Typing and Compositionality for Stateful Security Protocols

Hess, Andreas Viktor

Publication date:
2019

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Hess, A. V. (2019). Typing and Compositionality for Stateful Security Protocols. DTU Compute. DTU Compute PHD-2018, Vol.. 495

Typing and Compositionality for Stateful Security Protocols

Andreas Viktor Hess

Kongens Lyngby 2018

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

On the Internet computers communicate using security protocols. For example, when using an online banking application a client may log in with NemID, ensuring that the bank can authenticate the client. Similarly, the TLS protocol provides a confidential communication channel between the client and the bank ensuring that the data they exchange are protected from eavesdroppers and authenticates the bank to the client.

A question is whether protocols are designed well enough so that they always achieve their security goals like confidentiality and authentication. If not, an attacker may be able to hijack and manipulate communication to break the goals, even when all honest participants only use the protocols as intended. To combat this, protocols are formally verified which provides strong guarantees that their goals cannot be violated even in the presence of dishonest participants and powerful adversaries that control the communication network. There is a rich body of research on the formal verification of security protocols, i.e., mathematically proving the correctness. However, significantly less literature exists on the *composition* of protocols: In the banking example the TLS protocol usually provides a secure channel over which NemID runs, that is, the protocols are used in a composed fashion. It turns out that such composed protocols might not be secure even when their component protocols have been verified in isolation.

Composed protocols, however, can be rather difficult to verify since they are often too big for automated methods to verify in a reasonable amount of time. Worse still, a rather large amount of protocols are used on the Internet, and their number is growing; one cannot possibly verify the composition of all of them.

To mitigate these problems, we prove several compositionality results that have the following basic form: given a number of protocols that are secure in isolation and that satisfy a number of simple prerequisites, then also their composition is secure. Hence compositionality reduces a complex protocol verification problem to easier ones.

Existing compositionality results are for relatively simple “stateless” protocols, where the behavior of the protocol participants only depends on the messages they receive within a single session. In this work we present the first compositionality result for *stateful protocols*, where each participant may additionally maintain databases, meaning that the behavior of the participants also depends on what is contained in the databases. For instance, a keyserver may maintain a database of currently valid public keys. As part of the protocol, participants can insert entries into, or delete entries from, their databases. Moreover, the databases may be used in more than one protocol, for instance, the keyserver may offer several different protocols for establishing new keys, and revoking and updating existing keys. The contribution of this thesis is to provide a very general result for the composition of such stateful protocols and how they coordinate the access to any shared databases.

The proofs of compositionality results, however, are long and often contain subtle details. If there is a glitch in the proof of a compositionality theorem, then we may falsely rely on the security of a composition that is actually insecure. In fact, as part of this work we have found several such mistakes in existing compositionality results.

The problem of flawed mathematical proofs due to their complexity is very old. A new way to mitigate this problem are proof assistants like Isabelle/HOL, that require the user to formulate a proof in such detail that a computer can check the proof. Since this relies only on the correctness of a very small core program that checks basic logical deductions, it is virtually impossible to conduct a flawed proof that is accepted by the proof checker.

Compositionality results are a lot easier to prove if one assumes a *typed model* where ill-typed attacks cannot happen. *Typing results* shows that this is a sound restriction for a large class of protocols. Such results are useful by themselves, in that they can make protocol verification faster for automated tools. They also combine very well with compositionality results, both in terms of methods and in terms of requirements on protocols. Therefore we use typing results as a basis for compositionality results. As part of this thesis we formalize an existing typing result in Isabelle/HOL and we establish and formalize in Isabelle/HOL a new typing result for stateful protocols.

The main objective of this thesis is to formalize and prove existing and new

compositionality results in Isabelle/HOL. This means that we can be sure that the compositionality results we present here are really correct. Moreover, if we have an Isabelle/HOL proof that a set of given protocols is each secure in isolation and satisfies the prerequisites of the compositionality theorem, then we immediately obtain a complete Isabelle/HOL proof that their composition is also secure.

Summary (Danish)

Titel: Typing og Sammensætning for Tilstandsfulde Sikkerhedsprotokoller

På Internettet kommunikerer computere ved brug af sikkerhedsprotokoller. For eksempel, når en online bankapplikation bliver brugt så kan en klient logge ind med NemID, hvilket sikrer at banken kan autentificere klienten. Tilsvarende kan TLS protokollen give en konfidentiel kommunikationskanal mellem klienten og banken, hvilket sikrer at dataene der bliver udvekslet er beskyttet mod aflytning og som autentificerer banken til klienten.

Et spørgsmål er, om protokoller er designet godt nok til at de altid opnår deres sikkerhedsmål såsom konfidentialitet og autenticitet. Hvis ikke så kan en angriber måske kapre og manipulere kommunikation for at bryde målene, selv hvis alle parter kun bruger protokollen som tilsigtet. For at bekæmpe dette så er protokoller ofte formelt verificerede, hvilket giver stærke garantier for at deres mål ikke kan blive krænket, selv under tilstedeværelsen af uærlige deltagere og magtfulde modstandere som kontrollerer kommunikationsnetværket. Der har været masser af forskning inden for formel verifikation af sikkerhedsprotokoller, dvs. matematisk at bevise deres korrekthed. Dog eksisterer der væsentligt mindre litteratur inden for *sammensætningen* af protokoller: I bankeksemplet bliver TLS protokollen ofte brugt til at give en sikker kanal over hvilken NemID kører, dvs. protokollerne bliver brugt på en sammensat måde. Det viser sig at sådanne sammensatte protokoller ikke nødvendigvis er sikre selv om deres komponent-protokoller er blevet formelt verificerede i isolation.

Sammensatte protokoller kan også være ret svære at verificere fordi de ofte er for store til at blive verificeret af automatiske metoder inden for en rimelig tidsperiode. Endnu værre, så bliver der på Internettet brugt et ret stort antal af protokoller, og dette antal stiger; det er urimeligt at tro at sammensætningen af alle de protokoller kan verificeres. For at afhjælpe disse problemer så beviser vi adskillelige sammensætningsteoremer som har følgende grundlæggende form: hvis et givet antal protokoller hver især er sikre i isolation, og som opfylder et antal enkle forudsætninger, så er deres sammensætning også sikker. Sammensætningteoremer reducerer dermed et komplekst protokolverifikationsproblem til et enklere problem.

Eksisterende sammensætningsresultater virker kun for simple "tilstandsløse" protokoller, hvor protokoldeltagernes adfærd kun er afhængig af de beskeder de modtager indenfor en enkelt session. I denne afhandling præsenterer vi det første sammensætningsresultat for *tilstandsfulde protokoller*, hvor hver deltager også kan vedligeholde databaser, hvilket betyder at deltagernes adfærd også afhænger af hvad databaserne indeholder. En nøgleserver kan, for eksempel, vedligeholde en database indeholdende gyldige offentlige nøgler. Som en del af protokollen så kan deltagerer indsætte poster i, eller fjerne poster fra, deres databaser. Databaserne kan endvidere blive brugt i mere end én protokol af gangen. Nøgleserveren kan for eksempel udbyde flere forskellige protokoller til at etablere nye nøgler, og annullere og opdatere eksisterende nøgler. Denne afhandlings bidrag er at give et meget generelt resultat for sammensætningen af sådanne tilstandsfulde protokoller og hvordan de koordinerer adgangen til delte databaser.

Beviserne for sammensætningsresultater er lange og indeholder ofte subtile detaljer. Hvis der er en fejl i et bevis for et sammensætningsteorem så risikerer vi fejlagtigt at stole på sikkerheden af en sammensætning af protokoller som egentlig er usikker. Som en del af dette arbejde har vi endda fundet adskillelige af sådanne fejl i eksisterende sammensætningsresultater.

Problemet med fejl i matematiske beviser på grund af deres kompleksitet er meget gammel. En ny måde at afhjælpe dette problem er bevisassistenter såsom Isabelle/HOL, som kræver at brugeren formulerer beviser i et så detaljeret omfang at en computer selv kan tjekke beviserne. Da dette kun afhænger af en lille kerne program der tjekker basale logiske deduktioner, så er det næsten umuligt at udføre et fejlagtigt bevis som vil blive accepteret af bevistjekkeren.

Sammensætningsresultater er meget nemmere at bevise hvis man antager en *typed model* hvor såkaldte dårligt-typede angreb ikke kan finde sted. *Typeresultater* viser at dette er en sund restriktion for en stor klasse af protokoller. Sådanne resultater er i sig selv nyttige, idet de kan gøre protokolverifikation hurtigere for automatiserede værktøjer. De passer også meget godt sammen med sammensætningsresultater, både hvad angår metoder og hvad angår kravene til protokoller. Derfor bruger vi typeresultater som grundlag for sammensætningsresultater. Som en del af denne afhandling så formaliserer vi eksisterende

typeresultater i Isabelle/HOL og vi etablerer og formaliserer i Isabelle/HOL et nyt typeresultat for tilstandsfulde protokoller.

Hovedformålet med denne afhandling er at formalisere og bevise eksisterende og nye sammensætningsresultater i Isabelle/HOL. Dette betyder, at vi kan være sikre på at sammensætningsresultaterne som vi præsenterer her rent faktisk er korrekte. Derudover, hvis vi har et bevis i Isabelle/HOL for at nogle protokoller hver for sig er sikre i isolation, og som opfylder kravene for vores sammensætningsteorem, så kan vi med det samme få fat på et fuldstændigt bevis i Isabelle/HOL for korrektheden af protokollernes sammensætning.

Preface

This thesis was prepared at the Department of Applied Mathematics and Computer Science (DTU Compute) in fulfillment of the requirements for acquiring a PhD degree in Computer Science.

The research has been carried out under the supervision of Sebastian Mödersheim and Jørgen Villadsen in the period from October 2015 to September 2018.

This research was supported by the Sapere-Aude project “Composec: Secure Composition of Distributed Systems”, grant 4184-00334B of the Danish Council for Independent Research.

A substantial part of the work presented in this thesis is based on extensions to joint work with Sebastian Mödersheim and Achim Brucker, namely the following three published papers: *Formalizing and Proving a Typing Result for Security Protocols in Isabelle/HOL* [HM17], *A Typing Result for Stateful Protocols* [HM18], and *Stateful Protocol Composition* [HMB18]. Chapter 2 and Chapter 3 is mainly based on [HM17] whereas Chapter 4 is based on [HM18] and Chapter 5 is based on [HMB18].

Lyngby, 28-September-2018

Andreas V. Hess

Andreas Viktor Hess

Acknowledgements

I would like to thank my supervisor Sebastian Mödersheim and my co-supervisor Jørgen Villadsen. I am first of all grateful to have been provided the opportunity to work on such an interesting research project. Secondly, I am thankful for all their excellent help and continuous guidance which I benefited from immensely during my time as a PhD student.

I would like to thank Achim Brucker for being my host during my external stay at the University of Sheffield, and for insightful discussions and collaborations.

Thanks are due to the assessment committee for their many comments on a preliminary version of this thesis: Alberto Lafuente, Christoph Sprenger, and Joshua Guttman. Thanks also to those who have given me feedback on early versions of my published papers: Luca Viganò, Achim Brucker, and Anders Schlichtkrull.

I would also like to thank my colleagues at the Formal Methods section. Last but not least, I would like to thank my parents for their support.

Notation

Chapter 2

$\mathcal{V}$	The set of variables.	10
Σ	The set of function symbols.	10
Σ^n	The symbols of Σ of arity n.	10
$\mathcal{C}$	The constants (i.e., Σ^0).	10
$\mathcal{C}_{pub}$	The public constants.	10
$\mathcal{C}_{priv}$	The private constants.	10
s, t, u, m, e	Terms (also called messages) over signature Σ and variables $\mathcal{V}$.	10
M	Sets of terms.	10
f, g, h	Arbitrary function symbols, usually ranging over $\Sigma \setminus \mathcal{C}$.	10
x, y, z, v	Arbitrary variables from $\mathcal{V}$.	10
X, Y, Z	Sets of variables.	10
a, b, c, d, i, n	Arbitrary constants from $\mathcal{C}$ (a, b usually denotes honest agents, i usually denotes the intruder, c, d usually denotes arbitrary constants, and n usually denotes a nonce).	10
$f(t_1, \ldots, t_n)$	Composed terms.	10
$\mathit{Fun}\ f\ [t_1, \ldots, t_n]$	Composed terms (in Isabelle notation).	10
$\mathit{Fun}\ c\ []$	Constants (in Isabelle notation).	10
$\mathit{Var}\ x$	Variables (in Isabelle notation).	10
$\bar{x}, \bar{y}, \bar{c}, \bar{u}$	Finite sequences of terms.	10
$\mathit{fv}(t)$	The variables occurring in the term t.	11
$\mathit{subterms}(t)$	The set of subterms of t.	10
$\sqsubseteq$	The subterm relation.	10

$\sqsubset$	The proper subterm relation ($t \sqsubset s$ iff $t \neq s$ and $t \sqsubseteq s$).	10
$\theta, \sigma, \delta, \tau, \gamma$	Substitutions.	11
α	Variable renamings.	11
$dom(\theta)$	The substitution domain of the substitution θ.	12
$subst\text{-}domain\ \theta$	The substitution domain of the substitution θ (in Isabelle notation).	12
$ran(\theta)$	The substitution range of the substitution θ.	12
$subst\text{-}range\ \theta$	The substitution range of the substitution θ (in Isabelle notation).	12
$\mathcal{I}$	Interpretations.	13
$interpretation_{subst}\ \mathcal{I}$	Denotes that $\mathcal{I}$ is an interpretation (in Isabelle notation).	13
$[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$	The substitution with domain $\{x_1, \ldots, x_n\}$ and range $\{t_1, \ldots, t_n\}$.	12
$[]$	The identity substitution.	12
$M \vdash t$	Denotes that the intruder can derive t given the messages in M.	15

Chapter 3

$M \vdash_c t$	Denotes that the intruder can derive t given the messages in M without performing decomposition.	50
$\mathfrak{a}, \mathfrak{s}$	Constraint steps.	49
$\mathcal{A}$, $\mathcal{B}$	Symbolic constraints (also called intruder strands).	24
$(\mathcal{A}, \theta)$	Constraint states.	26
$trms_{st}\ \mathcal{A}$	Denotes the set of terms of the constraint $\mathcal{A}$	42
$wf_X(\mathcal{A})$	Denotes that $\mathcal{A}$ is a well-formed constraint w.r.t. the variables in X.	27
$wf_{st}\ X\ \mathcal{A}$	Denotes that $\mathcal{A}$ is a well-formed constraint w.r.t. the variables in X (in Isabelle notation).	27
$dual(\mathcal{A})$	The dual of the constraint $\mathcal{A}$.	49
$dual_{st}\ \mathcal{A}$	The dual of the constraint $\mathcal{A}$ (in Isabelle notation).	49
$fv(\mathcal{A})$	The free variables of the constraint $\mathcal{A}$.	27
$bvars(\mathcal{A})$	The bound variables of the constraint $\mathcal{A}$.	27
$\leadsto$	The lazy intruder constraint reduction relation.	34
$[\![M; \mathcal{A}]\!]\ \mathcal{I}$	Constraint semantics for stateless constraints. Denotes that $\mathcal{I}$ is a model of $\mathcal{A}$ given the initial knowledge M.	27

$[\![M;\mathcal{A}]\!]_c\ \mathcal{I}$	Constraint semantics for stateless constraints, using $\vdash_c$ instead of $\vdash$. Denotes that $\mathcal{I}$ is a model of $\mathcal{A}$ given the initial knowledge M.	29
$\mathcal{I} \models (\mathcal{A}, \theta)$	Constraint semantics. Denotes that $\mathcal{I}$ is a model of $\mathcal{A}$ (i.e., $[\![\emptyset;\mathcal{A}]\!]\ \mathcal{I}$) and that the substitution θ supports the interpretation $\mathcal{I}$.	26
$\mathcal{I} \models \mathcal{A}$	Constraint semantics. Equivalent to $\mathcal{I} \models (\mathcal{A}, [])$.	26
$\mathcal{I} \models_c \mathcal{A}$	Constraint semantics. Equivalent to $[\![\emptyset;\mathcal{A}]\!]_c\ \mathcal{I}$.	29
$\mathsf{send}(t)$, $\mathsf{receive}(t)$	Message transmission constraints.	24
Send t, *Receive* t	Message transmission constraints (in Isabelle notation).	24
$t \doteq t'$	Equality constraints on messages.	26
Equality $t\ t'$	Equality constraints on messages (in Isabelle notation).	26
$\forall \bar{x}.\ t \not\doteq t'$	Inequality constraints on messages.	26
Inequality $\bar{x}\ t\ t'$	Inequality constraints on messages (in Isabelle notation).	26
Γ	A function assigning types to terms.	39
$\mathfrak{T}_a$	The set of atomic types.	39
TAtom β	An atomic type β (in Isabelle notation).	39
TComp $f\ [\tau_1, \ldots, \tau_n]$	A composed type (in Isabelle notation).	39
$tfr_{stp}\ \mathfrak{a}$	Denotes that the constraint step $\mathfrak{a}$ is type-flaw resistant (in Isabelle notation).	42
$tfr_{st}\ \mathcal{A}$	Denotes that the constraint $\mathcal{A}$ is type-flaw resistant (in Isabelle notation).	42
$SMP(M)$	The sub-message patterns of the messages in the set M.	41

Chapter 4

$\mathcal{S}$	Strands, usually honest agent strands.	63
$\mathsf{insert}(t, s)$	Insertion of element t into database s.	63
Insert $t\ s$	Insertion of element t into database s (in Isabelle notation).	63
$\mathsf{delete}(t, s)$	Deletion of element t from database s.	63
Delete $t\ s$	Deletion of element t from database s (in Isabelle notation).	63
$t \mathrel{\dot{\in}} s$	Positive set-membership constraints.	63
InSet $t\ s$	Positive and negative set-membership constraints (in Isabelle notation).	63
$\forall \bar{x}.\ t \mathrel{\dot{\notin}} s$	Negative set-membership constraints.	63
NotInSet $\bar{x}\ t\ s$	Negative set-membership constraints (in Isabelle notation).	63

$\mathsf{assert}(t)$	Emits the event t.	63
$\mathsf{event}(t)$	Positive constraints on events.	63
$\forall \bar{x}.\ \neg\mathsf{event}(t)$	Negative constraints on events.	63
$[\![M, D; \mathcal{A}]\!]_s\ \mathcal{I}$	Constraint semantics for stateful constraints. Denotes that $\mathcal{I}$ is a model of $\mathcal{A}$ given the initial intruder knowledge M and the initial database mapping D.	71
$\mathcal{I} \models_s \mathcal{A}$	Constraint semantics for stateful constraints. Equivalent to $[\![\emptyset, \emptyset; \mathcal{A}]\!]_s\ \mathcal{I}$.	71
$\mathit{tfr}_{\mathit{sstp}}\ \mathfrak{a}$	Denotes that the stateful constraint step $\mathfrak{a}$ is type-flaw resistant (in Isabelle notation).	109
$\mathit{tfr}_{\mathit{sst}}\ \mathcal{A}$	Denotes that the stateful constraint $\mathcal{A}$ is type-flaw resistant (in Isabelle notation).	109
Chapter 5		
$\mathcal{P}$	Arbitrary protocols.	115
ℓ	Labels.	117
$\star$	The star label.	117
$\mathcal{A}\vert_\ell$	Projection of the constraint $\mathcal{A}$ to the steps labeled ℓ or $\star$.	117
$\mathcal{A}\vert_\star$	Projection of the constraint $\mathcal{A}$ to the $\star$-labeled steps.	117
$\mathcal{P}\vert_\ell$	Projection of the protocol $\mathcal{P}$ to the steps labeled ℓ or $\star$.	117
$\mathcal{P}\vert_\star$, $\mathcal{P}^\star$	Projection of the protocol $\mathcal{P}$ to the $\star$-labeled steps.	117
$\mathcal{L}$	The set of protocol-specific labels.	117
$\mathcal{P}_1 \parallel \mathcal{P}_2$	The parallel composition of protocols $\mathcal{P}_1$ and $\mathcal{P}_2$.	117
$\Vert_{i\in\mathcal{L}}\ \mathcal{P}_i$	The parallel composition of all protocols $\mathcal{P}_i$ indexed by $\mathcal{L}$.	117
$\mathit{GSMP}(M)$	The ground sub-message patterns of the messages in M.	123
$M \vdash_{\mathrm{hom}}^{\mathcal{A},\mathit{Sec}} t$	Denotes that the intruder can derive t given the messages in M using only homogeneous terms (w.r.t. the constraint $\mathcal{A}$ and the shared secrets Sec).	132

Contents

CHAPTER 1

Introduction

The typical use of communication networks like the Internet is to run a wide variety of security protocols in a composed fashion, such as TLS, IPSec, DNSSEC, and many others. For instance, protocols may be run side-by-side (i.e., *in parallel*), in *sequence*, or in a layered fashion where one protocol builds upon another (i.e., *vertically*).

While the security properties of many protocols have been analyzed in great detail, much less research has been devoted to their parallel composition. It is far from self-evident that the parallel composition of secure protocols is still secure, in fact one can systematically construct counter-examples. One such problem is if protocols have similar message structures of different meaning, so that an attacker may be able to abuse messages, or parts thereof, that he has learned in the context of one protocol, and use them in the context of another where the same structure has a different meaning. Thus, we have to exclude that the protocols in some sense "interfere" with each other. However, it is unreasonable to require that the developers of the different protocols have to work together and synchronize with each other. Similarly, we do not want to reason about the composition of several protocols as a whole, neither in manual nor automated verification because composed protocols are often too big to automatically verify in a reasonable amount of time. Instead, we want a set of sufficient conditions and a composition theorem of the form: every set of protocols that satisfies the conditions yields a secure composition, provided that each protocol is secure in isolation. The conditions should also be realistic so that many existing protocols like TLS (without modifications) actually satisfy them, and they should be simple, in the sense that checking them is a static task that does not involve considering actual protocol runs.

Existing parallel compositionality results like [HT94, GT00, ACG+08, Gut09, CD09, cCC10, CDKR13, ACD15, AMMV15] only apply to simple "stateless" protocols in which participants only have state local to a single session, like a session key. A more interesting and general class of protocols is one in which agents can additionally manipulate a global mutable state. In such protocols updating the global state during one session might influence other running sessions and we call such protocols *stateful*. There exist several tools and approaches for verifying stateful protocols [Möd10, ARR11, Gut12, AAA+12, MSCB13, MB16, KK16]. In a stateful protocol participants may maintain, e.g., a database independent of sessions. For instance, a keyserver may maintain a database of currently valid public keys. As part of the protocol, participants can insert entries into, or delete entries from, their databases. Moreover, the databases may be used in more than one protocol, for instance, the keyserver may offer several different protocols for establishing new keys, and revoking and updating existing keys.

The reason why stateful protocols are fundamentally different from stateless ones is that the global state in a stateful protocol does not necessarily grow monotonically during protocol execution: negative membership checks and deletion of elements from databases implies that what was true at some point in a protocol execution may not be true at a later point. We present in this thesis the first compositionality result that supports stateful protocols.

Besides establishing new compositionality results a great part of this work is concerned with the formalization of our technical results in Isabelle/HOL [NPW02]. Proof assistants like Isabelle allow us to formalize and check proofs with an almost "absolute" precision and reliability: once a theorem is proved, the chance of a mistake or hole in the proof is extremely low. This is very attractive for proofs of security protocols since security protocols are relatively small systems that are critical to our infrastructure and often have very subtle flaws that are easily overlooked. Paulson and Bella have proved numerous existing protocols secure in Isabelle and have established a general paradigm of modeling protocols [Pau98, Pau99, BMP06, Bel07].

