft. Deze relatie op cryptosystemen definiëren we als volgt:

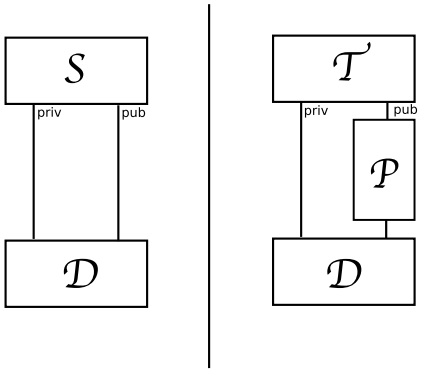
Definitie 4.3. *We noemen $\mathcal{C}$* tenminste zo veilig als *$\mathcal{C}'$, notatie $\mathcal{C} \succ \mathcal{C}'$, als voor alle omgevingen*[1] *$\mathcal{Z}$ het volgende geldt: Voor iedere aanvaller $\mathcal{A}$ die $\mathcal{C}$ gebruikt, is er een aanvaller $\mathcal{A}'$ die $\mathcal{C}'$ gebruikt zodanig dat*

$$|\Pr[\mathcal{Z}(\mathcal{C}^{priv}, \mathcal{A}(C^{pub})) = 1] - \Pr[\mathcal{Z}(\mathcal{C}'^{priv}, \mathcal{A}'(C'^{pub})) = 1]|$$

verwaarloosbaar is in de securityparameter k. Een cryptosysteem is computationeel tenminste zo veilig als *een ander als bovendien $\mathcal{Z}$,$\mathcal{A}$ en $\mathcal{A}'$ computationeel efficiënte algoritmen zijn.*

We definiëren de volgende term, die gebruikt kan worden voor systemen in het algemeen:

[1] We nemen opnieuw zonder beperking der algemeenheid aan dat de uitvoer binair is.



Figuur 4.1: Weergave van de verbindingen zoals bedoeld in definitie 4.5.

Definitie 4.4. *Zij $\mathcal{S}$ en $\mathcal{T}$ systemen. We noemen $\mathcal{S}$ en $\mathcal{T}$* indistinguishable[2] *als voor ieder algoritme $\mathcal{D}$ dat met deze systemen communiceert en een binaire uitvoer oplevert, geldt dat*

$$|\Pr[\mathcal{D}(\mathcal{S}_k) = 1] - \Pr[\mathcal{D}(\mathcal{T}_k) = 1]|$$

verwaarloosbaar is in de securityparameter k. Indistinguishability is computationeel als $\mathcal{D}$ computationeel efficiënt is in k.

Deze definitie zouden we graag toepassen op cryptosystemen. Echter, het blijkt dat het direct overnemen ervan een te sterke eis op de cryptosystemen legt. Daarom is de definitie aangepast tot de volgende:
Definitie 4.5. *Zij $\mathcal{S}$ en $\mathcal{T}$ resources. We noemen $\mathcal{S}$ en $\mathcal{T}$* indifferentiable*, notatie $\mathcal{S} \sqsubset \mathcal{T}$, als voor ieder algoritme $\mathcal{D}$ met binaire uitvoer er een algoritme $\mathcal{P}$ bestaat zodanig dat*

$$|\Pr[\mathcal{D}(\mathcal{S}_k^{priv}, S_k^{pub}) = 1] - \Pr[\mathcal{D}(\mathcal{T}_k^{priv}, \mathcal{P}(S_k^{pub})) = 1]|$$

verwaarloosbaar is in de securityparameter k. Zie ook figuur 4.1. Indifferentiability is computationeel als slechts computationeel efficiënte algoritmes gebruikt worden voor $\mathcal{D}$ en $\mathcal{P}$.

4.3 Compositiestelling en problemen bij het bewijzen

Deze definities nodigen uit tot het overbrengen van de compositiestelling naar dit raamwerk. Deze is in dit kader als volgt te formuleren:
Stelling 4.1. *Zij $\mathcal{S}$ en $\mathcal{T}$ resources. Laat $\mathfrak{C}$ de verzameling van alle cryptosystemen zijn. Dan geldt*

$$\mathcal{S} \sqsubset \mathcal{T} \Leftrightarrow \forall \mathcal{C} \in \mathfrak{C} : \mathcal{C}(\mathcal{S}) \succ \mathcal{C}(\mathcal{T}).$$

Dit geldt ook als beide zijden computationeel beschouwd worden.

[2]Voor deze term, en ook voor het later gedefinieerde *indifferentiable*, gebruiken we de Engelse uitdrukking, omdat het subtiele betekenisverschil tussen de twee in het Nederlands bij vertaling wegvalt.

Proberen we deze stelling te bewijzen in het raamwerk van indifferentiability, dan is dit geen automatische vertaaloefening. We lopen tegen meerdere problemen aan bij het overzetten van het bewijs zoals beschreven in sectie 3.4. Er worden hier enkele problemen genoemd, en suggesties gedaan voor de aanpak ervan.