Despite many automated proof tactics in Isabelle, conducting security proofs is still labor-intensive. There are many automated tools like ProVerif [Bla01] that can verify most of the protocols proven correct in Isabelle by Paulson and Bella within minutes. However, an implementation mistake in such a tool can easily lead to a "false negative": no attack is found even though there is one. To get full reliability, one could directly model such a method in Isabelle, using Isabelle as a kind of interpreter. A more efficient way is to have automated tools generate proofs that Isabelle can check as in the works of Brucker and Mödersheim [BM09] and Meier et al. [MCB13] (and by Goubault-Larrecq [Gou08] for Coq).

Many of the mentioned works [Pau98, Pau99, BMP06, Bel07, BM09] rely on a typed protocol model that excludes that the attacker can send any ill-typed messages and thereby rules out any type-flaw attacks. In general, such a restriction to a typed model makes many aspects of the analysis easier. It can significantly reduce verification time and in some approaches [BP05, AC08] protocol verification even becomes decidable in a typed model. Most notably, in the abstract interpretation method used by [Bla01, BM09], protocol security is still undecidable, but under the restriction to a typed model the question becomes decidable.

There are in fact several results that show the *relative soundness* of a typed model if the protocol satisfies certain reasonable sufficient conditions of a syntactic nature (i.e., can be checked without an exploration of the state space of the protocol): Heather et al. [HLS03], Chrétien et al. [CCD14],Mödersheim [Möd12], Arapinis and Duflot [AD14], and Almousa et al. [AMMV15]. Such *typing results* are of the form: if a protocol satisfies certain typing conditions, and has an attack, then it has a well-typed attack. That is, if a protocol that satisfies the sufficient conditions has an attack then it has a well-typed attack, in which the attacker only sends well-typed messages. In other words, if the protocol is secure in a typed model then it is secure in an untyped model. So if we can verify that the protocol has no attack in the typed model (with whatever method), then it also has no attack in the untyped model. Typing results are closely related to compositional reasoning. They have similar pre-requisites and proof techniques that fit quite well together, e.g., the mentioned works [CD09, AMMV15] rely on typing results to obtain parallel composition results. In fact, both typing results and compositionality results are part of the larger class of *relative soundness results* that all reduce a complex protocol verification problem to easier ones.

In a nutshell, when proving a typing result, one shows (for a given class of protocols) that from an ill-typed attack we can construct a similar well-typed attack, i.e., every ill-typed message that the intruder sends can be replaced with a well-typed one so that all the remaining steps of the attack can still be performed in a similar way. To avoid messy and round-about arguments, many existing typing results argue via a constraint-based representation of the intruder. In these constraints, all messages sent and received by the intruder may contain variables where the corresponding honest agent would accept any value. Every attack is then a solution of such a constraint. There is a sound, complete, and terminating reduction procedure for such intruder constraints that we call *the lazy intruder*. It thus suffices to show that for the considered class of protocols, this reduction procedure will never instantiate any variable with a term of a different type than the variable has. If the procedure leaves any variables uninstantiated (i.e., its concrete value does not matter for the attack to work) then the intruder may as well choose a well-typed value here. This therefore allows to conclude that if there is a solution (i.e., attack), then there is a well-typed one.

All the mentioned relative soundness results—both the typing results and the compositionality results—have so far been classical pen-and-paper proofs. They contain complex proof arguments that, despite not being formalized out to the last detail, span easily ten pages (including all relevant formal definitions and lemmas with their proofs). It is not unlikely that such a result can have mistakes, from simple holes in a proof to wrong statements. Relying on such results bears some similarity to relying on unverified tools: we may wrongly accept a protocol composition as secure that actually is not (in the considered model). "Checking" the proof of such a result may be as complex a task as verifying a verification tool. Another parallel to verification tools is: different works often have subtle differences in the protocol models and in the sufficient conditions that a casual user (who did not study the result in detail) may fail to notice. This bears the risk that in a hand-wavy fashion, one may accidentally apply a typing or compositionality result to protocols for which it does not hold (or, at least, has not been proven to hold).

All mentioned relative soundness results also only apply to stateless protocols.[1] Proving a relative soundness result for stateful protocols, however, is not trivial. If we consider as a global state a database to which entries can be added (without bound) and deleted, and where negative checks are allowed (i.e., that no entry of a particular form is present), then this is not possible with a straightforward extension of existing results. While one could encode the positive operations and checks as special messages, the negative ones essentially amount to checking that a particular operation or message did not occur, and this negation is at odds with the intruder constraints needed to perform the main proof argument of a relative soundness result.

Content and Contributions

The work presented in this thesis is based on three major publications, each of which has a dedicated chapter in this thesis:

- *Formalizing and Proving a Typing Result for Security Protocols in Is-*

[1] An exception being the typing result of [Möd12], but this paper contains significant mistakes and its result does not hold in this generality; we explain this in detail in Section 4.4.3.

abelle/HOL [HM17] by Hess and Mödersheim, published at the 30th IEEE Computer Security Foundations Symposium in 2017.

- *A Typing Result for Stateful Protocols* [HM18] by Hess and Mödersheim, published at the 31st IEEE Computer Security Foundations Symposium in 2018.
- *Stateful Protocol Composition* [HMB18] by Hess, Mödersheim, and Brucker, published at the 23rd European Symposium on Research in Computer Security in 2018.

Formalizing a Typing Result in Isabelle The first major contribution is the formalization in Isabelle of the typing result of [AMMV15] for stateless protocols. This work is presented in Chapter 3 and is based on [HM17].

As said before, to prove a typing result one needs to make an argument of the form: if a protocol has an attack then there exists an attack in which the intruder only makes well-typed choices. To make such an argument precise many works use a constraint-based approach where attacks on protocols are represented by symbolic constraints whose solutions constitute concrete attacks. The constraints are then reduced to simpler forms using a sound, complete, and terminating constraint-reduction system called the lazy intruder. Then it is sufficient to show that, given certain requirements on the constraints, the lazy intruder never makes an ill-typed choice during constraint-reduction and that the simple constraints have well-typed solutions. As part of formalizing the typing result, we formalize the lazy intruder in Isabelle and prove its soundness, completeness, and termination, and that it never makes an ill-typed choice during constraint reduction, for so-called type-flaw resistant constraints.

One aspect that makes the lazy intruder complicated, both in terms of an implementation in tools and in terms of proving completeness, lie in the intruder's ability to analyze (e.g., decrypt) messages that he knows. To make the formalization of the lazy intruder in Isabelle smoother we consider the following idea: we at first restrict the intruder so that he cannot analyze messages himself, but instead some external service performs the message analysis for the intruder. We then prove this equivalent to the standard intruder model. In order to facilitate easier reasoning in Isabelle we also introduce a new representation of constraints based on strands [THG99]. With the lazy intruder formalized, and its crucial properties proven, we can prove the typing result.

During the formalization effort we found several flaws in the work of [AMMV15]. There are mistakes in both the definitions of the lazy intruder and in the proofs that the simple constraints have well-typed solutions. In fact, their result [AMMV15] is not provable unless one makes an additional condition on protocols, and we provide such a condition. To make sure our fixed proofs are actually correct, we have verified them by formalizing them in Isabelle. Most importantly, this formalization serves as the foundation on which we later establish our typing and compositionality results for stateful protocols.

Establishing a Typing Result for Stateful Protocols Next we establish a new typing result that supports a large class of stateful protocols with a global state that consists of a collection of sets. This work is based on [HM18] and is presented in Chapter 4.

To have a simple and yet powerful formalism to work with, we first introduce a notion of *strands with set operations* to model honest agents. We use these new strands to define a notion of constraints with set operations that extends the constraints of Chapter 3. We call these constraints *stateful*. The set operations in stateful constraints then represent at which point the particular set operations occurred during an attack.

The typing result is established in a precise and declarative way that uses the typing result of Chapter 3 for stateless protocols as a basis. We show that we can reduce the satisfiability of stateful constraints to the satisfiability of constraints without set operations. This allows us to apply the result of Chapter 3 on the constraint-level. We finally lift the result to stateful protocols by defining a symbolic protocol transition system that builds up constraints during transitions and then applying the constraint-level results to constraint reachable in the transition system.

The formalism with set operations for honest agents and for intruder constraints is useful beyond our results to represent and work with stateful protocols. While our formalism is deliberately reduced to the essentials, we show how to connect other more complex formalisms for stateful protocols, namely using rewriting and process calculi, so that our results can be also applied accordingly in these languages.

We additionally point out several mistakes in a related typing result [Möd12]. A corrected version of [Möd12] that incorporates our fixes has been made available[2].

As in Chapter 3 we again formalize the typing result of Chapter 4 in Isabelle, extending the Isabelle formalization of Chapter 3.

[2] https://people.compute.dtu.dk/samo/taslanv3.pdf

Establishing a Compositionality Result for Stateful Protocols In Chapter 5 we establish the main result of this thesis, namely a parallel compositionality result for stateful protocols. This work is based on [HMB18].

We first extend the constraints of the previous chapters to incorporate labels that indicate which protocols the steps of the constraints came from. The proof of the main result is by a reduction to a problem on constraints: given a constraint that represents an attack on a composed protocol, we show that the projections of the constraint represent attacks on the individual protocols in isolation.

A key feature of our compositionality result is that we allow for databases to be shared. For instance, in the keyserver example, there could be several different protocols for registering, certifying, and revoking keys that all work on the same public-key database. Since such a shared database can potentially be exploited to trigger harmful interference, an important part of our result is a clear coordination of the ways in which each protocol is allowed to access the database. This coordination is based on assumptions and guarantees on the transactions that involve the database. Moreover, this also allows us to support protocols with the declassification of long-term secrets (e.g., that the private key to a revoked public key may be learned by the intruder without breaking the security goals).

Both the extension of the compositionality paradigm to stateful protocols and the support for shared databases are significant generalizations over previous results. Our result is so general that it even covers many forms of *sequential composition* as a special case, since one can, e.g., model that one protocol inserts keys into a database of fresh session keys, and another protocol "consumes" and uses them.

Building on the foundation developed in earlier chapters we formalize the core of the compositionality result, namely the result on the level of constraints, in Isabelle. The connection to the protocol level is also proved in Isabelle for stateless protocols, but still outstanding for stateful protocols. We have of course proved this result on paper.

More Benefits of Formalizing in Isabelle Since all our results are formalized in Isabelle we can use them directly in Isabelle-formalized security proofs like any other proved theorem in Isabelle. For instance, we may use any of the previously mentioned verification methods—manual or automatic—to prove the security of protocols in the typed model, and then use the Isabelle-formalized typing and compositionality theorems to infer that their parallel composition is secure in the untyped model as a theorem entirely proved within Isabelle. Formalizing the results in Isabelle also ensures that our theorems are only applicable if all the sufficient conditions are indeed satisfied (otherwise they will remain as open subgoals to prove). To demonstrate the practical feasibility of our result we show that the TLS 1.2 [DR08] handshake protocol satisfies the requirements of our typing result. Moreover, we define two stateful keyserver protocols and show that they satisfy the conditions of our compositionality result.

Since Isabelle-formalized proofs are often significantly more verbose than their pen-and-paper counterparts we have decided to present all the important proofs in a digestible form in the style of ordinary mathematical textbook proofs. The reason is that we have discovered and fixed flaws in existing proofs during formalization in Isabelle, asked ourselves how the proofs ideally should be presented, and extracted the essence of the Isabelle-formalized proofs. Thus the presentation is in a style that is also familiar to mathematicians that do not know Isabelle.

The full Isabelle formalization (with all proofs) is available at the following websites:

`http://orbit.dtu.dk/en/publications/typing-and-compositionality-for-stateful-security-protocols(72a413dc-d339-4931-8c59-5bb6cfc62bec).html`
`https://people.compute.dtu.dk/~samo/StateProtCompIsabelle.zip`

CHAPTER 2

Preliminaries

In this chapter we summarize some standard definitions along with a discussion of how we model them in Isabelle/HOL whenever there are some differences. Isabelle itself is a generic proof assistant suited for implementing different logical formalisms. The most widespread instance is Isabelle/HOL which is based on a higher-order logic and which is what we use in this thesis. While Isabelle and Isabelle/HOL are different we usually use both notions to refer to Isabelle/HOL.

We also give some new definitions and prove some small lemmas that are necessary for our technical results in subsequent chapters. This also gives us a chance to review a few features of Isabelle that are relevant for following the thesis in detail. In fact, we simplify the Isabelle notation in several places. For instance, the syntax of Isabelle/HOL resembles the syntax of a functional programming language and so function application is usually written without parentheses, i.e., $f\ t$. We use the notation $f\ t$ when we wish to make it clear that the expression is written in Isabelle-notation. In all other instances we use the notation $f(t)$.

2.1 Term Algebra

At the core of all definitions are the protocol messages that we model in a free first-order term algebra as is often done.

The standard notions like unification are already part of several Isabelle libraries namely the *Unification* example theory that ships with Isabelle and the IsaFoR library [TS09], and we point out only where our definitions augment them.[1]

Our definitions are parameterized over a set $\mathcal{V}$ of *variables* (typically denoted with letters x, y, z) and a set Σ of *function symbols* (typically denoted with letters f, g, h). We also assume a function $\mathit{arity} : \Sigma \to \mathbb{N}$ that assigns each function symbol its arity. We denote by $\mathcal{C}$ the subset of Σ of *constants*, i.e., the function symbols of arity 0 (typically denoted with letters a, b, c). The set Σ is furthermore partitioned into the *public symbols* Σ_{pub} (which the intruder has access to) and the *private symbols* Σ_{priv} (which the intruder cannot access). To express this partitioning in Isabelle we assume a predicate $\mathit{public} : \Sigma \to \mathit{bool}$. By Σ^n we then denote the subset of Σ containing all symbols of arity n. Similarly, Σ^n_{pub} (respectively Σ^n_{priv}) denotes the public (respectively private) symbols of arity n. The set of public constants and the set of private constants are then denoted by $\mathcal{C}_{pub}$ respectively $\mathcal{C}_{priv}$. We later define that the intruder has access to all constants in $\mathcal{C}_{pub}$ and that $\mathcal{C}_{pub}$ is infinite (modeling that the intruder has access to an unlimited supply of constants).

We now define the set of terms over Σ, $\mathcal{V}$ in Isabelle as an inductive datatype:

$$\textbf{datatype}\ (\Sigma, \mathcal{V})\ \mathit{term} = \mathit{Var}\ \mathcal{V} \mid \mathit{Fun}\ \Sigma\ ((\Sigma, \mathcal{V})\ \mathit{term}\ \mathit{list})$$

Here we have slightly changed the Isabelle notation that would have instead of Σ and $\mathcal{V}$ two type variables, which we for ease of notation do not distinguish from the universes of those values.[2] Moreover, the expression $\mathit{Fun}\ f\ [\mathit{Var}\ x,\ \mathit{Fun}\ c\ []]$ represents the term $f(x, c)$ in more conventional notation, and we will use that notation whenever possible. We also say that a term of the form $\mathit{Fun}\ f\ [t_1, \dots, t_n]$ is *composed* iff $n > 0$. Otherwise, if $n = 0$ then the term represents the constant f. Similarly, if a term is of the form $\mathit{Var}\ x$ then it represents the variable x and so we say the term is a variable. For finite sequences (or lists) of terms $[t_1, \dots, t_n]$ we use the notation $\bar{t}$. Hence $\bar{x}$ denotes a list of variables and $\bar{c}$ denotes a list of constants. Moreover, we denote by $\mathit{subterms}(t)$ the *set of subterms* of a term t defined as expected. We also extend the definition of *subterms* to sets of terms as expected. By $\sqsubseteq$ we denote the subterm relation defined as $t' \sqsubseteq t$ iff $t' \in \mathit{subterms}(t)$. The proper subterm relation is then denoted by $\sqsubset$ and is defined as $t' \sqsubset t$ iff $t' \neq t$ and $t' \sqsubseteq t$. Note that because the term definition introduces the data constructors *Var* and *Fun* as injective functions, we obtain

[1]All of the IsaFoR theory that we depend on has in a recent update become part of the library *First-Order Terms* [ST18] published at the Archive of Formal Proofs (the AFP, `https://www.isa-afp.org/`). Hence our formalization no longer depends on libraries external to the Isabelle distribution and the AFP, and we intend to publish our formalization at the AFP in the near future.

[2]In Isabelle, one has to rather write *UNIV* to refer to the set of values that belong to a particular type.

a free term algebra, i.e., $f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)$ iff $f = g$ and $t_i = s_i$ for $1 \leq i \leq n = m$.

Typically there is only a small finite set of non-constant function symbols representing the cryptographic primitives and the like; therefore this set is actually fixed in several other works on protocol security [Pau98, Bel07, MCB13] with one data constructor for each function symbol. Our parameterized version has the advantage that our proofs do not depend on the particular choice of operators used, so we do not have to update our proofs when adding a new operator. A slight disadvantage is that we cannot control as part of the data-type definition that a function symbol f with $\mathit{arity}\ f = n$ is only applied to a list of exactly n arguments. We can fix this by the following notion of *well-formed* terms:

Definition 2.1 A term t is well-formed, written $\mathit{wf}_{\mathit{trm}}\ t$, if it satisfies the following definition:

$$\mathit{wf}_{\mathit{trm}}\ t \equiv \forall f\ T.\ \mathit{Fun}\ f\ T \sqsubseteq t \longrightarrow \mathit{length}\ T = \mathit{arity}\ f$$

This definition is lifted to sets of terms M as expected:

$$\mathit{wf}_{\mathit{trms}}\ M \equiv \forall t \in M.\ \mathit{wf}_{\mathit{trm}}\ t$$

We deal only with well-formed terms and for simplicity omit writing it as a side-condition on all terms we use from this point onwards. We further define the function *fv* for the *free variables* of a term as standard, and say that a term t is *ground* if $\mathit{fv}(t) = \emptyset$. Both definitions are extended to sets as expected.

2.2 Substitutions and Interpretations

We use for substitutions and unifications the definitions and theorems of the IsaFoR library where substitutions (typically denoted with letters θ, δ, and σ) are functions from variables $\mathcal{V}$ to terms $(\Sigma, \mathcal{V})\ \mathit{term}$. They are homomorphically extended to functions on terms as expected, and we simply write $\theta(t)$ for applying substitution θ to term t (omitting the extensions function).

The *composition* $\theta \cdot \delta$ of substitutions θ and δ is defined as follows (this is following the convention of IsaFoR):

$$(\theta \cdot \delta)(t) \equiv \delta(\theta(t))$$

The *substitution domain* $\mathit{subst\text{-}domain} : (\Sigma, \mathcal{V})\ \mathit{subst} \to \mathcal{V}\ \mathit{set}$ of a substitution is the set of variables that are not mapped to themselves:

$$\mathit{subst\text{-}domain}\ \theta \ \equiv\ \{x \mid \theta(x) \neq \mathit{Var}\ x\}$$

For substitutions with finite domain we will use the common notation of value mappings, like $\theta = [x \mapsto s, y \mapsto t]$ for the substitution θ with substitution domain $\{x, y\}$ sending x to s and y to t. Thus, $[]$ denotes the identity substitution.

The *substitution range* $\mathit{subst\text{-}range} : (\Sigma, \mathcal{V})\ \mathit{subst} \to (\Sigma, \mathcal{V})\ \mathit{term\ set}$ is defined in terms of the domain by applying the substitution to every element of the domain (where $f \mathbin{`} S$ denotes the image of f under the set S):

$$\mathit{subst\text{-}range}\ \theta \ \equiv\ \theta \mathbin{`} \mathit{subst\text{-}domain}\ \theta$$

As a convention we write *dom* and *ran* instead of *subst-domain* and *subst-range* when we are not writing in Isabelle style.

To denote the substitution $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ with domain $\{x_1, \ldots, x_n\}$ and range $\{t_1, \ldots, t_n\}$, for some n, we sometimes use the notation $[\bar{x} \mapsto \bar{t}]$.

Every substitution that we will use has either a finite domain or its domain is the set of all variables $\mathcal{V}$ and it maps them to ground terms. The latter kind we call *interpretations*. We thus divert here from the common convention that substitution and interpretation are two disjoint notions, because they are conceptually so similar (e.g., they can be applied to all term-based data-structures) that having them separated would lead to two similar versions of many definitions and lemmas.

It is cumbersome to work with substitutions where some variable occurs both in the domain and the range like $[x \mapsto f(x)]$ as they are, for instance, not idempotent. Thus, we introduce a notion of well-formedness of substitutions that excludes any variable to occur both in domain and range and that requires a finite domain (because we will not use this notion on interpretations):

$$\mathit{wf}_{\mathit{subst}}\ \theta \equiv \mathit{subst\text{-}domain}\ \theta \cap \mathit{fv}(\mathit{subst\text{-}range}\ \theta) = \emptyset \wedge \mathit{finite}\ (\mathit{subst\text{-}domain}\ \theta)$$

Intuitively, a well-formed substitution represents a set of solutions (e.g., all solutions to a unification problem). We can prove the following lemma that is useful later when we compose substitutions in constraint reduction:

Lemma 2.2 *If θ_1 and θ_2 are well-formed substitutions such that*

1. *$\mathit{subst\text{-}domain}\ \theta_1 \cap \mathit{subst\text{-}domain}\ \theta_2 = \emptyset$ and*

2. subst-domain $\theta_1 \cap \mathit{fv}(\mathit{subst\text{-}range}\ \theta_2) = \emptyset$

then the composition $\theta_1 \cdot \theta_2$ *is also well-formed.*

An interpretation (typically denoted by the letter $\mathcal{I}$) is a substitution mapping every variable to a ground term:

$$\mathit{interpretation}_{\mathit{subst}}\ \mathcal{I} \equiv \mathit{subst\text{-}domain}\ \mathcal{I} = \mathcal{V} \wedge \mathit{ground}\ (\mathit{subst\text{-}range}\ \mathcal{I})$$

We define that a substitution θ *supports* an interpretation $\mathcal{I}$ as follows:

$$\theta\ \mathit{supports}\ \mathcal{I} \equiv \forall x.\ \mathcal{I}(\theta(x)) = \mathcal{I}(x)$$

A substitution θ is *idempotent* iff $\theta \cdot \theta = \theta$. We have proven that well-formed substitutions are idempotent.

Finally, we say that a substitution θ is *ground* iff θ maps every variable in its domain to a ground term, i.e., *ground* (*subst-range* θ). Hence interpretations are ground.

2.2.1 Injective and Fresh Substitutions

A function f is said to be bijective between sets A and B if it is injective on A and surjective between A and B. In Isabelle/HOL these concepts are defined as follows, where *inj-on* defines injectivity, *bij-betw* defines bijectivity, and f \`$A = B$ expresses surjectivity of f between A and B:

$$\begin{array}{rcl} \mathit{inj\text{-}on}\ f\ A & \equiv & \forall x \in A.\ \forall y \in A.\ f\ x = f\ y \longrightarrow x = y \\ \mathit{bij\text{-}betw}\ f\ A\ B & \equiv & \mathit{inj\text{-}on}\ f\ A \wedge f\ \grave{}\ A = B \end{array}$$

For substitutions θ we are only interested in injectivity on the substitution domain and bijectivity between the substitution domain and range. We therefore say that θ is *injective* if and only if *inj-on* θ (*subst-domain* θ). Similarly, θ is *bijective* if and only if *bij-betw* θ (*subst-domain* θ) (*subst-range* θ). Note that an injective substitution is also bijective and so we will use these two concepts interchangeably.

Later on we will need a stronger version of injectivity to express that a substitution maps different variables to different subterm-disjoint terms (i.e., terms that do not share any subterms). We call this property *subterm injectivity.*

Formally, we say that a function f is *subterm injective on* A if and only if *subterm-inj-on* f A where:

$$\textit{subterm-inj-on}\ f\ A \equiv \forall x \in A.\ \forall y \in A.\ (\exists t.\ t \sqsubseteq f\ x \wedge t \sqsubseteq f\ y) \longrightarrow x = y$$

That is, for f to be subterm injective it must be the case that $f\ x$ and $f\ y$ do not share subterms for all distinct $x, y \in A$.

Analogous to substitution injectivity we say that a substitution θ is subterm injective if and only if *subterm-inj-on* θ (*subst-domain* θ). Thus a subterm injective substitution is also an injective substitution.

Finally, we say that a substitution θ is *fresh* with respect to terms $t_1, \ldots, t_n$ iff θ is subterm injective and $\textit{subterms}(\textit{subst-range}\ \theta) \cap \textit{subterms}(\{t_1, \ldots, t_n\}) = \emptyset$.

2.2.2 Unification

A *most general unifier* (*mgu*) θ between two terms, t_1 and t_2, is defined as a substitution satisfying the following standard definition:

$$\textit{MGU}\ \theta\ s\ t \equiv \theta(s) = \theta(t) \wedge (\forall \delta.\ \delta(s) = \delta(t) \longrightarrow (\exists \gamma.\ \delta = \theta \cdot \gamma))$$

In other words, θ is a unifier which can be used to construct any other unifier δ of s and t using composition with a third substitution γ. *Well-formed mgu*s are furthermore restricted to the variables of the terms being unified. That is:

$$\begin{aligned} \textit{wf}_{\textit{MGU}}\ \theta\ s\ t \equiv\ & \textit{wf}_{\textit{subst}}\ \theta \wedge \textit{MGU}\ \theta\ s\ t\ \wedge \\ & \textit{subst-domain}\ \theta \cup \textit{fv}(\textit{subst-range}\ \theta) \subseteq \textit{fv}(s) \cup \textit{fv}(t) \end{aligned}$$

The IsaFoR library provides the function

$$\textit{mgu} :: (\Sigma, \mathcal{V})\ \textit{term} \Rightarrow (\Sigma, \mathcal{V})\ \textit{term} \Rightarrow (\Sigma, \mathcal{V})\ \textit{subst option}$$

that computes the most general unifier of two terms if one exists. We proved that this unifier is always well-formed:

Lemma 2.3 *If* $\textit{mgu}\ s\ t = \textit{Some}\ \delta$ *then* $\textit{wf}_{\textit{MGU}}\ \delta\ s\ t$.

(The data constructor *Some* is from the *option* datatype and the expression *Some* δ indicates here that a most-general unifier δ of s and t has been found.)

In addition to being well-formed we have also proved that the mgus computed by *mgu* satisfy the following useful property:

Lemma 2.4 *If mgu s t = Some δ then*

$$\mathit{subterms}(\mathit{ran}(\delta)) \setminus \mathcal{V} \subseteq \delta((\mathit{subterms}(s) \cup \mathit{subterms}(t)) \setminus \mathcal{V})$$

The proof is by induction over the (standard) mgu algorithm, showing that the property to prove is an invariant of the algorithm.

Some important consequences of Lemma 2.4 are that constants appearing in *subst-range* δ must occur in s and t, and that for any composed term of the form $f(y_1, \ldots, y_n)$ appearing in *subst-range* δ we can find a term of the form $f(x_1, \ldots, x_n)$ occurring in s or t where $\delta(x_i) = y_i$ for all $i \in \{1, \ldots, n\}$.

2.3 Dolev-Yao Style Intruder

We now define a standard symbolic Dolev-Yao style intruder deduction relation $M \vdash t$ to formalize that the intruder can derive the term t from the set of terms M, the *intruder knowledge*. Our model is similar to standard Dolev-Yao models but we parameterize ours over arbitrary signatures Σ instead of fixing a particular set of cryptographic primitives. The relation $\vdash$ is then defined inductively as the least relation closed under the following rules:

Definition 2.5 (The Intruder Model)

$$\frac{}{M \vdash t}\ \begin{array}{c}(\mathit{Axiom}),\\ t \in M\end{array} \qquad \frac{M \vdash t_1 \quad \cdots \quad M \vdash t_n}{M \vdash f(t_1, \ldots, t_n)}\ \begin{array}{c}(\mathit{Compose}),\\ f \in \Sigma^n,\\ \mathit{public}\ f\end{array}$$

$$\frac{M \vdash t \quad M \vdash k_1 \quad \cdots \quad M \vdash k_n}{M \vdash t_i}\ \begin{array}{c}(\mathit{Decompose}),\\ \mathsf{Ana}\ t = (K, T),\ t_i \in T,\\ K = \{k_1, \ldots, k_n\}\end{array}$$

The first rule expresses that the intruder can derive everything in his knowledge. The second rule allows the intruder to *compose* messages by applying *public* function symbols to messages he can already derive. (Note that f might have arity $n = 0$, in which case it is a constant.) To that end, we use the function *public* defined earlier that yields true for all public constants and function symbols. The third rule allows the intruder to *decompose* (i.e., *analyze*) messages. To avoid that we have to write a special decryption rule for each operator to consider, we assume a function Ana as an *analysis interface*. Intuitively, Ana $t = (K, T)$

means that the intruder can analyze the term t provided that he knows the "keys" in K and then obtain as the result of analysis the terms of T. For instance, to decrypt the message $\mathsf{crypt}(k, m)$ and obtain m we can require that the inverse key $\mathsf{inv}(k)$ must be provided, and we write $\mathsf{Ana}\ \mathsf{crypt}(k, m) = (\{\mathsf{inv}(k)\}, \{m\})$ to formally express this. Note that this would be similar to introducing a destructor dcrypt and an algebraic equation $\mathsf{dcrypt}(\mathsf{inv}(k), \mathsf{crypt}(k, m)) \approx m$ if we were not using the free algebra. More generally, if $\mathsf{Ana}\ t = (K, T)$ then the analysis of the term t results in the terms in T provided that all "keys" in K can be derived. The advantage of this interface function is that in order to add a new operator to the model, one simply has to specify the Ana function for it, but none of the following definitions or theorems require an update.

In Isabelle we use the notion of a *locale* to parameterize our theory over arbitrary signatures and analysis theories that satisfy our requirements. A locale allows us to assume the existence of functions and types in all proofs and definitions inside the scope of the locale. For instance, we parameterize our theory over Ana functions satisfying certain requirements. Those requirements are explained in detail in Chapter 3.

The locale definition for our intruder model, *intruder-model*, fixes the functions *arity*, *public*, and Ana, and two types Σ and $\mathcal{V}$, that must satisfy seven requirements (the last of which is our requirement that there are infinitely many public constants):

$$
\begin{array}{l}
\textbf{locale}\ \textit{intruder-model} = \\
\quad \textbf{fixes}\ \textit{arity} :: \Sigma \Rightarrow \textit{nat} \\
\quad\ \textbf{and}\ \textit{public} :: \Sigma \Rightarrow \textit{bool} \\
\quad\ \textbf{and}\ \mathsf{Ana} :: (\Sigma, \mathcal{V})\ \textit{term} \Rightarrow ((\Sigma, \mathcal{V})\ \textit{term set} \times (\Sigma, \mathcal{V})\ \textit{term set}) \\
\quad \textbf{assumes}\ \bigwedge t\ K\ M.\ \mathsf{Ana}\ t = (K, M) \Longrightarrow \textit{fv}(K) \subseteq \textit{fv}(t) \\
\quad\ \textbf{and}\ \bigwedge t\ K\ M.\ \mathsf{Ana}\ t = (K, M) \Longrightarrow \textit{finite}\ K \\
\quad\ \textbf{and}\ \bigwedge t\ k\ K\ M\ f\ S.\ \mathsf{Ana}\ t = (K, M) \\
\qquad\qquad \Longrightarrow (\bigwedge g\ T.\ \textit{Fun}\ g\ T \sqsubseteq t \Longrightarrow \textit{length}\ T = \textit{arity}\ g) \\
\qquad\qquad \Longrightarrow k \in K \Longrightarrow \textit{Fun}\ f\ S \sqsubseteq k \Longrightarrow \textit{length}\ S = \textit{arity}\ f \\
\quad\ \textbf{and}\ \bigwedge x.\ \mathsf{Ana}\ (\textit{Var}\ x) = (\emptyset, \emptyset) \\
\quad\ \textbf{and}\ \bigwedge f\ T\ K\ M.\ \mathsf{Ana}\ (\textit{Fun}\ f\ T) = (K, M) \Longrightarrow M \subseteq T \\
\quad\ \textbf{and}\ \bigwedge t\ \delta\ K\ M.\ \mathsf{Ana}\ t = (K, M) \\
\qquad\qquad \Longrightarrow K \neq \emptyset \vee M \neq \emptyset \Longrightarrow \mathsf{Ana}\ (\delta(t)) = (\delta(K), \delta(M)) \\
\quad\ \textbf{and}\ \textit{infinite}\ \{c \mid \textit{public}\ c \wedge \textit{arity}\ c = 0\}
\end{array}
$$

Here the Isabelle notation of the form $\bigwedge t_1 \cdots t_n.\ P_1 \Longrightarrow \ldots \Longrightarrow P_m \Longrightarrow Q$ denotes statements of the form $\forall t_1, \ldots, t_n.\ P_1 \wedge \cdots \wedge P_m \longrightarrow Q$ in more conventional notation.

The advantage of using a locale is that we do not need to fix a particular signature and analysis theory for our term model. Instead, to use our result, one can *interpret* the locale in Isabelle by providing functions that are proven to satisfy the type restrictions and requirements of the locale. For instance, locale interpretation in Isabelle could look as follows:

$$\begin{array}{l}\textbf{interpretation}\ \textit{intruder-model}\ \textit{arity}_c\ \textit{public}_c\ \mathsf{Ana}_c\\ \textbf{proof}\\ \quad \langle \textit{proof} \rangle\\ \textbf{end}\end{array}$$

where $\textit{arity}_c$, $\textit{public}_c$, and Ana_c are some user-defined functions and $\langle \textit{proof} \rangle$ is a proof that the provided functions satisfy the locale requirements. Everything in the scope of the interpreted locale is then available to use, e.g., to apply our results to protocols that uses particular signatures Σ and variables $\mathcal{V}$.

Note that the requirements on Ana regulate that the analysis interface is meaningful on symbolic terms containing variables in addition to ground terms. Note in particular that the third assumption defines that all "keys" K must be well-formed if the term t to be decomposed is well-formed. The terms in M are also well-formed since we require that the terms in M are immediate subterms of the terms in t, namely the fifth requirement:[3]

$$\bigwedge f\ T\ K\ M.\ \mathsf{Ana}\ (\mathit{Fun}\ f\ T) = (K, M) \Longrightarrow M \subseteq T$$

Example 2.1 *There are at least two ways to model asymmetric encryption, both of which are used in formal methods and protocol verification. We gave an example earlier where we used a function* inv *from public to private keys. Another possibility is to instead have a function* pub *from private to public keys. Both of these approaches have subtle advantages and disadvantages as we will see in later chapters.*

As a concrete Ana *theory using* pub *instead of* inv *consider the following set of non-constant operators (with their arities): asymmetric encryption* $\mathsf{crypt}/2$*, symmetric encryption* $\mathsf{scrypt}/2$*, signatures* $\mathsf{sign}/2$*, a function* $\mathsf{pub}/1$ *that yields the public key for a given private key, hash function* $\mathsf{hash}/1$*, a key derivation function* $\mathsf{kdf}/2$ *and message structuring formats* f_i/i *(*$i \in \mathbb{N}$*), together with the following* Ana *function:*

$$\begin{array}{lcl}\mathsf{Ana}\ \mathsf{scrypt}(k, m) & = & (\{k\}, \{m\})\\ \mathsf{Ana}\ \mathsf{crypt}(\mathsf{pub}(k), m) & = & (\{k\}, \{m\})\\ \mathsf{Ana}\ \mathsf{sign}(k, m) & = & (\emptyset, \{m\})\\ \mathsf{Ana}\ \mathsf{f}_i(t_1, \dots, t_i) & = & (\emptyset, \{t_1, \dots, t_i\})\\ \multicolumn{3}{c}{\textit{and in all other cases: } \mathsf{Ana}\ t = (\emptyset, \emptyset)}\end{array}$$

[3]Note that we have here simplified the Isabelle notation slightly. In Isabelle one would need to write $M \subseteq \mathit{set}\ T$ instead of the expression $M \subseteq T$ since M is a set while T is a list. The function *set* is a function provided by Isabelle that converts a list to the set containing exactly the elements of the input list, i.e., $\mathit{set}\ [a_1, \dots, a_n] = \{a_1, \dots, a_n\}$. We leave all applications of the *set* function implicit.

This describes the decryption of symmetric and asymmetric encryptions as expected; the contents of signatures we assume can be obtained without knowing the signing key (i.e., the signature primitive includes the signed text in clear). The functions pub, hash, *and* kdf *are one-way in the sense that they do not yield any information in analysis. The non-cryptographic message structuring formats* f_i *exist only to structure clear-text messages. Like [AMMV15] we use these formats instead of the classical "concatenate" operator: this allows for modeling abstractly the actual mechanisms of the implementation to structure messages unambiguously, such as tags, length information, or character encodings; compare for instance the TLS example in Section 3.2.2. The formats are* transparent, *i.e., the analysis function yields all direct subterms without requiring any key. For signatures, we similarly allow the intruder to obtain the signed message without any key. This models a signature scheme where the message being signed is given along with a signature on a hash of that message; thus one does not need any key to* obtain *the message, but only to* verify *the signature. We will come back to signature verification later.*

Let $M = \{\mathsf{scrypt}(\mathsf{kdf}(\mathsf{n1}, \mathsf{n2}), \mathsf{secret}), \mathsf{n1}, \mathsf{n2}\}$ *then for instance* $M \vdash \mathsf{secret}$ *since the intruder can first compose the key* $\mathsf{kdf}(\mathsf{n1}, \mathsf{n2})$ *and then decrypt the encrypted message.* □

2.3.1 Free Algebra

Recall that our definition of terms yields a free term algebra, i.e., two terms are equal only if they are syntactically equal. This prevents many interesting properties of operators like the property $g^{xy} = g^{yx}$ that is needed for all Diffie-Hellman-based protocols. Modeling algebraic properties in Isabelle is not trivial: one has to work with a quotient algebra (every term represents the *set* of terms that are algebraically equal) and thus everything becomes more complex, see for instance [SP13, HK13]. Therefore most protocol verification in Isabelle uses the free algebra. Most typing results also are limited to the free algebra (an exception being [Möd11]).

One may wonder, however, how this free algebra model compares to other protocol models like the Applied-π calculus model of ProVerif [Bla01] that supports algebraic properties to some extent. As an example, to describe signature verification, one may specify a *destructor* function verify that takes as arguments a signed message and a public key and yields true if the signature is correct w.r.t. that public key. This is expressed by the rewrite rule:

$$\mathsf{verify}(\mathsf{sign}(privkey, msg), \mathsf{pub}(privkey)) \to true$$

(Similar rewrite rules we have for decryption functions and the like.) This is in fact an algebraic property and it cannot be directly expressed in a free algebra term model. Note that internally, ProVerif works with Horn clauses in the free algebra as well. Therefore a transformation step is taken when translating a given Applied-π-calculus specification into Horn clauses. In the example of signature verification or similar theories of constructors and destructors, this would amount to pattern matching. Consider for instance an honest agent who receives an arbitrary message x and checks that applying verify with a particular key $\mathsf{pub}(\mathit{privkey})$ yields *true*; ProVerif's transformation would yield an agent who now receives only messages of *pattern* $\mathsf{sign}(\mathit{privkey}, y)$ where y is a variable to which the content of the signature is bound. This is precisely how free algebra approaches handle constructors—having no explicit destructors anymore.

Furthermore, ProVerif also allows for equations and it similarly applies a completion procedure to the Horn clauses to take into account all algebraic variants of a term. Note this feature may easily lead to non-termination and one must carefully craft the algebraic properties for this, see for instance [KT09]. In principle one can apply the same transformation also to strands in order to handle some algebraic properties, but we leave this as future work.

Chapter 3

Formalizing a Typing Result in Isabelle/HOL

In this chapter we formalize the typing result of Almousa et al. [AMMV15] in Isabelle. Typing results are of the form: if a protocol $\mathcal{P}$ satisfies certain typing conditions, and if $\mathcal{P}$ is secure in a well-typed model, then $\mathcal{P}$ is secure also in an untyped model. In order to prove such a typing result, one needs to make arguments of the form “in every step of an attack where the intruder sends something ill-typed, he may send something well-typed instead and the attack would work similarly.” To make such arguments in a clear and precise way—avoiding hand-wavy and roundabout proofs—existing typing results [CD09, Möd12, AD14, AMMV15] use a popular verification technique that uses symbolic intruder constraints and that we simply refer to as the *lazy intruder*. This idea is originally used to cope with the infinity induced by the Dolev-Yao model in automated verification [MS01, RT03, BMV05]: the intruder is lazy in the sense that he chooses parts of messages that he sends only in a demand-driven way, i.e., if a particular form is necessary for a particular attack. One can use this technique in a different way for the typing results by showing (for protocols that satisfy some requirements) that the lazy intruder never makes ill-typed choices, and all type-flaw attacks are ill-typed choices of message parts that the lazy intruder did not instantiate. This allows one to conclude that, if there is a solution to the constraints, then there is a well-typed one. This is at the core of all typing results and we thus formalize the lazy intruder in Isabelle, including the proof that the reduction procedure for constraints is sound, complete, and terminating, because the typing result relies on this.

During the formalization effort we discovered a number of errors in [AMMV15]. We have fixed these problems by imposing some additional conditions on the class of protocols. We argue that these conditions are reasonable restrictions and still more liberal than those of other typing results. This is discussed in detail in Section 3.3. To illustrate the feasibility of our requirements, as a real-world case study, we prove in Isabelle that the Transport Layer Security (TLS 1.2 [DR08]) protocol satisfies the requirement of the typing result, namely *type-flaw resistance*.

In order to facilitate easier reasoning in Isabelle, we have also made several simplifications to the lazy intruder. We also prove in Isabelle that these simplifications are without loss of generality, i.e., we prove the equivalence to a standard transition system with a full intruder. We use the Isabelle formalization of the lazy intruder only as a means to prove the main typing result, however it can also be employed directly to conduct lazy intruder-based proofs in Isabelle. More generally, we believe that all tools that use the lazy intruder technique can benefit from the simplification we made to the technique here.

Since this chapter consists of many definitions and theorems, including several variants of theorems, we here give a shortlist of definitions and the main typing result, i.e., everything that one needs to consider in order to apply our result:

- Given a protocol described as a countable set of closed strands, we define a state transition system with constraints (Definition 3.21).
- We define the semantics of constraints using a standard Dolev-Yao intruder deduction relation (in the free algebra) (Section 3.1.2 and Definition 2.5).
- We define the notion of *type-flaw resistance* for protocols (Definition 3.17).
- We define a requirement on the use of operators in the protocol, called *analysis-invariance*, needed to fix a mistake in [AMMV15] (Definition 3.24).
- The main result is that for any reachable state of a type-flaw resistant, analysis-invariant protocol, there is a solution for the constraints if and only if there is a well-typed solution (Theorem 3.27).

3.1 Modeling the Lazy Intruder in Isabelle/HOL

A naïve approach to model checking security protocols would be to devise a transition system that contains transitions for honest agents and term composition/decomposition steps for the intruder. Since term composition is infinite, one would not only bound the number of honest agents and the number of protocol runs they can participate in, but also the complexity of messages that the intruder can compose. (In fact, the typing result shows that for a large class of protocols this bounding of the intruder is without loss of attacks.) But even under tight bounds, the search space is infeasibly large. Therefore a technique has emerged that replaces this "eager" exploration of what the intruder can do by a symbolic approach with constraints and a demand-driven, "lazy" evaluation of these constraints [MS01, RT03, BMV05]. We thus like to call this technique the *lazy intruder*. While a successful method in the analysis and verification (for a bounded number of sessions) of security protocols, it has also been used as a proof argument for typing and compositionality results [CD09, AD14, AMMV15] like the one we formalize here in Isabelle. Note that in this way of using the lazy intruder, there is no bound on the number of sessions.

In most works the lazy intruder constraints are of the form $M \vdash t$ where now M and t can contain variables. Intuitively, M is the knowledge that the intruder had at a point where he sent a message of the form t to some honest agent. Here the term t may contain variables, so that t is a *pattern* of what messages the agent would accept and the variables are the places where the agent does not expect a particular value. This is where the lazy intruder is lazy: we do not right away try to determine a value for each variable. Therefore, the next messages this honest agent sends may contain variables from t and this is how variables can end up in the intruder knowledge M' in a successor state.

For a feasible procedure for checking the satisfiability of constraints, one needs to require a well-formedness condition on constraints: they can be ordered as $M_1 \vdash t_1, \ldots, M_n \vdash t_n$ (the order in which the constraints occurred) where

1. $M_i \subseteq M_{i+1}$ (for $1 \leq i < n$): the intruder knowledge grows monotonically; and
2. $fv(M_i) \subseteq \bigcup_{k=1}^{i-1} fv(t_k)$: all variables originate from a term sent by the intruder.

A large part of this chapter is to formalize in Isabelle/HOL these lazy intruder constraints, a reduction procedure for the constraints, and to prove the soundness, completeness, and termination of this procedure. Completeness and termination are quite difficult even as standard pen-and-paper proofs. We therefore had to first seek for any possibilities to make the task and the formalization as easy and light-weight as possible. The main simplifications are a different representation and an "out-sourcing" of decomposition steps, as we explain next.

The formalization we present here is a so-called *deep embedding*, i.e., we formalize constraints as objects in Isabelle that we can reason about. This is in contrast to a *shallow embedding* where we simply consider them as HOL formulae that use the $\vdash$ predicate. A shallow embedding would have advantages (both in terms of simplicity and performance) if one would like to directly perform constraint reasoning in Isabelle. A deep embedding is however necessary for our purpose, since we want to reason about *a procedure* that manipulates constraints, and in particular prove that this procedure is complete and terminates (the soundness proof could also be expressed in a shallow embedding).

3.1.1 The Lazy Intruder on the Beach

The first idea for keeping matters simple is to change the representation of the constraints by using *strands*.[1] A strand is a sequence of send and receive operations and strand spaces are a nice formalism to reason about protocol executions [THG99]. We therefore define an *intruder strand* (or *intruder constraint*) as a list of received and sent messages:

$$\begin{array}{l}\textbf{datatype}\ (\Sigma,\mathcal{V})\ \textit{strand-step} = \\ \quad \textit{Send}\ ((\Sigma,\mathcal{V})\ \textit{term}) \\ \mid\ \ \textit{Receive}\ ((\Sigma,\mathcal{V})\ \textit{term})\end{array}$$

$$\textbf{type-synonym}\ (\Sigma,\mathcal{V})\ \textit{strand} = (\Sigma,\mathcal{V})\ \textit{strand-step list}$$

Thus, the intruder knowledge at each point in the strand are the messages that the intruder has received up to this point, and each sent message must be something he can construct from the knowledge at that point.