Volgorde van kwantoren

Zoals we zien in de definitie van *tenminste zo veilig als*, mag de simulator die gemaakt wordt als reactie op een aanvaller afhangen van de omgeving waar deze in gebruikt wordt. Dit geeft een zwakkere veiligheidsnotie, i.e. deze geldt in meer gevallen. Het blijkt dat de algemenere formulering van de compositiestelling, waarbij een polynomiaal aantal aanroepen naar de component $\mathcal{S}$ of $\mathcal{T}$ gedaan mag worden, helaas niet van links naar rechts ("⇒") bewijsbaar is in dit zwakkere kader. Mogelijkerwijs kan dit wel als de simulator niet af mag hangen van de omgeving. Dit leidt tot de volgende aangepaste definitie:

Definitie 4.6. *We noemen* $\mathcal{C}$ tenminste zo veilig als $\mathcal{C}'$*, notatie* $\mathcal{C} \succ \mathcal{C}'$*, als het volgende geldt: Voor iedere aanvaller* $\mathcal{A}$ *die* $\mathcal{C}$ *gebruikt, is er een aanvaller* $\mathcal{A}'$ *die* $\mathcal{C}'$ *gebruikt zodanig dat* ***voor alle omgevingen*** $\mathcal{Z}$

$$|\Pr[\mathcal{Z}(\mathcal{C}^{priv}, \mathcal{A}(\mathcal{C}^{pub})) = 1] - \Pr[\mathcal{Z}(\mathcal{C}'^{priv}, \mathcal{A}'(\mathcal{C}'^{pub})) = 1]|$$

verwaarloosbaar is in de securityparameter k*. Een cryptosysteem is* computationeel tenminste zo veilig als *een ander als bovendien* $\mathcal{Z}$*,*$\mathcal{A}$ *en* $\mathcal{A}'$ *computationeel efficiënte algoritmen zijn.*

Merk op dat de zinsnede 'voor alle omgevingen $\mathcal{Z}$' verder naar rechts verplaatst is, zodat $\mathcal{A}'$ niet meer af kan hangen van $\mathcal{Z}$.

Dummy adversary

Het bewijs van de compositiestelling in het UC-raamwerk wordt aanzienlijk gesimplificeerd door het gebruik van de dummy adversary. Hierdoor hoeft niet meer over alle mogelijke aanvallers op de reële implementatie van het protocol gekwantificeerd te worden, en worden de redeneringen vereenvoudigd. Het bewijs kan ook zonder deze dummy adversary opgesteld worden. Verder kan de dummy adversary overgebracht worden naar het raamwerk van indifferentiability, met een analogie van lemma 3.1 voor resources:

Lemma 4.2. *Zij* $\mathcal{C}, \mathcal{C}'$ *resources. Dan geldt:* $\mathcal{C} \succ \mathcal{C}'$ *dan en slechts dan als* $\mathcal{C} \succ \mathcal{C}'$ *met betrekking tot de dummy aanvaller, die als uitvoer altijd met kans 1 zijn invoer teruggeeft.*

Meerdere protocolinstanties en niet-uniforme input

In het bewijs van Canetti voor de compositiestelling is ruimte voor protocollen die meerdere instanties van het subprotocol aanroepen. Het aantal instanties dat aangeroepen

mag worden is begrensd door een polynoom in de securityparameter, en dit vraagt om een omzichtiger aanpak bij het bewijzen van de stelling. Het raamwerk van indifferentiability ondersteunt dit niet van nature, en zal daarom aangepast moeten worden om dergelijke protocollen ook te ondersteunen. Merk op dat het niet voldoende is om een flink (vast) aantal subprotocolinstanties 'alvast' aan te roepen en te verbinden met het hoofdprotocol: Omdat de bovengrens polynomiaal is, is kan aantal subprotocolinstanties in principe willekeurig groot worden.

In het bewijs van de compositiestelling wordt één van de instanties van het subprotocol 'buitengesloten'. De index van deze instantie, die in het bewijs l heet, wordt aan $\mathcal{Z}_\rho$ meegegeven als invoer, opdat deze weet welke variant van het hybride model gedraaid moet worden. In het ITM-rekenmodel kon dit op een natuurlijke wijze doorgegeven worden. Het probleem is, dat deze informatie in de nu gestelde definitie van systemen, resources en indifferentiability, lastig of niet door te geven is aan $\mathcal{Z}_\rho$. Hier zal ook een aanpassing nodig zijn van de gestelde termen.

Bibliografie

Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. IACR ePrint Archive, January 2005.

Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *Theory of Cryptography, Proceedings of TCC 2004*, number 2951 in Lecture Notes in Computer Science, pages 21–39. Springer-Verlag, 2004.