Example 3.1 *In this chapter we use the analysis theory defined in Example 2.1. Consider now the following constraints in traditional representation:*

$$\begin{array}{ll}\{\mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}),\mathsf{secret}),\mathsf{k_i}\} & \vdash \mathsf{crypt}(\mathsf{pub}(x),y) \\ \{\mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}),\mathsf{secret}),\mathsf{k_i},y\} & \vdash \mathsf{secret}\end{array}$$

In the strand representation we would write this as:

$$\begin{array}{l}\textit{Receive}\ \mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}),\mathsf{secret}).\textit{Receive}\ \mathsf{k_i}. \\ \textit{Send}\ \mathsf{crypt}(\mathsf{pub}(x),y).\textit{Receive}\ y.\textit{Send}\ \mathsf{secret}.0\end{array}$$

[1]Note that the word *strand* in Danish and German means *beach*.

Instead of $[st_1, \dots, st_n]$ *we rather write* $st_1. \; \dots \; .st_n.0$ *like in process calculi.* □

The advantage of our representation is that we have "built-in" the first condition of the well-formedness: that the intruder knowledge monotonically grows. The second condition—that all variables originate in terms sent by the intruder—is now easy to formulate as $wf_{st}\ \emptyset$ where wf_{st} is defined as follows:

$$\begin{array}{rcl} wf_{st}\ X\ 0 & \text{iff} & \mathit{True} \\ wf_{st}\ X\ (\mathit{Receive}\ t.\mathcal{A}) & \text{iff} & \mathit{fv}(t) \subseteq X \text{ and } wf_{st}\ X\ \mathcal{A} \\ wf_{st}\ X\ (\mathit{Send}\ t.\mathcal{A}) & \text{iff} & wf_{st}\ (X \cup \mathit{fv}(t))\ \mathcal{A} \end{array}$$

Here the parameter X of $wf_{st}\ X\ \mathcal{A}$ is meant to denote the free variables of all sent messages that have occurred in a prefix of the parameter $\mathcal{A}$. The *intruder knowledge* of an intruder strand $\mathcal{A}$, written $ik_{st}\ \mathcal{A}$, is the set of received messages. That is, $t \in ik_{st}\ \mathcal{A}$ iff *Receive* t occurs in $\mathcal{A}$.

3.1.2 Constraint Semantics

We define the semantics of intruder strands based on the Dolev-Yao deduction relation $\vdash$. Recall that an interpretation $\mathcal{I}$ maps all variables to ground terms. We write $[\![M; \mathcal{A}]\!]\ \mathcal{I}$ to denote that $\mathcal{I}$ is a solution of an intruder strand $\mathcal{A}$ where M is an (initially empty) set of messages available to the intruder at the start, and define this relation as follows:

$$\begin{array}{rcl} [\![M; 0]\!]\ \mathcal{I} & \text{iff} & \mathit{True} \\ [\![M; \mathit{Send}\ t.\mathcal{A}]\!]\ \mathcal{I} & \text{iff} & M \vdash \mathcal{I}(t) \text{ and } [\![M; \mathcal{A}]\!]\ \mathcal{I} \\ [\![M; \mathit{Receive}\ t.\mathcal{A}]\!]\ \mathcal{I} & \text{iff} & [\![(\{\mathcal{I}(t)\} \cup M); \mathcal{A}]\!]\ \mathcal{I} \end{array}$$

Thus, every message he receives is simply added to the parameter M that collects the intruder knowledge in this inductive definition. For every message t that the intruder sends, we require that he can derive it from knowledge M (under the interpretation $\mathcal{I}$).

Example 3.2 *Consider the intruder strand from the previous example. Any* $\mathcal{I}$ *with* $\mathcal{I}(x) = \mathsf{k_a}$ *and* $\mathcal{I}(y) = \mathsf{secret}$ *is a solution of the constraint. Consider only the prefix up to (and including) the first Send step; then also* $\mathcal{I}(x) = \mathcal{I}(y) = \mathsf{k_i}$ *is a solution, and so is any interpretation that maps* x *and* y *to terms that the intruder can generate from his knowledge at that point.* □

During constraint reduction below we will consider pairs $(\mathcal{A}, \theta)$ of an intruder strand $\mathcal{A}$ and a well-formed substitution θ that represents the (partial) solution obtained so far (like the solution for x and y in the example above). At the beginning of the reduction, θ is simply the identity. Whenever θ is augmented during reduction, we apply it also to $\mathcal{A}$, i.e., $\mathcal{A}$ contains no variables in the domain of θ (that are already "solved").[2] Formally:

Definition 3.1 A *constraint state* is a pair $(\mathcal{A}, \theta)$, where $\mathcal{A}$ is an intruder strand and θ is a substitution. A constraint state $(\mathcal{A}, \theta)$ is furthermore *well-formed* if

1. $\mathcal{A}$ is a well-formed intruder strand,
2. θ is a well-formed substitution, and
3. the domain of θ and the variables of $\mathcal{A}$ are disjoint.

The interpretation $\mathcal{I}$ is said to be a *model* of $(\mathcal{A}, \theta)$ (with initial intruder knowledge M_0) written $M_0, \mathcal{I} \models (\mathcal{A}, \theta)$, iff θ supports $\mathcal{I}$ and $[\![M_0; \mathcal{A}]\!]\ \mathcal{I}$ holds. For the default $M_0 = \emptyset$ we simply write $\mathcal{I} \models (\mathcal{A}, \theta)$, and $\mathcal{I} \models \mathcal{A}$ if additionally $\theta = []$.

3.1.3 Extending Constraints with Message Checks

Intruder constraints have been extended in [AMMV15] with positive message checks $t \doteq t'$ (called *equalities*) and universally quantified negative message checks $\forall \bar{x}.\ t \not\doteq t'$ (called *inequalities*).

To support such constraints in our Isabelle-formalization we augment the datatype *strand-step* with two new constructors called *Equality* and *Inequality*. The datatype is therefore defined as follows:

$$\begin{aligned}
&\textbf{datatype}\ (\Sigma, \mathcal{V})\ \textit{strand-step} = \\
&\quad \textit{Send}\ ((\Sigma, \mathcal{V})\ \textit{term}) \\
&|\quad \textit{Receive}\ ((\Sigma, \mathcal{V})\ \textit{term}) \\
&|\quad \textit{Equality}\ ((\Sigma, \mathcal{V})\ \textit{term})\ ((\Sigma, \mathcal{V})\ \textit{term}) \\
&|\quad \textit{Inequality}\ (\mathcal{V}\ \textit{list})\ ((\Sigma, \mathcal{V})\ \textit{term})\ ((\Sigma, \mathcal{V})\ \textit{term})
\end{aligned}$$

Here a step of the form *Equality* $t\ t'$ represents the constraint that t and t' must be equal whereas *Inequality* $\bar{x}\ t\ t'$ represents that t and t' must be unequal under every instantiation of the variables $\bar{x}$.

[2] In Isabelle, we have to define explicitly an extension of substitution to functions on constraints (defined homomorphically as expected) but we leave this implicit in the notation in this thesis, for simplicity.

Now that we have introduced quantification over variables in our constraint language we need to distinguish between the variables bound by a quantifier and the remaining variables. For a given constraint $\mathcal{A}$ we therefore define its *bound variables* to be those which occur in the variable list $\bar{x}$ of some inequality $\mathit{Inequality}\ \bar{x}\ t\ t'$ whereas the remaining variables are its *free variables.*

With the syntax extended we must also extend the semantics. Given an interpretation $\mathcal{I}$ an $\mathit{Equality}\ t\ t'$ constraint is satisfied if t and t' are equal under $\mathcal{I}$ while an $\mathit{Inequality}\ \bar{x}\ t\ t'$ constraint is satisfied if no grounding of the variables in $\bar{x}$ unifies t and t' under $\mathcal{I}$.

Definition 3.2 The constraint semantics $[\![\cdot;\cdot]\!]$ is defined as follows:

$$\begin{array}{rll}
[\![M;0]\!]\ \mathcal{I} & \text{iff} & \mathit{True}\\
[\![M;\mathit{Send}\ t.\mathcal{A}]\!]\ \mathcal{I} & \text{iff} & M \vdash \mathcal{I}(t) \text{ and } [\![M;\mathcal{A}]\!]\ \mathcal{I}\\
[\![M;\mathit{Receive}\ t.\mathcal{A}]\!]\ \mathcal{I} & \text{iff} & [\![(\{\mathcal{I}(t)\} \cup M);\mathcal{A}]\!]\ \mathcal{I}\\
[\![M;\mathit{Equality}\ t\ t'.\mathcal{A}]\!]\ \mathcal{I} & \text{iff} & \mathcal{I}(t) = \mathcal{I}(t') \text{ and } [\![M;\mathcal{A}]\!]\ \mathcal{I}\\
[\![M;\mathit{Inequality}\ \bar{x}\ t\ t'.\mathcal{A}]\!]\ \mathcal{I} & \text{iff} & [\![M;\mathcal{A}]\!]\ \mathcal{I} \text{ and}
\end{array}$$
$$(\forall \delta.\ \mathit{subst\text{-}domain}\ \delta = \bar{x} \wedge \mathit{ground}\ (\mathit{subst\text{-}range}\ \delta) \longrightarrow \mathcal{I}(\delta(t)) \neq \mathcal{I}(\delta(t')))$$

The well-formedness condition is also extended. To prevent free variables from being captured by quantifiers during constraint reduction we require the free and the bound variables of constraints to be disjoint. The full well-formedness condition thus becomes the following:

Definition 3.3 An intruder strand $\mathcal{A}$ is *well-formed* iff the free variables and the bound variables of $\mathcal{A}$ are disjoint and $\mathit{wf}_{st}\ \emptyset\ \mathcal{A}$ holds where:

$$\begin{array}{rll}
\mathit{wf}_{st}\ X\ 0 & \text{iff} & \mathit{True}\\
\mathit{wf}_{st}\ X\ (\mathit{Receive}\ t.\mathcal{A}) & \text{iff} & \mathit{fv}(t) \subseteq X \text{ and } \mathit{wf}_{st}\ X\ \mathcal{A}\\
\mathit{wf}_{st}\ X\ (\mathit{Send}\ t.\mathcal{A}) & \text{iff} & \mathit{wf}_{st}\ (X \cup \mathit{fv}(t))\ \mathcal{A}\\
\mathit{wf}_{st}\ X\ (\mathit{Equality}\ t\ t'.\mathcal{A}) & \text{iff} & \mathit{fv}(t') \subseteq X \text{ and } \mathit{wf}_{st}\ (X \cup \mathit{fv}(t))\ \mathcal{A}\\
\mathit{wf}_{st}\ X\ (\mathit{Inequality}\ \bar{x}\ t\ t'.\mathcal{A}) & \text{iff} & \mathit{wf}_{st}\ X\ \mathcal{A}
\end{array}$$

Note that we here allow variables to originate at the left-hand side t of positive checks $\mathit{Equality}\ t\ t'$. Such variables are essentially abbreviations of terms occurring at the right-hand side t' and the variables in t' originate from earlier steps of the constraint. Note also that we have no requirements on the variables occurring in inequalities but that this does not imply that variables can originate from there. Thus each occurrence of a variable in a constraint either still originates from the intruder or is an abbreviation of something that does.

Definition 3.4 (The Composition-only Intruder Model) We define a restricted variant $\vdash_c$ of the relation $\vdash$ that is the closure only under (*Axiom*) and (*Compose*), but omitting (*Decompose*):

$$\frac{}{M \vdash_c t}\ \begin{array}{c}(\mathit{Axiom}),\\ t \in M\end{array} \qquad \frac{M \vdash_c t_1 \quad \cdots \quad M \vdash_c t_n}{M \vdash_c f(t_1, \ldots, t_n)}\ \begin{array}{c}(\mathit{Compose}),\\ f \in \Sigma^n,\\ \mathit{public}\ f\end{array}$$

Figure 3.1: The definition of the composition-only intruder model.

3.1.4 Out-sourcing Analysis

One aspect that makes the lazy intruder complicated, both in terms of an implementation in tools, and in terms of proving completeness and termination of the reduction procedure below, is analysis of terms. For instance, if the intruder learns an encrypted term where the subterm for the key contains a variable, then whether he can decrypt the term may depend on the substitution for that variable. In general, one then has to make a case split: the case that the message can be deciphered and the case that it cannot, since ignoring either case may eliminate solutions. In fact, in the case a message has not been decrypted, after each received message another case split is necessary whether or not the term should now be decrypted. Another complication arises from the fact that a term in the intruder knowledge could directly be a variable that may represent a decryptable term, and one has to carefully argue that the term in question was known to the intruder earlier (and could have been decrypted then), but this argument requires that the earlier constraints have already been simplified.

To avoid these complications, we now consider the following idea: we limit the intruder to composition steps only (i.e., we define a restricted variant of $\vdash$, called $\vdash_c$, and which is defined in Definition 3.4), and "out-source" all decomposition steps to the transition system. To that end, we can imagine special honest agents that perform decryption operations for the intruder. We discuss this in detail (and prove correctness) in Section 3.3.

One may wonder why we even bother with the lazy intruder and do not simply out-source also the composition steps of $\vdash_c$ as well. Recall that the lazy intruder was conceived to counter the problem that the $\vdash_c$ closure is infinite (while closure under analysis is finite). In other words, in a forward exploration of a transition system, composition leads to blind exploration (while analysis is not a problem). A backwards search like lazy intruder constraint solving is a clear demand-driven way to handle composition steps. Exactly this demand-driven, lazy aspect is what we shall exploit in the typing results: while the intruder *can* compose ill-typed messages, this is *never necessary* to mount the attack, and this "never necessary" is captured by the laziness of the intruder. Not having analysis steps as part of that argument does not hurt, because the analysis of terms is not what introduces ill-typed messages. In fact, we believe that even for automated tools that use the lazy intruder technique, the out-sourcing of analysis could be beneficial, since it drastically simplifies the technique, and as far as we can see every provision for efficiency (e.g., eagerly performing analysis steps that require no substitution) can be similarly applied at the transition system level.

We define a variant $[\![\cdot;\cdot]\!]_c$ of the semantics $[\![\cdot;\cdot]\!]$ restricted to composition steps by replacing $\vdash$ with $\vdash_c$ in the definition of $[\![\cdot;\cdot]\!]$. Similarly, we define $\models_c$ by replacing $[\![\cdot;\cdot]\!]$ with $[\![\cdot;\cdot]\!]_c$ in the definition of $\models$.

3.1.5 Constraint Reduction

The goal of the constraint reduction procedure is to determine all solutions of a constraint. There are in general infinitely many solutions, but they can be finitely represented, namely by *simple* constraints:

Definition 3.5 An intruder strand $\mathcal{A}$ is *simple* if and only if

1. $t \in \mathcal{V}$ for every *Send* t that occurs in $\mathcal{A}$,
2. no positive check *Equality* t t' occurs in $\mathcal{A}$, and
3. every negative check *Inequality* $\bar{x}$ t t' occurring in $\mathcal{A}$ is satisfiable.

The point is that simple constraints are always satisfiable: the remaining variables are arbitrary, so the intruder can choose any term from his knowledge. A key point of the typing result is: when the variables are annotated with an intended type (and the intruder knows values for each type), then there always also exists a well-typed solution for a simple constraint.

Note that inequalities occurring in simple constraints must be satisfiable. Satisfiability of inequalities can in fact be determined using the *mgu* algorithm: given *Inequality* $\bar{x}$ t t' pick any substitution θ whose range consists of fresh constants and whose domain covers the free variables of the inequality and is disjoint from the bound variables. Then *Inequality* $\bar{x}$ t t' is satisfiable if and only if $\theta(t)$ and $\theta(t')$ are not unifiable (i.e., *mgu* $\theta(t)$ $\theta(t') =$ *None* where the data constructor

None is from the *option* data type and indicates here that there is no most-general unifier of $\theta(t)$ and $\theta(t')$). Lemma 3.7 shows that such a verification procedure is sufficient.

However, it is not trivial to prove that a substitution θ as defined in the previous paragraph is a solution to a satisfiable inequality. An attempt at proving such a statement exists in [AMMV15] (more specifically, their Lemma 2), but we discovered during our Isabelle-formalization effort that their proof is incomplete at best: They rely on obtaining an arbitrary substitution ξ whose range needs to be disjoint from the range of θ, but θ is fixed before ξ is obtained and so the substitution ranges may overlap. We saw no way to repair their proof, and so we decided to prove the statement differently, in Lemma 3.6. Note that Lemma 3.6 will also be useful later on and for that purpose we here prove a slightly more general lemma than what is needed at this point.

Lemma 3.6 *Let $\theta = [\bar{x} \mapsto \bar{l}]$ be a fresh substitution w.r.t. terms s and t where $\bar{l}$ are ground terms, and let $\sigma = [\bar{x} \mapsto \bar{u}]$ where $\bar{u}$ are ground terms. Assume that at least one of the following conditions hold:*

1. *all terms in the range of θ (i.e., $\bar{l}$) are constants, or*
2. *there does not exist a subterm of s or t of the form $f(z_1, \dots, z_n)$ such that $z_1, \dots, z_n$ are variables, $n > 0$, and $\{z_1, \dots, z_n\} \cap \bar{x} = \emptyset$.*

If $\theta(s)$ and $\theta(t)$ are unifiable then $\sigma(s)$ and $\sigma(t)$ are unifiable.

Proof. We essentially need to prove that unifiability of s and t are independent of the variables in $\bar{x}$, if we know that $\theta(s)$ and $\theta(t)$ are unifiable. Intuitively, this means that a most general unifier of s and t cannot constrain the variables in $\bar{x}$. Otherwise some instance (i.e., substitution) ς of the variables from $\bar{x}$ would not be supported by a most general unifier of s and t, rendering $\varsigma(s)$ and $\varsigma(t)$ non-unifiable.

First, we obtain an mgu δ of s and t. This is possible since $\theta(s)$ and $\theta(t)$ are unifiable. Together with our assumptions, Lemma 2.3, and Lemma 2.4 we have the following properties of δ:

$$\text{For any unifier } \gamma \text{ of } \theta(s) \text{ and } \theta(t) \text{ there exists a } \tau \text{ such that } \theta \cdot \gamma = \delta \cdot \tau \tag{1}$$

$$\text{If } c \sqsubseteq \delta(z) \text{ then either } c \sqsubseteq s \text{ or } c \sqsubseteq t, \text{ for any } z \in \mathit{dom}(\delta) \text{ and } c \in \mathcal{C} \tag{2}$$

$$\mathit{dom}(\delta) \cap \mathit{fv}(\mathit{ran}(\delta)) = \emptyset \tag{3}$$

$$\begin{gathered}\text{If } f(z_1, \dots, z_n) \sqsubseteq \delta(z) \text{ then there exists a subterm of } s \text{ or } t \\ \text{of the form } f(z'_1, \dots, z'_n) \text{ for some } z'_1, \dots, z'_n \in \mathcal{V} \text{ such that} \\ \delta(z'_i) = z_i \text{ for all } i \in \{1, \dots, n\}, \text{ for any } z \in \mathcal{V} \text{ and } f \in \Sigma\end{gathered} \tag{4}$$

From (1) we can obtain a substitution τ such that $\theta(x) = \tau(\delta(x))$ for all $x \in \bar{x}$.

In the following let $\bar{y}$ denote the variables $(\mathit{fv}(s) \cup \mathit{fv}(t)) \setminus \bar{x}$. Suppose that there exists an $x \in \mathit{dom}(\delta) \cap \bar{x}$. Then $\theta(x) = \tau(\delta(x))$. Now suppose also that $x' \in \mathit{fv}(\delta(x))$ for some $x' \in \bar{x}$. Then $x \neq x'$ and $\tau(x') \sqsubseteq \tau(\delta(x))$ and $\tau(\delta(x')) = \theta(x')$. The variable x' cannot be in the domain of δ because of (3) and so $\tau(\delta(x')) = \tau(x') = \theta(x')$. But then $\theta(x') \sqsubseteq \theta(x)$, contradicting subterm injectivity of θ. Therefore $\mathit{fv}(\delta(\mathit{dom}(\delta) \cap \bar{x})) \subseteq \bar{y}$.

We can furthermore show that δ is injective on $\mathit{dom}(\delta) \cap \bar{x}$: If $\delta(x) = \delta(x')$, for arbitrary $x, x' \in \bar{x}$, then $\theta(x) = \tau(\delta(x)) = \tau(\delta(x')) = \theta(x')$ and so $x = x'$ because of injectivity of θ.

Hence δ is of the form

$$\delta = [y_1 \mapsto t_1, \dots, y_k \mapsto t_k, x_1 \mapsto s_1, \dots, x_\ell \mapsto s_\ell]$$

where $\mathit{fv}(\{s_1, \dots, s_\ell\}) \subseteq \bar{y} \setminus \{y_1, \dots, y_k\}$.

We can also show that each s_i is a variable, implying that $\delta(\mathit{dom}(\delta) \cap \bar{x}) \subseteq \bar{y}$. Suppose that there exists a variable $x_i \in \mathit{dom}(\delta) \cap \bar{x}$ such that $\delta(x_i)$ is a composed term or a constant. No constant c can occur in $\delta(x_i)$ because otherwise $c \sqsubseteq \tau(\delta(x_i)) = \theta(x_i)$ by the properties of τ and either $c \sqsubseteq s$ or $c \sqsubseteq t$ by (2), contradicting freshness of θ. So all atomic subterms of $\delta(x_i)$ must be variables. We can now perform a case analysis on whether $\bar{l} \subseteq \mathcal{C}$:

- If $\bar{l} \subseteq \mathcal{C}$ then $\theta(x_i) = c_i$ for some $c_i \in \mathcal{C}$. Since $\theta(x_i) = \tau(\delta(x_i))$ and since $\delta(x_i)$ is either a composed term or a constant it must be the case that $\delta(x_i) = c_i$. However, we just showed that no constant can occur in $\delta(x_i)$ and so we have arrived at a contradiction.

- If $\bar{l} \not\subseteq \mathcal{C}$ then we know by the assumptions of this lemma that no subterm of the form $f(y_1, \dots, y_n)$ where $n > 0$ can occur in s or t. We also know that $\mathit{fv}(\delta(x_i)) \subseteq \bar{y}$ and so we can obtain a subterm u' of $\delta(x_i)$ of the form

$f(y'_1, \dots, y'_m)$ where $\{y'_1, \dots, y'_m\} \subseteq \bar{y}$, because no constant occur in $\delta(x_i)$. This term u' cannot occur in s or t by the assumptions of this lemma. So it must be the case that there exists some $j \in \{1, \dots, m\}$ such that $\delta(x_j) = y'_j$ by (4), and so also $x_i \neq x_j$ because $\delta(x_i)$ is a composed term while $\delta(x_j)$ is not. But then y'_j occurs in both $\delta(x_i)$ and $\delta(x_j)$, implying that $\theta(x_i)$ and $\theta(x_j)$ share the subterm $\tau(y'_j)$, contradicting subterm injectivity of θ.

In both cases we arrive at a contradiction and so the assumption that $\delta(x_i)$ is either a constant or a composed term is false. Hence $\delta(x_i) \in \bar{y} \setminus \{y_1, \dots, y_k\}$ for all $x_i \in \mathit{dom}(\delta) \cap \bar{x}$ and so $\{s_1, \dots, s_\ell\} \subseteq \bar{y} \setminus \{y_1, \dots, y_k\}$.

We can now obtain a unifier of s and t whose domain is a subset of $\bar{y}$ as follows: Let $\alpha = [s_1 \mapsto x_1, \dots, s_\ell \mapsto x_\ell]$. Then $\delta \cdot \alpha$ (i.e., $\lambda z.\ \alpha(\delta(z))$) is a unifier of s and t because δ is, and for each $x_i \in \mathit{dom}(\delta) \cap \bar{x}$ the variable renaming α sends $\delta(x_i)$ back to x_i, so $x_i \notin \mathit{dom}(\delta \cdot \alpha)$.

Let $\delta' = \delta \cdot \alpha$. Since δ' is a unifier of s and t we have that $\delta' \cdot \sigma$ is also a unifier of s and t. We now show that $\sigma \cdot \delta' \cdot \sigma = \delta' \cdot \sigma$, which implies that $\delta' \cdot \sigma$ is a unifier of $\sigma(s)$ and $\sigma(t)$. For any variable z:

- If $z \in \bar{x}$ then $\sigma(z) = u$ is a ground term and $z \notin \mathit{dom}(\delta')$. Hence $\sigma(\delta'(\sigma(z))) = \sigma(\delta'(u)) = u$ and $\sigma(\delta'(z)) = \sigma(z) = u$.
- If $z \in \bar{y}$ then $z \notin \mathit{dom}(\sigma)$. Hence $\sigma(\delta'(\sigma(z))) = \sigma(\delta'(z))$.
- Otherwise $\sigma(\delta'(\sigma(z))) = z = \sigma(\delta'(z))$.

Thus $\sigma(s)$ and $\sigma(t)$ are unifiable. □

We can now use Lemma 3.6 to show that satisfiability of inequalities can be determined using the *mgu* algorithm:

Lemma 3.7 [3] *Let $\theta = [\bar{y} \mapsto \bar{c}]$ be a fresh substitution w.r.t. the terms t and t' where $\bar{c}$ are fresh constants and $\bar{y}$ are the free variables of Inequality $\bar{x}$ t t'. Then Inequality $\bar{x}$ t t' is satisfiable if and only if $\theta(t)$ and $\theta(t')$ are not unifiable.*

Proof. Note that $\mathit{fv}(\theta(t)) \subseteq \bar{x}$ and $\mathit{fv}(\theta(t')) \subseteq \bar{x}$. Note also that θ is injective because $\bar{c}$ are fresh constants.

[3] Note that the statement *"If $\theta(t)$ and $\theta(t')$ are not unifiable then Inequality $\bar{x}$ t t' is satisfiable"* is not formalized in Isabelle yet. The other direction of the implication—namely the one relying on Lemma 3.6 and the one needed in later proofs—is fully formalized in Isabelle.

If *Inequality* $\bar{x}$ t t' is unsatisfiable then $\sigma(t)$ and $\sigma(t')$ must be unifiable for any ground substitution σ whose domain is the set of free variables of the inequality. So in particular $\theta(t)$ and $\theta(t')$ must be unifiable. Therefore, if $\theta(t)$ and $\theta(t')$ are not unifiable then *Inequality* $\bar{x}$ t t' is satisfiable.

For the other direction we can apply Lemma 3.6. □

The goal of the reduction procedure is now to transform a given constraint into an equivalent set of simple constraints. That is, we define a reduction relation $\rightsquigarrow$ on constraint states[4] (see Definition 3.8), and for a given $(\mathcal{A}_0, \theta_0)$, we consider the reachable constraints, i.e., $(\mathcal{A}_0, \theta_0) \rightsquigarrow^* (\mathcal{A}, \theta)$. (The relation $\rightsquigarrow^*$ denotes the reflexive transitive closure of $\rightsquigarrow$ while $\rightsquigarrow^+$ then denotes the transitive closure.) We prove in Isabelle that there are finitely many (termination), and that the union of the models of the reachable simple constraints is exactly the set of models of $(\mathcal{A}_0, \theta_0)$ (soundness and completeness). Strictly speaking, for termination we prove that the relation $\rightsquigarrow$ is well-founded, i.e., there are no infinite reduction chains. To prove that there are only finitely many reachable constraints one could furthermore prove that the relation is finitely branching and then apply König's lemma. However, well-foundedness alone is sufficient for our purposes.

The $(Compose_{LI})$ rule of Definition 3.8 corresponds to the $(Compose)$ rule of the Dolev-Yao model, i.e., if the intruder has to produce $f(t_1, \ldots, t_n)$ for a public symbol f, one way to do it is to produce each of the t_i (in whatever way) and apply f to them.

The $(Unify_{LI})$ rule of Definition 3.8 corresponds to the $(Axiom)$ rule of the Dolev-Yao model: it states that the intruder can send a message s if he has previously learned a message t that can be unified with s. More precisely, the most general unifier δ of s and t (if it exists) represents all interpretations of the constraint, under which s and t are equal, and thus under which the *Send* s can be removed as this requirement is satisfied. However, we have to integrate δ by composing it with the existing solution θ and applying it to the rest of the constraint, so that no variable in the domain of δ remains in the intruder strand.

Note that the $(Unify_{LI})$ rule is not applicable if the term s to be generated is a variable, because we are lazy: since so far there is no more constraint on

[4]This relation $\phi \rightsquigarrow \psi$ is sometimes written $\frac{\psi}{\phi}$, i.e, in the form of a proof calculus for satisfiability. One can read each such rule top down: every solution of ψ is also a solution of ϕ. The procedure, however, works by backwards exploring the rules: the solutions of ϕ include all solutions of ψ.

Definition 3.8 The lazy intruder is the least relation $\rightsquigarrow$ between constraint states closed under the following rules (where *map* is provided by Isabelle and is defined as usual, i.e., it satisfies the equation $map\ f\ [a_1, \ldots, a_n] = [f\ a_1, \ldots, f\ a_n]$):

$(Compose_{LI})$:
$$(\mathcal{A}.Send\ f(t_1, \ldots, t_n).\mathcal{A}', \theta) \rightsquigarrow (\mathcal{A}.(map\ Send\ [t_1, \ldots, t_n]).\mathcal{A}', \theta)$$
$$\text{if } simple\ \mathcal{A}, f \in \Sigma^n, \text{and } public\ f$$

$(Unify_{LI})$:
$$(\mathcal{A}.Send\ s.\mathcal{A}', \theta) \rightsquigarrow (\delta(\mathcal{A}.\mathcal{A}'), \theta \cdot \delta)$$
$$\text{if } s, t \notin \mathcal{V}, simple\ \mathcal{A}, t \in ik_{st}\ \mathcal{A}, \text{and } mgu\ s\ t = Some\ \delta$$

$(Equality_{LI})$:
$$(\mathcal{A}.Equality\ s\ t.\mathcal{A}', \theta) \rightsquigarrow (\delta(\mathcal{A}.\mathcal{A}'), \theta \cdot \delta)$$
$$\text{if } simple\ \mathcal{A} \text{ and } mgu\ s\ t = Some\ \delta$$

Figure 3.2: The definition of the lazy intruder.

what s should be precisely, it is pointless to explore options—the intruder can always generate *something*. This is a key to the typing result later. In following reduction steps, this variable may be replaced with a more concrete term, and then we explore how that term can be generated. Finally, the rule also forbids that the term t that we unify with s is a variable (and this is again crucial in the typing result), but that this restriction is without loss of generality is a tricky part of the completeness proof.

For the $(Equality_{LI})$ rule of Definition 3.8 we simply compute the most general solution δ to the equality constraint. As with the $(Unify_{LI})$ rule we must integrate δ by composing it with the existing solution θ and apply δ to the remaining constraint.

Note that in contrast to many other lazy intruder methods, these rules are only applicable to the first step of the form $Equality\ t\ t'$ or $Send\ t$ where t is not a variable, i.e., all prior constraint steps must be simple already. This restriction is without loss of generality again as our proofs show; since this removes some non-determinism, it also makes some arguments later easier, because we can rely on the prefix to be simple.

Example 3.3 *Let us reduce the constraint from Example 3.1 (with [] as initial substitution). With one $\rightsquigarrow$ step (addressing the first Send) we can get (using $(Compose_{LI})$) to: $\ldots Send\ \mathsf{pub}(x).Send\ y.Receive\ y.Send\ \mathsf{secret}.0$.*

Since we cannot unify the $\mathsf{pub}(x)$ with anything, we then are forced to make *another compose, leading to:* $\ldots Send\ x.Send\ y.Receive\ y.Send\ \mathsf{secret}.0$ *The remaining non-simple Send* secret *cannot be solved (if* $\mathsf{secret} \in \mathcal{C}_{priv}$*), i.e., it has no successor under* $\rightsquigarrow$*. Since it is not simple, it does not have any solution by completeness of the lazy intruder (the actual completeness theorem will be introduced later).*

From the original constraint we can however also reach another constraint when using ($Unify_{LI}$*) between the received encrypted message and the one to be sent, giving the unifier* $\delta = [x \mapsto \mathsf{k_a}, y \mapsto \mathsf{secret}]$*:* $\ldots Receive\ \mathsf{secret}.Send\ \mathsf{secret}.0$ *which can trivially be solved with another unify step.* □

3.1.6 Proving Soundness & Completeness

A large part of the contribution of this chapter lies in the Isabelle proof of the soundness and completeness of lazy intruder reduction. This is following basically the proofs in [AMMV15]. In fact, with analysis (that we out-source to the transition system) we found several mistakes in [AMMV15]; these are discussed on the transition system level in Section 3.3.2, along with a correction.

We first prove that all reductions preserve well-formedness of the constraint states:

Lemma 3.9 (Well-formedness preservation) *If* $(\mathcal{A}_1, \theta_1)$ *is a well-formed constraint state and* $(\mathcal{A}_1, \theta_1) \rightsquigarrow^* (\mathcal{A}_2, \theta_2)$ *then* $(\mathcal{A}_2, \theta_2)$ *is also a well-formed constraint state.*

Proof. Such invariants of the reduction system are of course first proved for single reductions $\rightsquigarrow$ and then follows for $\rightsquigarrow^*$ by an induction on the reflexive transitive closure. The single-reduction case follows from a case analysis on the lazy intruder relation. □

From the well-formedness we can quite easily derive soundness, i.e., that no reduction step introduces new solutions:

Theorem 3.10 (Soundness) *If* $(\mathcal{A}, \theta)$ *is well-formed,* $(\mathcal{A}, \theta) \rightsquigarrow^* (\mathcal{A}', \theta')$*, and* $\mathcal{I} \models_c (\mathcal{A}', \theta')$*, then* $\mathcal{I} \models_c (\mathcal{A}, \theta)$*.*

Proof. Again proven by induction on the reflexive transitive closure and case analysis on the lazy intruder relation. □

The proof of completeness relies on the termination which we thus prove first.

Lemma 3.11 (Termination) *The relation $\rightsquigarrow$ is well-founded, i.e., for any state $(\mathcal{A},\theta)$ there are no infinite reduction chains $(\mathcal{A},\theta) \rightsquigarrow (\mathcal{A}',\theta') \rightsquigarrow \cdots$.*

Proof. Not having to deal with analysis rules in this proof makes matters simpler. The standard approach is here to define a well-founded measure $<$ on constraint states, and show that on every reduction the constraint state decreases according to the measure, so there cannot be an infinite chain $(\mathcal{A}_1,\theta_1) \rightsquigarrow (\mathcal{A}_2,\theta_2) \rightsquigarrow \cdots$ of reductions. The measure $<$ is lexicographically first in the number of free variables and secondly in the size of constraints (sum of the size of constraint terms). A unification or equality step where the unifier is not the identity reduces the first component of this measure, while all other steps leave it equal and reduce the second component. $\square$

The next step is to show that all simple constraints are satisfiable:

Lemma 3.12 (Simple constraints are satisfiable) *If $(\mathcal{A},\theta)$ is well-formed and $\mathcal{A}$ is simple then there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models_c (\mathcal{A},\theta)$.*

Proof. Here the idea is to consider any injective interpretation $\mathcal{I}$ that interprets the variables of $\mathcal{A}$ with fresh elements of $\mathcal{C}_{pub}$ (i.e., the range of $\mathcal{I}$ consists of public constants that do not occur anywhere in $(\mathcal{A},\theta)$). Then all *Send* steps of the simple intruder strand $\mathcal{A}$ are satisfied under $\mathcal{I}$. We also know from Lemma 3.7 that $\mathcal{I}$ must be a solution to the inequalities occurring in $\mathcal{A}$. Since *subst-domain* θ and *fv*(*subst-range* θ) are disjoint we know that $\theta \cdot \theta = \theta$ and so θ supports $\theta \cdot \mathcal{I}$. Finally, since the domain of θ is disjoint from the variables of $\mathcal{A}$ we have that $(\theta \cdot \mathcal{I})(\mathcal{A}) = \mathcal{I}(\mathcal{A})$ and thus $\theta \cdot \mathcal{I}$ is a solution to $(\mathcal{A},\theta)$. $\square$

The most difficult lemma to prove is that for any non-simple constraint with a solution $\mathcal{I}$ there exists a reduction $\rightsquigarrow$ that preserves $\mathcal{I}$.

Lemma 3.13 (Completeness, single step) *If $(\mathcal{A},\theta)$ is a well-formed constraint state, $\mathcal{I} \models_c (\mathcal{A},\theta)$, and $\mathcal{A}$ is not simple, then there exists $(\mathcal{A}',\theta')$ such that $(\mathcal{A},\theta) \rightsquigarrow (\mathcal{A}',\theta')$ and $\mathcal{I} \models_c (\mathcal{A}',\theta')$.*

Proof. In a nutshell, when given a constraint $(\mathcal{A},\theta)$ with a model $\mathcal{I}$, then every sent term t must be composed from the set M of previously received terms under $\mathcal{I}$, (i.e., $\mathcal{I}(M) \vdash_c \mathcal{I}(t)$), every *Equality* $t\ t'$ occurring in $\mathcal{A}$ is satisfied (i.e., $\mathcal{I}(t) = \mathcal{I}(t')$), and every *Inequality* $\bar{x}\ t\ t'$ of $\mathcal{A}$ is satisfied.

Since $\mathcal{A}$ is satisfiable and not simple we know that there exists a prefix of $\mathcal{A}$ of the form $\mathcal{B}.\mathit{Send}\ t$ or $\mathcal{B}.\mathit{Equality}\ t'\ t''$ where $\mathcal{B}$ is a simple intruder constraint and t is not a variable.

In the case where the prefix ends with *Equality* $t'\ t''$ we know that $\mathcal{I}(t') = \mathcal{I}(t'')$ and so there exists a most general unifier δ of t' and t''. We can therefore apply the ($\mathit{Equality}_{LI}$) rule to proceed with the reduction. We can also prove that δ supports $\mathcal{I}$ because δ is a most general unifier of t' and t'', and so $\mathcal{I}$ is also a model of the reduced constraint after application of ($\mathit{Equality}_{LI}$).

Otherwise, consider the first *Send* t of $\mathcal{A}$ where t is not a variable. Recall that $\vdash_c$ is inductively defined, by a set of rules: i.e., $\mathcal{I}(t)$ is either obtained with the (*Axiom*) rule or with the (*Compose*) rule defining $\vdash_c$. In the (*Compose*) case, we can invoke the ($\mathit{Compose}_{LI}$) rule of the lazy intruder and get to the solution. In case of the (*Axiom*) rule, there must be a term $s \in M$ such that $\mathcal{I}(s) = \mathcal{I}(t)$. If s is not a variable, we can use the (Unify_{LI}) rule. If $s \in \mathcal{V}$ however, then by well-formedness s occurs previously in the part of the constraint that is already simple, i.e., *Send* s occurs previously in the strand and the intruder thus uses a term that he previously sent. This term $\mathcal{I}(s)$ was itself derived with $\vdash_c$ using (*Axiom*) or (*Compose*). In the case of (*Axiom*), there is another earlier received term s' with $\mathcal{I}(s') = \mathcal{I}(t)$ which we could have picked instead. In the (*Compose*) case the ($\mathit{Compose}_{LI}$) of the lazy intruder is applicable. $\square$

More generally, the proof shows that for every concrete solution $\mathcal{I}$, for every *relevant* step of the ground Dolev-Yao intruder $\vdash_c$ under $\mathcal{I}$, we can find a corresponding step of the lazy intruder—where *relevant* steps are only those steps that are needed to solve one non-simple step of the constraint. The difficult cases in the proof are those where we need to regress to other steps that the intruder has made before (exploiting the well-formedness). Note that if one also considers analysis steps, this regression becomes much more complicated, since one needs to consider also analysis of terms that may be a variable or that resulted from a composition step. For that reason one needs then to also consider proof normalization, i.e., eliminating redundant parts of proofs where the intruder first composes terms that he then decomposes again.

From this, we obtain the completeness: a well-formed constraint is either simple (and thus satisfiable), or we can make further reductions (and no solution gets lost), or else we are stuck at an irreducible constraint (that is thus unsatisfiable). Together with termination we thus have that all satisfiable constraints can be reduced to simple ones without losing any solutions:

Theorem 3.14 (Completeness) *If $(\mathcal{A}, \theta)$ is well-formed and $\mathcal{I} \models_c (\mathcal{A}, \theta)$ then there exists a $(\mathcal{A}', \theta')$ such that $\mathcal{A}'$ is simple, $(\mathcal{A}, \theta) \leadsto^* (\mathcal{A}', \theta')$, and $\mathcal{I} \models_c (\mathcal{A}', \theta')$.*

Proof. If $\mathcal{A}$ is simple then the conclusion follows by reflexivity of $\leadsto^*$. So in the following we will assume that $\mathcal{A}$ is not simple.

We say that a state $(\mathcal{B}, \sigma)$ is *stuck* if there is no successor state $(\mathcal{B}', \sigma')$ in $\leadsto$ with model $\mathcal{I}$. From Lemma 3.9 and Lemma 3.13 it follows that for any stuck $(\mathcal{B}, \sigma)$ with model $\mathcal{I}$ where $(\mathcal{A}, \theta) \leadsto^+ (\mathcal{B}, \sigma)$ the constraint $\mathcal{B}$ must be simple.

Now denote by $\leadsto_\mathcal{I}$ the subrelation of $\leadsto$ defined as $(\mathcal{B}, \sigma) \leadsto_\mathcal{I} (\mathcal{B}', \sigma')$ iff $(\mathcal{B}, \sigma) \leadsto (\mathcal{B}', \sigma')$ and $\mathcal{I}$ is a model of both $(\mathcal{B}, \sigma)$ and $(\mathcal{B}', \sigma')$, for any two states $(\mathcal{B}, \sigma)$ and $(\mathcal{B}', \sigma')$. The transitive closure is then denoted by $\leadsto_\mathcal{I}^+$.

The idea is now to show that there exists a stuck state $(\mathcal{A}', \theta')$ reachable from $(\mathcal{A}, \theta)$ in $\leadsto_\mathcal{I}$, i.e., $(\mathcal{A}, \theta) \leadsto_\mathcal{I}^+ (\mathcal{A}', \theta')$ where $\leadsto_\mathcal{I}^+ (\mathcal{A}', \theta')$ has no successor in $\leadsto_\mathcal{I}$. This implies that such a $\mathcal{A}'$ is simple, and that $\mathcal{I}$ is a model of $(\mathcal{A}', \theta')$, thereby proving the theorem. The proof is by contradiction: We first assume that there is no stuck state $(\mathcal{A}', \theta')$ reachable from $(\mathcal{A}, \theta)$ in $\leadsto_\mathcal{I}$ and then we show that this assumption leads to a contradiction. More specifically, we will show that there exists an infinite chain in $\leadsto_\mathcal{I}$, contradicting Lemma 3.11 because $\leadsto_\mathcal{I}$ is a subrelation of $\leadsto$.

Formally, we express in higher-order logic an infinite chain in $\leadsto_\mathcal{I}$ as a function f from the natural numbers to states such that $f\ i \leadsto_\mathcal{I} f\ (i + 1)$ for all $i \in \mathbb{N}$. We can easily construct such a function if we can show the following:

1. There exists a state $(\mathcal{B}, \sigma)$ such that $(\mathcal{A}, \theta) \leadsto_\mathcal{I} (\mathcal{B}, \sigma)$.
2. For any $(\mathcal{B}, \sigma)$ such that $(\mathcal{A}, \theta) \leadsto_\mathcal{I}^+ (\mathcal{B}, \sigma)$ there exists some state $(\mathcal{B}, \sigma')$ where $(\mathcal{B}, \sigma) \leadsto_\mathcal{I} (\mathcal{B}', \sigma')$.

An application of Lemma 3.13 proves the first requirement. So all that remains is to show that there always exists a successor state in $\leadsto_\mathcal{I}$ for any state $(\mathcal{B}, \sigma)$ reachable from $(\mathcal{A}, \theta)$ in $\leadsto_\mathcal{I}$. However, we know by assumption that any such $(\mathcal{B}, \sigma)$ cannot be stuck, and so we can always pick a successor state of $(\mathcal{B}, \sigma)$ in $\leadsto_\mathcal{I}$, proving the second requirement. $\square$

3.2 Typed Model

So far there has been no notion of types. We now define a simple type system and consider the restriction where the intruder is limited to sending only well-typed messages. The main result of this section is the formalization of a typing result on intruder constraints for a large class of protocols, i.e., that the restrictions imposed on the intruder is without loss of attacks for many protocols: if a constraint state $(\mathcal{A}, \theta)$ has a solution $\mathcal{I}$, then it also has a well-typed solution $\mathcal{I}'$. Thus, if we can verify a protocol in a typed model (all constraints that arise only have well-typed solutions) then we can infer that it is also secure in the untyped model. In this section we first develop the typing result on the level of constraints (without analysis) and then extend it to entire protocols and transition systems in Section 3.3.

3.2.1 The Type System

Recall that our notion of terms is parameterized over a set Σ of function symbols, so one can easily introduce new operators without updating all the proofs. Similarly, for the type system we introduce another set over which our result is parameterized: a finite set $\mathfrak{T}_a$ of *atomic* types. An example is $\mathfrak{T}_a = \{\mathsf{Agent}, \mathsf{Nonce}, \mathsf{SymmetricKey}, \mathsf{PrivateKey}\}$. (Note that public keys do not have an atomic type in this example, because we build them using operator pub from private keys.) Next, we introduce composed types as built like terms from $\mathfrak{T}_a$ and the operators of Σ, for instance $\mathsf{crypt}(\mathsf{pub}(\mathsf{PrivateKey}), \mathsf{Nonce})$ could be a composed type. As types are thus very similar to normal terms, we re-use the definition for terms:

$$\textbf{type-synonym } (\Sigma, \mathfrak{T}_a)\ \textit{term-type} \equiv (\Sigma, \mathfrak{T}_a)\ \textit{term}$$

Thus, we put atomic types in every place where normal terms would have variables. (Note that our type system has no type variables, atomic types are like constants.) To avoid confusion and make definitions nicer to read, we introduce two synonyms for the constructors *Var* and *Fun* of terms, namely *TAtom* and *TComp*, and consistently use them when talking about types in Isabelle-notation. We inherit all previous notions from terms, e.g., well-formedness for types (all operators are used with correct arity). However, we additionally require that no constants occur in types. Further, our result is parameterized over a *typing function* $\Gamma : (\Sigma, \mathcal{V})\ \textit{term} \Rightarrow (\Sigma, \mathfrak{T}_a)\ \textit{term-type}$ that maps each term to a type and that must satisfy the following properties:

1. $\Gamma(c) \in \mathfrak{T}_a$ for every $c \in \mathcal{C}$.

2. $\Gamma(f(t_1,\ldots,t_n)) = f(\Gamma(t_1),\ldots,\Gamma(t_n))$ for every $f \in \Sigma \setminus \mathcal{C}$.
3. $\Gamma(v)$ must be a well-formed type for every $v \in \mathcal{V}$.

In fact, it is sufficient to specify Γ for all constants (as atomic types) and all variables (as arbitrary well-formed types), and then homomorphically extend Γ to arbitrary terms. Thus, every well-formed term t has a well-formed type $\Gamma(t)$.

Finally, we want to give the intruder an unbounded supply of terms of every type. Thus we require the following:

1. $\mathcal{C}_{pub}$ contains infinitely many constants of every atomic type:
$$\forall a.\ \mathit{infinite}\ \{c.\ \Gamma\ (\mathit{Fun}\ c\ [])\ =\ \mathit{TAtom}\ a \wedge \mathit{public}\ c\}$$
2. all functions of arity greater than zero (i.e., all symbols of $\Sigma\setminus\mathcal{C}$) are public.

Note that we now only consider public function symbols for the symbols occurring in $\Sigma \setminus \mathcal{C}$, i.e., we require that *public* returns true for all symbols of $\Sigma \setminus \mathcal{C}$. One can simulate however a private function symbol of arity $n > 0$ by a public function symbol of arity $n+1$ where the additional argument is used with a special constant that is never given to the intruder; in this way all results can be lifted to a model with both private and public function symbols. For instance we can encode $\mathsf{inv} \in \Sigma^1$ in terms of a public symbol $\mathsf{inv}' \in \Sigma^2$ and a special secret constant $\mathsf{sec}_{\mathsf{inv}}$.

Note also that we require every atomic type to have an infinite number of values. This is necessary because we need to find solutions to inequalities, and for that reason we need to always be able to pick an arbitrary number of fresh constants.

To model the type system in Isabelle/HOL we extend the *intruder-model* locale by parameterizing over typing functions Γ that satisfy our requirements:

locale *typed-model* = *intruder-model arity public* Ana
  for *arity* :: $\Sigma \Rightarrow$ *nat*
  and *public* :: $\Sigma \Rightarrow$ *bool*
  and Ana :: $(\Sigma,\mathcal{V})$ *term* $\Rightarrow$ $((\Sigma,\mathcal{V})$ *term set* $\times$ $(\Sigma,\mathcal{V})$ *term set*$)$
$+$ **fixes** Γ :: $(\Sigma,\mathcal{V})$ *term* $\Rightarrow$ $(\Sigma,\mathfrak{T}_a$ :: *finite*$)$ *term-type*
  assumes $\bigwedge c.\ \mathit{arity}\ c = 0 \Longrightarrow \exists a.\ \forall T.\ \Gamma(\mathit{Fun}\ c\ T) = \mathit{TAtom}\ a$
  and $\bigwedge f\ T.\ \mathit{arity}\ f > 0 \Longrightarrow \Gamma(\mathit{Fun}\ f\ T) = \mathit{TComp}\ f\ (\mathit{map}\ \Gamma\ T)$
  and $\bigwedge a.\ \mathit{infinite}\ \{c.\ \Gamma(\mathit{Fun}\ c\ []) = \mathit{TAtom}\ a \wedge \mathit{public}\ c\}$
  and $\bigwedge t\ f\ T.\ \mathit{TComp}\ f\ T \sqsubseteq \Gamma(t) \Longrightarrow \mathit{arity}\ f > 0$
  and $\bigwedge x.\ \mathit{wf}_{\mathit{trm}}\ \Gamma(\mathit{Var}\ x)$
  and $\bigwedge f.\ \mathit{arity}\ f > 0 \Longrightarrow \mathit{public}\ f$

The notation $\mathfrak{T}_a :: \textit{finite}$ expresses that $\mathfrak{T}_a$ must be a type with a finite number of values, i.e., that $\mathfrak{T}_a$ must be of sort *finite*.

An important point why this type system is of foundational interest is that it limits the size of terms that can be substituted for a variable, e.g., when the protocol requires a value to be of type nonce, it cannot be a composed term in the typed model anymore. Abstract interpretation approaches like the one used in ProVerif (where Σ is finite) become decidable under this restriction, and several Isabelle proof methodologies are based on a typed model [Pau98, Pau99, BMP06, Bel07, BM09]. This restriction on substitutions—that they preserve typing—is captured by the following definition:

Definition 3.15 A substitution δ is *well-typed* if and only if $\Gamma(\delta(x)) = \Gamma(x)$ for all $x \in \mathcal{V}$.

The requirement that we need for our typing result is that the messages and sub-messages of a protocol must have a different shape whenever they have different types. For that reason we specify the set of *sub-message patterns* given the set of messages M. In the next section we will use as M the set of all messages of the protocol description (containing variables, hence *message patterns*).

Definition 3.16 (Sub-message patterns) The set of *sub-message patterns* $\mathit{SMP}(M)$ of a set of messages M is defined as the least set closed under the following rules:

1. $M \subseteq \mathit{SMP}(M)$.
2. If $t \in \mathit{SMP}(M)$ and $t' \sqsubset t$ then $t' \in \mathit{SMP}(M)$.
3. If $t \in \mathit{SMP}(M)$ and δ is a well-typed substitution then $\delta(t) \in \mathit{SMP}(M)$.
4. If $t \in \mathit{SMP}(M)$ and $\mathsf{Ana}\ t = (K, T)$ then $K \subseteq \mathit{SMP}(M)$.

The intuition behind this definition is that during constraint reduction we can get to subterms of the initially given terms, apply substitutions, and (when we revisit term analysis) analyze messages. Note that the fourth rule is only closed under the "keys" K required for analysis $\mathsf{Ana}\ t = (K, T)$. This is because the result T of analyzing a term t must always be immediate subterms of t and so the second rule already covers the terms in T. We will show that for the considered class of protocols these substitutions will always be well-typed, so we never fall out of $\mathit{SMP}(M)$.

The intention is that we can apply SMP to the message patterns $\mathit{trms}(\mathcal{P})$ of a protocol $\mathcal{P}$, and $\mathit{SMP}(\mathit{trms}(\mathcal{P}))$ is then an over-approximation of the messages that the intruder might ever learn from the honest agents of $\mathcal{P}$ (or send out to the honest agents) in any well-typed protocol run. The definition is generalized over an arbitrary set of terms, so that we can also apply *SMP* to messages occurring in a strand or a constraint. Consider, for instance, the set of sub-message patterns $\mathit{SMP}(\mathit{trms}(\mathcal{P}))$ built from the terms that occur in some protocol $\mathcal{P}$. The set then covers all message patterns of every message that might be sent over the network, and any pattern in a check made by an honest agent, for well-typed choices of the variables in the patterns.

We can now define the main requirement for our typing result, as a property of the set $\mathit{SMP}(M)$. We require that any two sub-message patterns, that are not variables, are unifiable only if they have the same type. For now we keep the definition on the level of constraints—later, when we formally define our protocol model for this chapter, we will lift all relevant definitions to the protocol-level.

Definition 3.17 (Type-flaw resistance) We say a set M of messages is *type-flaw resistant*, written $\mathit{tfr}_{\mathit{set}}\ M$, where $\mathit{tfr}_{\mathit{set}}$ is defined as follows:

$$\mathit{tfr}_{\mathit{set}}\ M \equiv (\forall s, t \in \mathit{SMP}(M) \setminus \mathcal{V}.\ (\exists \delta.\ \delta(s) = \delta(t)) \longrightarrow \Gamma(s) = \Gamma(t))$$

We may also apply the notion of type-flaw resistance to an intruder strand $\mathcal{A}$ to mean that the set of all terms $\mathit{trms}_{\mathit{st}}\ \mathcal{A}$ occurring in $\mathcal{A}$ is type-flaw resistant, and that the steps of $\mathcal{A}$ satisfy the following predicate:

$$\begin{array}{rll}
\mathit{tfr}_{\mathit{stp}}\ (\mathit{Equality}\ t\ t') & \text{iff} & (\exists \delta.\ \delta(t) = \delta(t')) \longrightarrow \Gamma(t) = \Gamma(t') \\
\mathit{tfr}_{\mathit{stp}}\ (\mathit{Inequality}\ \bar{x}\ t\ t') & \text{iff} & \forall x \in (\mathit{fv}(t) \cup \mathit{fv}(t')) \setminus \bar{x}.\ \Gamma(x) \in \mathfrak{T}_a \\
& \text{or} & \forall f\ T.\ \mathit{Fun}\ f\ T \in \mathit{subterms}\ t \cup \mathit{subterms}\ t' \\
& & \qquad \longrightarrow T = [] \vee (\exists s \in T.\ s \notin \bar{x}) \\
\mathit{tfr}_{\mathit{stp}}\ \mathfrak{s} & \text{iff} & \mathit{True}\ \text{if}\ \mathfrak{s}\ \text{is not an equality or inequality step}
\end{array}$$

Formally, type-flaw resistance of an intruder strand is thus defined as follows:

$$\mathit{tfr}_{\mathit{st}}\ \mathcal{A} \equiv \mathit{tfr}_{\mathit{set}}\ (\mathit{trms}_{\mathit{st}}\ \mathcal{A}) \wedge \mathit{list\text{-}all}\ \mathit{tfr}_{\mathit{stp}}\ \mathcal{A}$$

where *list-all* P L is true iff all elements occurring in the list L satisfy P (we can apply *list-all* to our strands since they are technically lists).

Hence type-flaw resistance requires that the left-hand side and right-hand side of positive checks have the same type if they are unifiable and that the negative checks satisfy one of two different conditions. These conditions are sufficient to show that type-flaw resistant message checks have well-typed models. The condition for positive checks ensures that only well-typed substitutions are produced during constraint reduction when solving these checks. For a negative check we require that its free variables have atomic types or that there is no composed subterm of the form $f(x_1, \ldots, x_n)$ where $n > 0$ and all x_i are bound variables. Essentially, the second condition for negative checks allows us, in most cases, to have both bound and free variables of composed type. This is useful in situations where we wish to model that an honest agent decrypts a message and responds differently depending on whether the message successfully decrypts.

Note that the conditions for inequalities essentially define where we allow free variables of composed type to occur in inequalities. The reason for such restrictions is that inequalities such as *Inequality* $[x_1, \ldots, x_n]\ y\ f(x_1, \ldots, x_n)$ where $n > 0$ (i.e., $\forall x_1, \ldots, x_n.\ y \neq f(x_1, \ldots, x_n)$ in more conventional notation) do not have any well-typed solution if y and $f(x_1, \ldots, x_n)$ have the same type (which is the only interesting case). Thus we require either that all free variables have atomic types or that there is no composed term whose immediate parameters are all bound variables.

The notion of type-flaw resistance also requires that we cannot unify any subterms (except variables) that have different types, i.e., terms that have different meaning must be clearly distinguishable. This is a bit more general than results that are based on adding *tags* to messages to make them distinguishable, like [HLS03, BP05] since we do not impose a particular mechanism to disambiguate messages, such as tags, but rather have a very general definition: to prove type-flaw resistance you just have to ensure that terms of different types are not unifiable (hence distinguishable). We illustrate this with a real-world example, also formalized in Isabelle, by proving type-flaw resistance of TLS.

3.2.2 TLS Example

As a real-world example, let us consider the messages of the TLS Handshake protocol [DR08]. TLS defines several concrete message structuring formats, e.g., the first message of the TLS handshake is called clientHello, and contains essentially five distinct pieces of information (such as a time stamp and a random number); the concrete message format includes also length information and a tag (to distinguish the clientHello from other messages). We represent it in our term algebra by an abstract function of five arguments $\mathsf{clientHello}(T, R, S, C, K)$ and define it as a transparent function in Ana, i.e., the intruder can extract all fields from a known message of this format (without knowing any keys). All other formats of TLS are modeled the same way. The entire TLS handshake protocol can then be represented by the following set of message patterns M (i.e., all messages occurring in the TLS handshake protocol are well-typed instances of the messages in M):

$$\begin{array}{l}
\mathsf{clientHello}(T_1, R_A, S, \mathit{Cipher}, \mathit{Comp}),\\
\mathsf{serverHello}(T_2, R_B, S, \mathit{Cipher}, \mathit{Comp}),\\
\mathsf{serverCert}(\mathsf{sign}(\mathit{Pr}_{ca}, \mathsf{x509}(B, P_B))),\\
\mathsf{clientKeyExchange}(\mathsf{crypt}(P_B, \mathsf{pmsForm}(\mathit{PMS}))),\\
\mathsf{finished}(\mathsf{prf}(\mathsf{clientFinished}(\\
\quad \mathsf{prf}(\mathsf{master}(\mathit{PMS}, R_A, R_B)), R_A, R_B, \mathsf{hash}(\mathit{HSMsgs}))))
\end{array}$$

Here crypt is again asymmetric encryption, sign is signature, and master, prf and hash are one-way functions for hashing, key derivation, and MAC'ing; all other functions are formats. Most variables are of atomic type except for P_B being of type pub(PrivateKey) and *HSMsgs* which represents the concatenation of all handshake messages, i.e., its type is $\mathsf{concat}(\mathsf{clientHello}(\ldots), \ldots, \mathsf{finished}(\ldots))$ for yet another format concat.

One may wonder at this point how this finite set M is sufficient to represent the protocol with an unbounded number of sessions. In fact, we will define below a protocol by an unbounded number of strands for the honest agents (essentially the initial state of a transition system). In fact, the sent and received messages of these strands shall be *well-typed* instances of M: We rename variables so that strands use pairwise disjoint sets of variables, but this renaming is well-typed, and we may instantiate some variables with ground terms, e.g., in all client strands the variable *PMS* shall be instantiated with a unique constant of the according type. Collecting all messages from these strands we thus obtain an infinite set M'. However, $\mathit{SMP}(M) \supseteq \mathit{SMP}(M')$ since M' contains only well-typed instances of M, and thus if M is type-flaw resistant, so is M'. More generally, for checking that a protocol is type-flaw resistant, it is sufficient to consider any set M that subsumes all messages of the protocols' honest agent strands as well-typed instances.

It is not too difficult to show that M for TLS is type-flaw resistant (we have in fact formally proven it in Isabelle): every operator except prf is applied to arguments of the same type throughout $\mathit{SMP}(M)$; for prf the argument is either of the form $\mathsf{clientFinished}(\cdot)$ or $\mathsf{master}(\cdot)$ (but never a variable, because prf is always applied to non-variable arguments in M and it does not occur in the type of any term in M). Due to the free algebra, it follows almost immediately that two unifiable elements of $\mathit{SMP}(M) \setminus \mathcal{V}$ have the same type.

3.2.3 Constraint-level Typing Result & Formalization in Isabelle

For our typing result on the constraint-level we first prove that well-typedness and type-flaw resistance are invariants of the constraint reduction:

Lemma 3.18 (Invariants) *If $(\mathcal{A}, \theta)$ is well-formed, $\mathcal{A}$ is type-flaw resistant, θ is well-typed, and $(\mathcal{A}, \theta) \rightsquigarrow^* (\mathcal{A}', \theta')$, then $\mathcal{A}'$ is type-flaw resistant and θ' is well-typed.*

Proof. Follows by induction on the reflexive transitive closure of $\rightsquigarrow$ and case analysis on $\rightsquigarrow$. The proof idea is that all terms in the constraint reduction are elements of $\mathit{SMP}(\mathcal{A})$ and thus any unifier between non-variable terms must be well-typed and $\mathit{SMP}(\mathcal{A}') \subseteq \mathit{SMP}(\mathcal{A})$. Note that preservation of type-flaw resistance is actually the core of the typing result: in no step of the constraint reduction does the intruder need to do something ill-typed and he can instead choose something well-typed. □

Recall that by Lemma 3.12, every simple constraint has an interpretation. We now show that it even has a well-typed interpretation. This is because the intruder can generate terms of any type (as he knows constants of any type and can compose with public functions). The most difficult part to show here is that the inequalities also have a well-typed solution (that is also compatible with what the intruder can construct).

Lemma 3.19 (Simple constraints are well-typed satisfiable) *If $(\mathcal{A}, \theta)$ is well-formed, $\mathcal{A}$ is simple, and θ is well-typed, then there exists a well-typed interpretation $\mathcal{I}_\tau$ such that $\mathcal{I}_\tau \models_c (\mathcal{A}, \theta)$.*

Proof. Denote by X_{ineq} the free variables of the inequalities of $\mathcal{A}$. We first define two substitutions that will serve as the solutions to the inequalities of $\mathcal{A}$ and to the *Send* steps of $\mathcal{A}$:

- For the *Send* steps (which contain the messages that the intruder has to generate) we define the following substitution:

  $$\mathcal{I}_{send} \equiv \lambda\, v.\, \varepsilon\, t.\, \Gamma(v) = \Gamma(t) \wedge \emptyset \vdash_c t$$

  where ε is the Hilbert operator, i.e., $\varepsilon\, t.\ \phi$ yields a value t such that ϕ holds, and $\emptyset \vdash_c t$ means that the intruder can generate t without any prior knowledge except for the public constants.

- For the inequalities we obtain a substitution $\mathcal{I}_{ineq}$ satisfying the following:
  1. $dom(\mathcal{I}_{ineq}) = X_{ineq}$,
  2. $\mathcal{I}_{ineq}$ is fresh w.r.t. *subterms*(*trms*$_{st}$ $\mathcal{A}$)), and
  3. $\mathcal{I}_{ineq}$ is well-typed and each term in *subst-range* $\mathcal{I}_{ineq}$ is well-formed.

Both $\mathcal{I}_{send}$ and $\mathcal{I}_{ineq}$ exists and the proof is as follows:

1. We have already defined $\mathcal{I}_{send}$, but using the Hilbert operator. Hence what remains to be shown is that, given any variable v, we can always find a term t such that $\Gamma(v) = \Gamma(t)$ and $\emptyset \vdash_c t$. Recall that we have assumed that there exists infinitely many public constants of each atomic type, and that all function symbols with arity greater than zero are public. Terms of any well-formed type can therefore be derived using (*Compose*) alone, and this can be shown by induction.

2. For $\mathcal{I}_{ineq}$ we need a stronger argument since it has to be fresh. For that reason we first define a function *fresh-pgwt* that constructs a fresh term given a well-formed type τ and a finite set S of "used" symbols from Σ:

$$\begin{array}{lcl} \textit{fresh-pgwt}\ S\ (\textit{TAtom}\ a) & \equiv & \\ \qquad \textit{Fun}\ (\varepsilon\, c.\ c \notin S \wedge \Gamma(\textit{Fun}\ c\ []) = \textit{TAtom}\ a \wedge \textit{public}\ c)\ [] & & \\ \textit{fresh-pgwt}\ S\ (\textit{TComp}\ f\ T) & \equiv & \textit{Fun}\ f\ (\textit{map}\ (\textit{fresh-pgwt}\ S)\ T) \end{array}$$

   Given a set of terms M we denote by S_M the symbols from Σ that occurs in M. For finite M and well-formed terms t the term *fresh-pgwt* S_M $(\Gamma(t))$ then has the following properties:

   - *subterms*(*fresh-pgwt* S_M $(\Gamma(t))) \subseteq \{t \mid \emptyset \vdash_c t\}$,
   - *fresh-pgwt* S_M $(\Gamma(t))$ is well-formed,
   - $\Gamma($*fresh-pgwt* S_M $(\Gamma(t))) = \Gamma(t)$, and
   - no symbol $f \in S_M$ occurs in *fresh-pgwt* S_M $(\Gamma(t))$, implying that *subterms*(*fresh-pgwt* S_M $(\Gamma(t)))$ and *subterms*(M) are disjoint.

   Now let $S_{\mathcal{A}}$ be the symbols from Σ occurring in *trms*$_{st}$ $\mathcal{A}$. Note that $S_{\mathcal{A}}$ is finite. Since we want the domain of $\mathcal{I}_{ineq}$ to be exactly the free variables occurring in the inequalities of the simple constraint $\mathcal{A}$, which is a finite set, we can construct $\mathcal{I}_{ineq}$ inductively as follows:

   (a) If $X_{ineq} = \emptyset$ then we use the empty substitution *Var* for $\mathcal{I}_{ineq}$.

   (b) In the inductive case assume that $X_{ineq} = X'_{ineq} \cup \{x\}$ where $x \notin X'_{ineq}$. For the induction hypothesis assume that σ_{ineq} is a substitution that satisfy the properties we need and that $dom(\sigma_{ineq}) = X'_{ineq}$.

We now have to construct a substitution σ'_{ineq} with $dom(\sigma'_{ineq}) = X'_{ineq} \cup \{x\}$ and which also satisfy the properties we need for $\mathcal{I}_{ineq}$. The construction is as follows. First, let S_σ be the symbols from Σ that occurs in *subst-range* σ_{ineq}. Then define σ'_{ineq} as follows:

$$\sigma'_{ineq} \equiv \lambda v.\ \textbf{if}\ v = x\ \textbf{then}\ \textit{fresh-pgwt}\ (S_{\mathcal{A}} \cup S_\sigma)\ (\Gamma(x))\ \textbf{else}\ \sigma_{ineq}(v)$$

It follows from the properties of *fresh-pgwt* and σ_{ineq} that this substitution has the desired properties, thus concluding the case.

Define $\mathcal{I}_{simple}$ as the substitution $\theta \cdot \mathcal{I}_{ineq} \cdot \mathcal{I}_{send}$. We show in the following that $\mathcal{I}_{simple}$ is a well-typed model of $(\mathcal{A}, \theta)$.

First, note that θ is idempotent because of well-formedness, i.e., $\theta(\theta(t)) = \theta(t)$ for any term t. Hence θ supports $\mathcal{I}_{simple}$ by definition of substitution support and substitution composition.

Note also that a) $\mathcal{I}_{simple}$ is an interpretation because $\mathcal{I}_{send}$ maps every variable to a ground term, and b) $\mathcal{I}_{simple}$ is well-typed because all its component substitutions (i.e., θ, $\mathcal{I}_{ineq}$, and $\mathcal{I}_{send}$) are well-typed.

It remains to be shown that $[\![\emptyset; \mathcal{A}]\!]_c\ \mathcal{I}_{simple}$. Since $\theta(\mathcal{A}) = \mathcal{A}$ (this again follows from well-formedness) we can instead prove $[\![\emptyset; \mathcal{A}]\!]_c\ (\mathcal{I}_{ineq} \cdot \mathcal{I}_{send})$, and we do so by a case analysis on each step occurring in $\mathcal{A}$:

- Consider any intruder deduction constraint *Send* t of $\mathcal{A}$. We need to show that $M \vdash_c (\mathcal{I}_{ineq} \cdot \mathcal{I}_{send})(t)$ where M is the intruder knowledge available at the point in $\mathcal{A}$ where *Send* t occurs. Since all intruder deduction constraints in simple constraints are of the form *Send* x for some variable x all variables of the same type can safely be interpreted as the same public ground term. Hence $\emptyset \vdash_c (\mathcal{I}_{ineq} \cdot \mathcal{I}_{send})(x)$ and thus $\mathcal{I}_{ineq} \cdot \mathcal{I}_{send}$ is a solution to the *Send* steps of $\mathcal{A}$ because $\vdash_c$ is monotonic in the intruder knowledge.
- Now consider any *Inequality* $\bar{x}\ t\ t'$ occurring in $\mathcal{A}$. Note that this inequality is satisfiable because $\mathcal{A}$ is simple. Note also that $\mathcal{I}_{ineq}$ sends all free variables of *Inequality* $\bar{x}\ t\ t'$ to ground terms, and so it suffices to show that $\mathcal{I}_{ineq}$ is a model of *Inequality* $\bar{x}\ t\ t'$. From type-flaw resistance we know that the inequality constraint has one of two different forms:
  1. All free variables of the inequality have atomic types. Hence $\mathcal{I}_{ineq}$ sends all the free variables of the inequality to constants because $\mathcal{I}_{ineq}$ is well-typed.
  2. In the second case we know that there is no subterm of t and t' of the form $f(t_1, \ldots, t_n)$ where $n > 0$ and $t_1, \ldots, t_n \in \bar{x}$.

In both cases we can simply apply Lemma 3.6 to conclude the case.

Finally note that there are no constraints of the form *Equality* $t\ t'$ occurring in simple constraints and so we have covered all constraint steps of $\mathcal{A}$. Thus we have shown that $\mathcal{I}_\tau \models_c (\mathcal{A}, \theta)$ for some well-typed interpretation $\mathcal{I}_\tau$. □

From this we get the typing result on the constraint level:

Theorem 3.20 (Constraint-level typing result) *If* $(\mathcal{A}, \theta)$ *is well-formed and type-flaw resistant,* θ *is well-typed, and* $\mathcal{I} \models_c (\mathcal{A}, \theta)$*, then there exists a well-typed interpretation* $\mathcal{I}_\tau$ *such that* $\mathcal{I}_\tau \models_c (\mathcal{A}, \theta)$*.*

Proof. The proof is by a simple lifting argument using the previously established results. In a nutshell we can reduce $(\mathcal{A}, \theta)$ to a simple, type-flaw resistant, and satisfiable constraint using the constraint reduction $\rightsquigarrow$. This reduced constraint then has some well-typed model which must also be a model of $(\mathcal{A}, \theta)$ by correctness of $\rightsquigarrow$. □

3.3 Protocol Transition Systems

The previous sections have established the typing result on the level of constraints and we now lift it to transition systems. Since we had out-sourced the entire question of analysis, we also have to take care of it now.

3.3.1 Definitions

We now represent also the honest agents by strands (reusing the definition of intruder strands), and we define a protocol to be a countably infinite set of such honest agent strands:

$$\textbf{type-synonym}\ (\Sigma, \mathcal{V})\ \textit{protocol} \equiv (\Sigma, \mathcal{V})\ \textit{strand set}$$

As is usual we allow the intruder full control of all communication happening in the protocol: whenever an honest agent receives a message the intruder must have sent it, and whenever an honest agent sends a message the intruder intercepts it. Hence, for protocol execution, we define a symbolic transition system in which honest agents can send and receive messages (that might contain variables, hence *symbolic*) and where we record the steps taken during these transitions. A *state* $(\mathcal{P};\mathcal{A})$ then consists of a protocol $\mathcal{P}$ and an intruder strand $\mathcal{A}$ which represents the steps taken from the intruder's point of view and which we will build up during execution of $\mathcal{P}$. Since the goal of this section is to lift the typing result to the transition system, where we use the full semantics $\models$, we interpret the constraints in states under $\models$ and *not* $\models_c$ as we did in previous sections. For the *initial state* the intruder strand is empty, that is $(\mathcal{P}_0; 0)$ where $\mathcal{P}_0$ denotes the initial protocol and 0 the empty intruder strand.

Definition 3.21 (Protocol transition system)

$$
\begin{aligned}
&TS_1 : (\mathcal{P};\mathcal{A}) \Rightarrow^\bullet (\mathcal{P}\setminus\{0\};\mathcal{A}) \text{ if } 0 \in \mathcal{P}\\
&TS_2 : (\mathcal{P};\mathcal{A}) \Rightarrow^\bullet (\{\mathcal{S}\}\cup(\mathcal{P}\setminus\{\mathit{Send}\ t.\mathcal{S}\});\mathcal{A}.\mathit{Receive}\ t) \text{ if } \mathit{Send}\ t.\mathcal{S}\in\mathcal{P}\\
&TS_3 : (\mathcal{P};\mathcal{A}) \Rightarrow^\bullet (\{\mathcal{S}\}\cup(\mathcal{P}\setminus\{\mathit{Receive}\ t.\mathcal{S}\});\mathcal{A}.\mathit{Send}\ t) \text{ if } \mathit{Receive}\ t.\mathcal{S}\in\mathcal{P}\\
&TS_4 : (\mathcal{P};\mathcal{A}) \Rightarrow^\bullet (\{\mathcal{S}\}\cup(\mathcal{P}\setminus\{\mathit{Equality}\ t\ t'.\mathcal{S}\});\mathcal{A}.\mathit{Equality}\ t\ t')\\
&\hspace{20em}\text{if } \mathit{Equality}\ t\ t'.\mathcal{S}\in\mathcal{P}\\
&TS_5 : (\mathcal{P};\mathcal{A}) \Rightarrow^\bullet (\{\mathcal{S}\}\cup(\mathcal{P}\setminus\{\mathit{Inequality}\ \bar{x}\ t\ t'.\mathcal{S}\});\mathcal{A}.\mathit{Inequality}\ \bar{x}\ t\ t')\\
&\hspace{20em}\text{if } \mathit{Inequality}\ \bar{x}\ t\ t'.\mathcal{S}\in\mathcal{P}
\end{aligned}
$$

The first rule simply removes empty strands, i.e., honest agents that have finished execution. The second rule allows honest agents to send messages, in which case the intruder intercepts and receives this message. Hence we extend the intruder knowledge (by adding a *Receive* step to the intruder strand) at that point with the message that is sent. The third rule allows an honest agent to receive a message, and in this case we require that the intruder must generate this message. Thus we extend the intruder strand with an additional derivation requirement by adding a *Send* step. The two remaining rules simply record the positive and negative message checks made by honest agents. As usual we write $\Rightarrow^{\bullet *}$ for the reflexive transitive closure.

Note that we require intruder strands to be well-formed, including those emerging from an execution of a protocol. For this reason, we impose a requirement on the variables in all honest-agent strands of the protocols we consider that is dual to the requirement for intruder strands: while in the intruder strands all variables must originate in a *Send* step, we require that in an honest agent strand they are all originating in a *Receive* step. Moreover, the bound and free variables occurring in a protocol must also be disjoint. Formally, we first define the *dual* of a strand $\mathcal{S}$ as "swapping" the direction of the steps of $\mathcal{S}$:

$$
\begin{aligned}
\mathit{dual}_{st}\ 0 &= 0\\
\mathit{dual}_{st}\ (\mathit{Send}\ t.\mathcal{S}) &= \mathit{Receive}\ t.(\mathit{dual}_{st}\ \mathcal{S})\\
\mathit{dual}_{st}\ (\mathit{Receive}\ t.\mathcal{S}) &= \mathit{Send}\ t.(\mathit{dual}_{st}\ \mathcal{S})\\
\mathit{dual}_{st}\ (\mathfrak{s}.\mathcal{S}) &= \mathfrak{s}.(\mathit{dual}_{st}\ \mathcal{S}) \text{ otherwise}
\end{aligned}
$$

Then we define protocol well-formedness using Definition 3.3:

Definition 3.22 A protocol $\mathcal{P}$ is *well-formed*, written $wf_{sts}\ \mathcal{P}$, where wf_{sts} is defined as follows:

$$wf_{sts}\ \mathcal{P} \equiv \forall \mathcal{S} \in \mathcal{P}.\ wf_{st}\ \emptyset\ (dual_{st}\ \mathcal{S}) \wedge (\forall \mathcal{S} \in \mathcal{P}.\ \forall \mathcal{S}' \in \mathcal{P}.\ fv(\mathcal{S}) \cap bvars(\mathcal{S}') = \emptyset)$$

where $bvars(\mathcal{S})$ are the bound variables of $\mathcal{S}$ (i.e., $x \in bvars(\mathcal{S})$ iff there exists $\bar{x}$, t, and t' such that *Inequality* $\bar{x}\ t\ t'$ occurs in $\mathcal{S}$ and $x \in \bar{x}$).

It is now immediate that all intruder strands of reachable states are well-formed if the initial protocol is well-formed.

3.3.2 Problems of [AMMV15]

Recall that in our Isabelle formalization of the lazy intruder, we have decided to "out-source" the analysis step from the intruder to the transition system. Therefore, we need to now show that the transition system from the previous section (that assumes the full intruder in its semantics) is equivalent to a transition system where the intruder is restricted to composition steps (i.e., $\vdash_c$) and that has special transition steps for analysis—and make that work with the typing result. Upon trying to prove these results in Isabelle we discovered several problems in the result of Almousa et al. [AMMV15]. In fact, that paper handles analysis as part of the lazy intruder, but the problems appear in a similar form. In fact, discovering and provably fixing all such mistakes is indeed the main goal of the Isabelle formalization of this chapter. We discuss first the errors and ways to fix them, and then how other typing results are doing on these issues.

The lazy intruder analysis rule of [AMMV15] would in our notation look like this:

$$(Decompose_{LI})\quad (\mathcal{A}.\mathcal{A}', \theta) \rightsquigarrow (\mathcal{A}.Send\ K.Receive\ T.\mathcal{A}', \theta)$$
$$\text{if } s \in ik_{st}\ \mathcal{A}, \mathsf{Ana}\ s = (K, T), T \not\subseteq ik_{st}\ \mathcal{A}$$

Here we use *Send* K and *Receive* T for sets K and T of messages as obvious abbreviation for sequences of send and receive steps. The rule means, at any point in an intruder strand, the intruder can attempt the analysis of a term s that he learned before that point, and this attempt would mean that he has to generate ("Send") the key terms K and would obtain ("Receive") the resulting messages T. (In fact, our handling of analysis as part of the transition system adds analysis steps that similarly produce such sending and receiving steps in the intruder strand.)

Like [AMMV15], we make the following requirement on the Ana function that the resulting terms are subterms of the term being analyzed and that no new variables are introduced:

$$\mathsf{Ana}_0\colon \mathsf{Ana}\ t = (K, T) \Longrightarrow T \subset \mathit{subterms}\ t$$

In [AMMV15] the keys K also need to be subterms of the term t being analyzed. We have made a slight generalization here and do not require the keys to be subterms of the term being analyzed. This is only because it allows us to model private keys using a function inv from public to private keys (as we do in later chapters) and it is not because it is needed to fix any mistakes in [AMMV15]. We do need to ensure that analysis does not introduce new variables in keys and that the set of keys required for analysis is always finite and well-formed:

$$\mathsf{Ana}_1\colon \mathsf{Ana}\ t = (K, T) \Longrightarrow \mathit{fv}(K) \subseteq \mathit{fv}(t) \land \mathit{finite}\ K \land \mathit{wf}_{\mathit{trms}}\ K$$

Variables should furthermore not be analyzable:

$$\mathsf{Ana}_2\colon \mathsf{Ana}\ (\mathit{Var}\ x) = (\emptyset, \emptyset)$$

The Ana_0 requirement is necessary for termination, since without such a restriction to subterms one could encode undecidable problems into analysis. This is however not enough as our first counter-example to correctness of [AMMV15] shows:

Example 3.4 *Suppose for two public unary operators f and g we define:* $\mathsf{Ana}\ f(g(x)) = (\emptyset, \{x\})$. *Then the constraint Receive $g(c)$.Send c has a solution since $\{g(c)\} \vdash c$. This solution would however be missed by ($\mathit{Decompose}_{LI}$), thus the lazy intruder of [AMMV15] is incomplete.* □

The same problem does not occur in other typing results [AD14, CD09] because they consider a fixed set of operators where none has a destructor-like behavior upon analysis. A property of analysis that all these approaches use is that the intruder does not learn anything new from analyzing terms that he composed himself, e.g., encrypting a term and then decrypting it will not reveal new information, and without loss of generality we thus can exclude intruder-composed terms from analysis. Also [AMMV15] uses this argument, but as Example 3.4 shows, this is not true for all intruder theories they allow. We thus make an additional restriction on Ana, namely that analysis can only yield *direct* subterms:

$$\mathsf{Ana}_3\colon \mathsf{Ana}\ f(t_1, \ldots, t_n) = (K, T) \Longrightarrow T \subseteq \{t_1, \ldots, t_n\}$$

Example 3.5 *Let now f be a binary operator with the following* Ana *rule:*

$$\mathsf{Ana}\ f(s,t) = (\emptyset, \{t\})\ \textit{if}\ s \in \mathcal{V}$$

This is hardly a reasonable analysis rule since it gives results only for symbolic terms, but not for ground terms. The constraint Send x.Receive $f(x,c)$.Send c has no solution since there is no interpretation $\mathcal{I}$ with $\mathcal{I}(\{f(x,c)\}) \vdash c$. However, constraint reduction with rule $(Decompose_{LI})$ yields a simple (and thus satisfiable) constraint, and we thus also have a counter-example for soundness of [AMMV15]. □

To correct this, we add the following requirement:

$$\mathsf{Ana}_4 : \mathsf{Ana}\ t = (K,T) \neq (\emptyset,\emptyset) \Longrightarrow \mathsf{Ana}\ \delta(t) = (\delta(K), \delta(T)) \\ \text{for any substitution } \delta$$

Thus, when applying Ana on any analyzable term t, then any instance $\delta(t)$ must allow for the same analysis under δ.

Example 3.6 *Consider again our standard* Ana *from Example 2.1 (which satisfies all four requirements). For the full intruder model $\vdash$ (that is not restricted to composition only) the lazy intruder with $(Decompose_{LI})$ is not complete: The constraint Send x.Receive $\mathsf{crypt}(x,\mathsf{c})$.Send c has the solution $\mathcal{I} = [x \mapsto \mathsf{pub}(\mathsf{c}')]$ for some constant c'. However, this solution is not found by the lazy intruder (with the above analysis rule) because* Ana $\mathsf{crypt}(x,\mathsf{c}) = (\emptyset,\emptyset)$. *The problem is that the case* Ana $\mathsf{crypt}(\mathsf{pub}(k), m)$ *does not* match *the term we need to analyze, since it has the variable x in the key position (and this is actually what the authors of [AMMV15] meant to do). As our example shows, however, one needs to actually apply this rule under* unification *with a term in the intruder knowledge, but that would require to apply the unifier (in the example $[x \mapsto \mathsf{pub}(x')]$ for a new x') to the rest of the constraint—while all other rules of [AMMV15] explicitly denote such unifiers; moreover this reading of the rule would lead to non-termination.* □

In the other typing results like [AD14] (and also in [CD09]), this problem does not occur because they fix the public-key infrastructure, i.e., they cannot model that an honest agent receives an arbitrary public key x in a message and use it for encrypting a message, i.e., $\mathsf{crypt}(x,m)$. When fixing the public key infrastructure, all keys used for public key encryption are of the form $\mathsf{pub}(\cdot)$ (in our notation) and then the mentioned problem does not occur. However, we do not want to impose this strong restriction to a fixed public key infrastructure and rather allow for protocols that can also exchange public keys. A milder restriction is that all terms used as a first argument of crypt must have the form $\mathsf{pub}(\cdot)$, for instance the strand of an honest agent could be: *Receive* $\mathsf{pub}(x)$.*Send* $\mathsf{crypt}(\mathsf{pub}(x), \mathsf{c})$. This implies the restriction that this agent only accepts a public key as input, i.e., restricting this bit to a typed model by assumption. There are several ways to justify this restriction, e.g., it is common in protocols where a new public key t can be introduced that the creator has to sign any message with the corresponding private key, proving that $t = \mathsf{pub}(s)$ for some private key s (and without the recipient learning s). Also, when receiving a public key as part of a certificate from a trusted authority, one may rely that the authority has required this kind of proof from the owner of the public key, and thus it is justifiable to model the certified key to have the form $\mathsf{pub}(\cdot)$. In a later chapter we give an alternative solution using a private function inv.

While the pub-requirement solves the problem for the concrete example crypt, we need a general requirement for arbitrary operators. The example shows that this cannot be a property of Ana alone, but relates to the use of the operators in the protocol. Essentially, any message of the protocol that the intruder can potentially analyze during an attack should have a similar requirement. This includes the subterms of messages sent by honest agents, but that is not sufficient. Recall that each variable in the protocol represents either a choice made by the intruder or an abbreviation of something else (namely a term occurring at the right-hand side of an equality constraint). The variables that represent intruder choices can be safely ignored. The other variables, however, are abbreviations of terms occurring at the right-hand side of positive message checks and these abbreviations need to be taken into account. We therefore define a function $\mathit{eqs\text{-}rhs}_{st}$ that collects the terms occurring at the right-hand side of positive checks:

Definition 3.23 Let $\mathcal{A}$ be a constraint. Then the *set of terms occurring at the right-hand side of equalities of* $\mathcal{A}$, $\mathit{eqs\text{-}rhs}_{st}\ \mathcal{A}$, is defined as follows:

$$\begin{aligned}
\mathit{eqs\text{-}rhs}_{st}\ 0 &= \emptyset \\
\mathit{eqs\text{-}rhs}_{st}\ (\mathit{Equality}\ t\ t'.\mathcal{A}) &= \{t'\} \cup \mathit{eqs\text{-}rhs}_{st}\ \mathcal{A} \\
\mathit{eqs\text{-}rhs}_{st}\ (\mathfrak{a}.\mathcal{A}) &= \mathit{eqs\text{-}rhs}_{st}\ \mathcal{A} \text{ otherwise}
\end{aligned}$$

The requirement on protocols is then as follows. Like with type-flaw resistance we first define our requirement on sets of terms and then use this definition to define a requirement for protocols and constraints:

Definition 3.24 A set of terms M is *analysis-invariant* iff

$$\forall t \in (\mathit{subterms}\ M) \setminus \mathcal{V}.\ \forall K, T, \delta.\ \mathsf{Ana}\ t = (K, T) \longrightarrow \mathsf{Ana}\ \delta(t) = (\delta(K), \delta(T))$$

A protocol $\mathcal{P}_0$ is then analysis-invariant iff the set containing the messages of $\mathcal{P}_0$ that are either sent by honest agents or occur at the right-hand side of positive checks is analysis-invariant, i.e., iff the set

$$\Big(\bigcup ik_{st} \mathbin{`} (dual_{st} \mathbin{`} \mathcal{P}_0)\Big) \cup \Big(\bigcup eqs\text{-}rhs_{st} \mathbin{`} \mathcal{P}_0\Big)$$

is analysis-invariant.

Similarly, a constraint $\mathcal{A}$ is analysis-invariant iff the set $ik_{st}\ \mathcal{A} \cup eqs\text{-}rhs_{st}\ \mathcal{A}$ is analysis-invariant.

Thus we require that any subterm t of the protocol, except variables, can be analyzed if some instance $\delta(t)$ can be analyzed. This excludes a term like $\mathsf{crypt}(x, t)$ since it cannot be analyzed while the instance $\mathsf{crypt}(\mathsf{pub}(c), t)$ can. In general, this restriction affects only those operators f where the analysis rule has the form $\mathsf{Ana}\ f(t_1, \ldots, t_n)$ where some t_i is not a pattern variable; then the protocol cannot use a variable for that argument.

These restrictions are sufficient to conclude the typing result on the transition system level, as described next, and they still support strictly more protocols than the previous typing results (except the flawed [AMMV15]).

3.3.3 Handling Analysis

As an intermediate step towards the result, we now define a second transition system $\Rightarrow^{\bullet}_{c}$ similar to $\Rightarrow^{\bullet}$, but where the intruder does not handle analysis himself (interpreting constraints under $\models_c$ instead of the full $\models$ as in $\Rightarrow^{\bullet}$) and where we have special transitions for analysis. An easy way to handle this would be to simply define a set of honest agents that behave like the analysis functionality, e.g.,

$$\mathit{Receive}\ \mathsf{crypt}(\mathsf{pub}(x), y).\mathit{Receive}\ x.\mathit{Send}\ y$$

(This is similar to the penetrator strands of Thayer et al. [THG99].)

In fact, this works fine (and does not even need the requirement of analysis invariance we introduced before) as far as the equivalence to the standard transition system $\Rightarrow^{\bullet}$ is concerned. However this does not directly work with the typing result: the notion of type-flaw resistance would have to be satisfied on the set of all honest agent strands, including the ones for analysis. This would be violated for many reasonable protocols (that have no type-flaw problems).

Luckily, there is a (more complicated) solution that requires no further restriction on protocols.

The idea is that the intruder is allowed to attempt analysis for every non-variable subterm of a term in his knowledge. Note that this includes subterms he may be unable to derive, but as part of the analysis step he has to prove he can produce them, so this is sound. Note also that some variables in the intruder knowledge may not directly denote choices made by the intruder, because the well-formedness requirements allow variables to be introduced at the left-hand side of positive message checks and they can then later occur in messages transmitted by honest agents. Such variables would instead be abbreviations of terms occurring at the right-hand side of the checks and so we need to take this into account. Thus the protocol transition system $\Rightarrow^{\bullet}_{c}$ is defined like $\Rightarrow^{\bullet}$ plus the following additional rule:[5]

$$\begin{aligned} TS^{\mathsf{c}}_{6}:\quad & (\mathcal{P};\mathcal{A}) \Rightarrow^{\bullet}_{c} (\mathcal{P};\mathcal{A}.\mathit{Send}\ t.\mathit{Send}\ k_1.\cdots.\mathit{Send}\ k_m.\\ & \qquad\qquad\qquad \mathit{Receive}\ s_1.\cdots.\mathit{Receive}\ s_n)\\ & \text{where } t \in \mathit{subterms}(\mathit{ik}_{st}\ \mathcal{A} \cup \mathit{eqs\text{-}rhs}_{st}\ \mathcal{A}) \setminus \mathcal{V}\\ & \text{and } \mathsf{Ana}\ t = (\{k_1,\dots,k_m\},\{s_1,\dots,s_n\}) \end{aligned}$$

Example 3.7 *Consider the protocol $\mathcal{P}_0 = \{\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3\}$ where*

$$\begin{aligned} \mathcal{S}_1 &= \mathit{Send}\ \mathsf{k_a}.\mathit{Send}\ \mathsf{scrypt}(\mathsf{k_b},\mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}),\mathsf{secret}))\\ \mathcal{S}_2 &= \mathit{Receive}\ \mathsf{scrypt}(\mathsf{k_b},x).\mathit{Send}\ x\\ \mathcal{S}_3 &= \mathit{Receive}\ \mathsf{secret} \end{aligned}$$

Here, the strand $\mathcal{S}_3$ represents a strand to check a secrecy goal, i.e., we want to check that we cannot reach a state where $\mathcal{S}_3$ has executed and the intruder constraint is satisfiable. Consider the execution of the steps of $\mathcal{S}_1$ and $\mathcal{S}_2$, i.e., $(\mathcal{P}_0;0) \Rightarrow^{\bullet *}_{c} (\{\mathcal{S}_3\};\mathcal{A})$ where

$$\begin{aligned} \mathcal{A} = {} & \mathit{Receive}\ \mathsf{k_a}.\mathit{Receive}\ \mathsf{scrypt}(\mathsf{k_b},\mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}),\mathsf{secret})).\\ & \mathit{Send}\ \mathsf{scrypt}(\mathsf{k_b},x).\mathit{Receive}\ x \end{aligned}$$

For the intruder to obtain the secret, we can now make an analysis step with rule (TS^{c}_{6}) for the term $t = \mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}),\mathsf{secret})$, yielding the state $(\{\mathcal{S}_3\};\mathcal{A}.\mathcal{D})$ with

$$\mathcal{D} = \mathit{Send}\ \mathsf{crypt}(\mathsf{pub}(\mathsf{k_a}),\mathsf{secret}).\mathit{Send}\ \mathsf{k_a}.\mathit{Receive}\ \mathsf{secret}$$

and then execute $\mathcal{S}_3$, yielding state $(\emptyset;\mathcal{A}.\mathcal{D}.\mathcal{A}')$ with $\mathcal{A}' = \mathit{Send}\ \mathsf{secret}$. This constraint $\mathcal{A}.\mathcal{D}.\mathcal{A}'$ is satisfiable in $\models_c$.

[5] Note that repeated application of TS^{c}_{6} to the same subterm does not change the meaning of the intruder constraint in the protocol transition system (because the intruder would then send and receive the exact same messages multiple times) and we can therefore exclude that. Hence we can make the protocol transition system finite for finite protocols (i.e., a bounded number of protocol sessions).

In the standard transition system the corresponding state would just omit the analysis step, i.e., $(\mathcal{P}_0; 0) \Rightarrow^{\bullet *} (\emptyset; \mathcal{A}.\mathcal{A}')$. *This constraint* $\mathcal{A}.\mathcal{A}'$ *is satisfiable for the full intruder* $\models$. □

More generally, we prove in Isabelle that the two transition systems are equivalent. In particular, for every reachable state $(\mathcal{P}; \mathcal{A})$ of $\Rightarrow^{\bullet}$ and every solution $\mathcal{I} \models \mathcal{A}$, an equivalent state $(\mathcal{P}; \mathcal{A}')$ of $\Rightarrow^{\bullet}_{c}$ is reachable where $\mathcal{A}'$ is like $\mathcal{A}$ augmented with analysis steps, and $\mathcal{I} \models_c \mathcal{A}'$. From the initial state $(\mathcal{P}_0; 0)$ this can be stated as follows:

Lemma 3.25 (transition systems equivalence, part 1) *If protocol* $\mathcal{P}_0$ *is well-formed and analysis-invariant,* $(\mathcal{P}_0; 0) \Rightarrow^{\bullet *} (\mathcal{P}; \mathcal{A}_1. \cdots .\mathcal{A}_n)$, *and* $\mathcal{I} \models \mathcal{A}_1. \cdots .\mathcal{A}_n$ *where each* $\mathcal{A}_i$ *emerged from exactly one application of* (TS_1), (TS_2), (TS_3), (TS_4), *or* (TS_5), *then there exists* $\mathcal{D}_1, \ldots, \mathcal{D}_{n-1}$ *such that*

$$(\mathcal{P}_0; 0) \Rightarrow^{\bullet *}_{c} (\mathcal{P}; \mathcal{A}_1.\mathcal{D}_1. \cdots .\mathcal{A}_{n-1}.\mathcal{D}_{n-1}.\mathcal{A}_n)$$

and

$$\mathcal{I} \models_c \mathcal{A}_1.\mathcal{D}_1. \cdots .\mathcal{A}_{n-1}.\mathcal{D}_{n-1}\mathcal{A}_n$$

and where each $\mathcal{D}_i$ *emerged from zero or more applications of* (TS^c_6).

The most complicated aspect of the proof is to show that, if $\mathcal{I}(ik_{st}\ \mathcal{A}) \vdash \mathcal{I}(t)$ for some intruder strand $\mathcal{A}$ with model $\mathcal{I}$ and some term t, then there exists some sequence of (TS^c_6) steps $\mathcal{D}$ such that $\mathcal{I}(ik_{st}\ (\mathcal{A}.\mathcal{D})) \vdash_c \mathcal{I}(t)$ and where $\mathcal{I}$ is also a model of $\mathcal{A}.\mathcal{D}$. This proof proceeds by an induction on the derivation of $\mathcal{I}(t)$. Not surprisingly, this case bears many similarities to completeness proofs of lazy intruder constraint reduction systems like [AMMV15, CD09] where the intruder can analyze terms. The most complicated case—both in our proof and the proofs of completeness—is where the last step of the derivation is an application of the (*Decompose*) rule, i.e., where $\mathcal{I}(t)$ is derived by analyzing another ground term t'. In the completeness proofs we would in this case have to inspect the derivation tree for t', eliminate redundant parts (namely, analysis of intruder-composed terms), and, in the case where the last step in the derivation is yet another application of (*Decompose*), regress to a point in the derivation tree for t' where no (*Decompose*) has occurred yet. In our setting, because we have a clear separation between term analysis and composition (because of the outsourcing of analysis and by considering the sub-relation $\vdash_c$ of $\vdash$), we immediately get from the induction hypothesis that there exists some $\mathcal{D}'$ (where $\mathcal{I}$ is still a model of $\mathcal{A}.\mathcal{D}'$) such that $\mathcal{I}(ik_{st}\ (\mathcal{A}.\mathcal{D}')) \vdash_c t'$. Hence we essentially perform the regression by simply applying the induction hypothesis instead of inspecting and transforming derivation trees, making the proof slightly easier.

The other direction of the equivalence is more straightforward and does not require any assumptions on the protocol. It follows easily by an induction on reachability:

Lemma 3.26 (Transition systems equivalence, part 2) *If*

$$(\mathcal{P}_0; 0) \Rightarrow_c^{\bullet *} (\mathcal{P}; \mathcal{A}_1.\mathcal{D}_1.\cdots.\mathcal{A}_{n-1}.\mathcal{D}_{n-1}.\mathcal{A}_n)$$

and

$$\mathcal{I} \models_c \mathcal{A}_1.\mathcal{D}_1.\cdots.\mathcal{A}_{n-1}.\mathcal{D}_{n-1}.\mathcal{A}_n$$

where each $\mathcal{A}_i$ emerged from exactly one application of (TS_1), (TS_2), (TS_3), (TS_4), or (TS_5), and where each $\mathcal{D}_i$ emerged from zero or more applications of (TS_6^c), then

$$(\mathcal{P}_0; 0) \Rightarrow^{\bullet *} (\mathcal{P}; \mathcal{A}_1.\cdots.\mathcal{A}_n) \text{ and } \mathcal{I} \models \mathcal{A}_1.\cdots.\mathcal{A}_n$$

3.3.4 Lifting the Typing Result

With this equivalence between the transition systems proven, we can now lift the typing result of Theorem 3.20 to constraints reachable in $\Rightarrow^\bullet$ where these constraints are interpreted under the full intruder $\models$ instead of $\models_c$. First we define that an entire protocol $\mathcal{P}_0$ (a set of strands for honest agents) is type-flaw resistant if the set of all messages of $\mathcal{P}_0$ is type-flaw resistant and if its strands satisfy tfr_{stp}. Written formally, we require the following two conditions:

$$\mathit{tfr}_{set}\left(\bigcup(\mathit{trms}_{st} \mathbin{`} \mathcal{P}_0)\right) \text{ and } \forall \mathcal{S} \in \mathcal{P}_0.\ \mathit{list\text{-}all}\ \mathit{tfr}_{stp}\ \mathcal{S}$$

It is now immediate that all intruder strands reachable from $(\mathcal{P}_0; 0)$ in both transition systems we defined (including analysis steps) are also type-flaw resistant, because the set of sub-message patterns are closed under subterms and term analysis.

We can now first apply Lemma 3.25 to any satisfiable state $(\mathcal{P}; \mathcal{A}_1.\cdots.\mathcal{A}_n)$ reachable in $\Rightarrow^\bullet$ to obtain an equivalent state $(\mathcal{P}; \mathcal{A}_1.\mathcal{D}_1.\cdots.\mathcal{A}_{n-1}.\mathcal{D}_{n-1}.\mathcal{A}_n)$ with the same solution reachable in our intermediate transition system $\Rightarrow_c^\bullet$. Then we can lift the typing result from the constraint level to $\Rightarrow_c^\bullet$, since here constraints are interpreted in $\models_c$, i.e., solving the constraints does not require analysis steps and thus our constraint-level typing result Theorem 3.20 applies. Then, by the equivalence to $\Rightarrow^\bullet$ with the full intruder model $\models$, i.e., Lemma 3.26, we obtain our main result that every reachable state of a type-flaw resistant and analysis-invariant protocol has a solution iff it has a well-typed one:

Theorem 3.27 *If protocol $\mathcal{P}_0$ is well-formed, type-flaw resistant and analysis-invariant, $(\mathcal{P}_0; 0) \Rightarrow^{\bullet *} (\mathcal{P}; \mathcal{A})$, and $\mathcal{I} \models \mathcal{A}$, then there exists a well-typed interpretation $\mathcal{I}_\tau$ such that $\mathcal{I}_\tau \models \mathcal{A}$.*

As a consequence of Theorem 3.27 we can also find well-typed solutions to constraints interpreted in $\models$. This is because $dual_{st}$ is its own inverse, i.e., $dual_{st}\ (dual_{st}\ \mathcal{A}) = \mathcal{A}$, and so any constraint $\mathcal{A}$ is reachable in $\Rightarrow^{\bullet}$ from the initial protocol $\{dual_{st}\ \mathcal{A}\}$, i.e., $(\{dual_{st}\ \mathcal{A}\}; 0) \Rightarrow^{\bullet *} (\emptyset; \mathcal{A})$.

Corollary 3.28 *If the constraint $\mathcal{A}$ is well-formed, type-flaw resistant and analysis-invariant, and $\mathcal{I} \models \mathcal{A}$, then there exists a well-typed interpretation $\mathcal{I}_\tau$ such that $\mathcal{I}_\tau \models \mathcal{A}$.*

3.4 Summary and Related Work

Over the past years, several typing results have emerged for security protocols, gradually extending the class of protocols that can be supported, in particular [HLS03, AD14, AMMV15]. A common idea for proving such typing results is to use a notion of symbolic constraints to represent executions (in particular attacks) and show that whenever there is a solution then there is a well-typed one. The requirement that the protocols have to fulfill for such a result is only that all messages of different intended type have sufficiently different structure to never be confused.

In the present chapter we established such a typing result in Isabelle: Given an Isabelle proof of security of a protocol where the intruder is limited to well-typed messages (e.g., like proofs in the works of [Pau98] and [Bel07]), then the typing result allows us to lift this proof to an intruder model without the restriction to well-typed messages. As an example, we have proven that the type-flaw resistance requirement of our result is indeed satisfied by the TLS protocol.

The particular value of this is the high reliability of proofs checked with Isabelle, in contrast to pen-and-paper proofs in partially natural language. This is illustrated by several errors we discovered in the pen-and-paper proofs of Almousa et al. [AMMV15]. Strictly speaking, their result does not hold without further restrictions on the supported operators and protocols. The complexity of such results (as well as verification tools) makes such mistakes likely and this bears the risk of accepting false security proofs. The Isabelle proof of the typing result under some additional restrictions is thus also a step towards "cleaning up".

In other typing results, such as Arapinis and Duflot [AD14], the problems of [AMMV15] do not arise, since they have fixed public key infrastructures (and fixed sets of supported operators). Our restrictions in contrast are so liberal that we do allow also for protocols where public keys are exchanged, though one must ensure or assume that the received terms are indeed public keys, but this is often in our opinion a realistic restriction—in the next chapter we give slightly different conditions for the analysis interface Ana which does not require us to impose a typing assumption on public keys.

At the core of our typing result is the formalization of the lazy intruder constraint reduction system and its correctness. This, as well as the proving of the typing result, gives insights for modeling and proving protocols in general: Since Isabelle forces one to be precise about every single detail, one is compelled to abstract, generalize, and simplify as far as possible, to reduce the formalization to the absolute essence. We have simplified the constraint representation and shown how to "out-source" the analysis steps of the intruder to steps in the protocol transition system. We believe that such insights are helpful beyond the result itself.

Chapter 4

Extending the Typing Result to Stateful Protocols

The previous chapter established a typing result in Isabelle/HOL. Until now this typing result only supports "stateless" protocols that can only maintain local state for single sessions. A more interesting and general class of protocols is one in which agents can additionally manipulate a global state spanning multiple sessions. Such protocols we call *stateful*. In stateful protocols updating the global state during one session might influence other running sessions, and the behavior of protocol participants is therefore also dependent on the global state. In this chapter we extend the typing result of the previous chapter to stateful protocols.

The chapter is organized as follows:

- In Section 4.1 we state the requirements on the analysis interfaces Ana that we consider in the rest of the thesis.
- In Section 4.2 we introduce a new strand-based protocol model for stateful protocols.
- In Section 4.3 we extend intruder constraints with set operations and define a reduction mechanism on constraints that we prove sound and complete.

- We prove our main theorem, the typing result, in Section 4.4.
- Finally, we present case studies and connections to other formalisms in Section 4.5 and 4.6.

4.1 Changes to the Analysis Interface

Compared to Chapter 3 there is one small change that we make to the requirements of the analysis interface Ana and a corresponding simplification: In Chapter 3 the rule Ana_4 is stated only for terms that do not yield $(\emptyset, \emptyset)$ while in the remaining chapters we assume a stronger variant of Ana_4. The version of Ana_4 used in Chapter 3 is necessary when modeling public-key encryption with a function pub from private to public keys, but it also leads to complications when proving a typing result, namely the need for analysis-invariance (Definition 3.24). Since we can model private keys using a function inv from public to private keys instead, we have decided to assume a stronger version of Ana_4 (named Ana_4' below) that is also stated for terms that yield $(\emptyset, \emptyset)$. This simplifies the typing result, since all constraints and protocols we consider are now guaranteed to be analysis-invariant, and so we have decided to stick with Ana_4' in all remaining chapters.

The analysis interfaces we consider in this chapter will therefore be subject to the following restrictions:

Ana_1: $\mathsf{Ana}(t) = (K, T)$ implies that K is finite, $fv(K) \subseteq fv(t)$, and that all keys in K are well-formed if t is well-formed.

Ana_2: $\mathsf{Ana}(x) = (\emptyset, \emptyset)$ for variables $x \in \mathcal{V}$.

Ana_3: $\mathsf{Ana}(f(t_1, \dots, t_n)) = (K, T)$ implies that $T \subseteq \{t_1, \dots, t_n\}$.

Ana_4': $\mathsf{Ana}(f(t_1, \dots, t_n)) = (K, T)$ implies $\mathsf{Ana}(\delta(f(t_1, \dots, t_n))) = (\delta(K), \delta(T))$.

Recall that Ana must be defined for arbitrary terms, including terms with variables (while the standard Dolev-Yao deduction is typically applied to ground terms). The four conditions regulate that Ana is also meaningful on symbolic terms. The first requirement, Ana_1, restricts the set of keys K to be finite and to not introduce any new variables, but the keys are otherwise independent of the term being decomposed. This is useful when modeling asymmetric decryption as we can then require the intruder to derive the inverse key $\mathsf{inv}(k)$ of the key k used for the encryption. The second requirement, Ana_2, ensures that variables cannot be decomposed. The third requirement Ana_3 states that all terms in T must be immediate subterms of the term being decomposed, and so the intruder cannot obtain any new terms by decomposing something that he composed himself. This is a technical requirement that is crucial in proofs of typing results. In fact, the typing result of [Möd12] has a counter-example because it lacks this requirement (see Section 4.4.3). Finally, the new rule Ana'_4 expresses that decomposition is invariant under substitution.

Example 4.1 *In concrete examples of the remaining chapters we use the following* Ana *theory on the usual set of cryptographic primitives, but instead of the function* pub *from private to public keys we use the function* inv *from public to private keys:*

$$\begin{aligned}
\mathsf{Ana}(\mathsf{crypt}(k,m)) &= (\{\mathsf{inv}(k)\},\{m\})\\
\mathsf{Ana}(\mathsf{scrypt}(k,m)) &= (\{k\},\{m\})\\
\mathsf{Ana}(\mathsf{sign}(k,m)) &= (\emptyset,\{m\})\\
\mathsf{Ana}(\langle t,t'\rangle) &= (\emptyset,\{t,t'\}) \textit{ where } \langle\cdot,\cdot\rangle\in\Sigma^2 \textit{ is a pairing operator}\\
\mathsf{Ana}(t) &= (\emptyset,\emptyset) \textit{ for all other terms } t
\end{aligned}$$

4.2 Stateful Protocols

In this section we will define our protocol model. There are several protocol models based on strands where protocol execution is defined in terms of a state transition system, e.g., [CM12, AMMV15]. In these works a state is a set (or multi-set) of strands that represents the honest agents, and a representation of the intruder knowledge. We extend this model with strands that work with sets to model long-term mutable state information. Thus a distinguishing feature of our strands is that honest agents can query and update sets. A protocol state in our model will thus contain not only short-term session information but also the long-term contents of sets, and we call protocols based on these strands *stateful*.

4.2.1 Strands with Sets

We now define the syntax of *strands with sets* as an extension of the strands of Chapter 3 (the part of the syntax marked with $\star$ corresponds to the strands of Chapter 3):

$$
\begin{aligned}
\mathcal{S} \ ::= \ & \phi.\mathcal{S} \mid \overbrace{\psi.\mathcal{S} \mid 0}^{\star} \\
& \text{with } \psi ::= \mathsf{send}(t) \mid \mathsf{receive}(t) \mid t \doteq t' \mid \forall \bar{x}.\ t \not\doteq t' \\
& \text{and } \ \phi ::= \mathsf{insert}(t, s) \mid \mathsf{delete}(t, s) \mid t \mathrel{\dot\in} s \mid \forall \bar{x}.\ t \mathrel{\dot\notin} s
\end{aligned}
$$

where t, t', s range over terms, and $\bar{x}$ ranges over finite sequences $x_1, \ldots, x_n$ of variables from $\mathcal{V}$.

Many formalisms have a notion of *events*. We can model such events using set operations and a distinguished set events containing the emitted events. For that reason we define the following syntactic sugar to express constraints on events, where e range over terms:

$$
\begin{aligned}
\mathsf{assert}(e) &\equiv \mathsf{insert}(e, \mathsf{events}) \\
\mathsf{event}(e) &\equiv e \mathrel{\dot\in} \mathsf{events} \\
\forall \bar{x}.\ \neg\mathsf{event}(e) &\equiv \forall \bar{x}.\ e \mathrel{\dot\notin} \mathsf{events}
\end{aligned}
$$

That is, $\mathsf{assert}(e)$ emits the event e while $\mathsf{event}(e)$ and $\forall \bar{x}.\ \neg\mathsf{event}(e)$ are positive and negative queries on the emitted events. Note that we do not define syntactic sugar for retracting events, although we could do that using a delete operation. The common notion of events is that they are persistent, i.e., the set of emitted events usually grow monotonically during protocol execution. For that reason one should also ensure that deletion from events never occurs, but this is easy to achieve.

Strands built according to the above grammar but using only the cases marked with $\star$ are referred to as *ordinary strands.* A strand consists of a sequence of *steps* and we use here a process calculus notation where we delimit steps by periods and mark the end of a strand with a 0. We normally omit writing the end-marker 0 when it is obvious from the context. We will also omit writing the quantifier $\forall \bar{x}$ whenever $\bar{x}$ is the empty sequence.

The steps can be categorized into three parts: the message transmission steps (send and receive), the equality checks ($\doteq$ and $\not\doteq$), and the set operations (insert, delete, $\dot\in$, and $\dot\notin$). The most basic ones are the message transmission steps which denote transmission over an insecure network. A $\mathsf{send}(t)$ step then means that an agent transmits t and $\mathsf{receive}(t)$ means that an agent waits for a *message pattern* (since it might contain variables) of the form t. Like [AMMV15], we have extended strands with equalities and inequalities: they represent checks that must hold true to proceed.Finally, the novel addition to the concept of strands are the set operations. They allow for updates (insert and delete) and queries ($\dot\in$ and $\dot\notin$) of sets. Here, the delete operation allows for removal of elements that have previously been inserted into a set, and so the contents of sets do not necessarily grow monotonically during transitions. This is in contrast to the messages that the intruder has seen, i.e., the messages sent by honest agents; we cannot force the intruder to forget a message he has learned. Thus the set of messages sent over the network grow monotonically during transitions.

The set of *terms occurring in a strand* $\mathcal{S}$ is denoted by $trms(\mathcal{S})$. The *free variables*, denoted by $fv(\mathcal{S})$, are the variables occurring in $\mathcal{S}$ which are not bound by a universal quantifier, and when $fv(\mathcal{S}) = \emptyset$ then $\mathcal{S}$ is said to be *closed*. In many formalisms like process calculi, variables in a receive step would also be considered as bound variables. Since we, however, also express pattern matching here (since we allow arbitrary terms in receive steps, and, in particular, the same variable can occur in several receive steps and more than once), we like to refer to all such variables as free variables, anyway. We will later introduce a notion of well-formed constraints that requires all free variables to first occur in a receive step or a positive check, and thus corresponds to a notion of closedness in other formalisms. Moreover, given a substitution δ we can apply it to a constraint $\mathcal{S}$ as expected, written $\delta(\mathcal{S})$, by applying δ to every free occurrence of a variable in $\mathcal{S}$. Note that the variables of a substitution δ might clash with the bound variables occurring in a strand $\mathcal{S}$, e.g., for $\delta = [y \mapsto f(x)]$ and $\mathcal{S} = \forall x.\ x \not\doteq y$ we have that $\delta(\mathcal{S}) = \forall x.\ x \not\doteq f(x)$. However, we can always avoid these issues by variable-renaming. For simplicity we therefore assume that the bound and free variables of strands are disjoint. Note also that we restrict ourselves here to a "bare metal" formalism by discarding all notions that are not relevant to our typing result. For instance, we have no notion of repetition, since one can simply consider an infinite set of such strands. We then also do not need a construct for creating fresh constants since we can simply consider a set of strands with uniquely chosen constants. However, we do support an unbounded number of sessions and freshly generated nonces, by modeling protocols as infinite set of transaction strands. This is similar to Guttman's original strand spaces [THG99] that can model an infinite number of strands containing an infinite number of fresh constants. The actual specification language for an end-user should include constructs like creating fresh nonces and repetitions. For that reason we show in Section 4.6 how to connect to formalisms like Set-π and AIF-ω.

4.2.2 A Keyserver Example

Before we proceed with the formal definition of our protocol model we introduce a small keyserver protocol example adapted from [MB16]. In this protocol each participant u has an associated keyring $\mathsf{ring}(u)$ of currently used public keys. Any agent (or user) can register public keys with a trusted keyserver and these public keys can later be revoked. The lifetime of a key may span multiple sessions, but whenever it is revoked the corresponding private key will be publicly known, and it should therefore not be used in a later session. Thus the keyserver needs to maintain the current status of keys and to model this feature we consider sets $\mathsf{valid}(u)$ and $\mathsf{revoked}(u)$ containing the valid respectively revoked keys for each user u. As an initial rule of the protocol we model an out-of-band registration of fresh keys (e.g., the user physically visits the server). Suppose we have a (countably infinite) set of constants that represents the users. For every user u and for every $j \in \mathbb{N}$ we then declare the strand:

$$\mathsf{insert}(pk_{u,j}, \mathsf{ring}(u)).\mathsf{insert}(pk_{u,j}, \mathsf{valid}(u)).\mathsf{send}(pk_{u,j}) \tag{T1}$$

where each $pk_{u,j}$ is a public key. Here, j is a "session number" and $pk_{u,j}$ represents a fresh public key the user u "has created in session j". This strand represents an out-of-band registration between a user u and the server, e.g., the user physically visits a registration office on the server side: the user u creates a fresh key $pk_{u,j}$ and inserts it into its keyring, and the server then additionally inserts the key into its own set of valid keys. Lastly, the key is made public by sending it out.

We will later define the semantics of protocols by a state transition system, where in the initial state all sets are empty and no messages have been sent. Then for user $u = a$ and session $j = 1$, the above strand would get us to a new state where $pk_{a,1}$ is contained in $\mathsf{ring}(a)$ and $\mathsf{valid}(a)$ and the message $pk_{a,1}$ has been sent. Note that we do not have any built-in notion of set ownership, so we can model here strands that represent a mutual action of a user and the server.

As a second rule we model a key-revocation mechanism consisting of two separate strands: one for the users and one for the server. In the first strand the condition $PK_{u,j} \mathrel{\dot{\in}} \mathsf{ring}(u)$ expresses that $PK_{u,j}$ can be any value in the keyring. Not having any other condition, this models that the user can arbitrarily select a key from its keyring. Then it generates a fresh key $npk_{u,j}$, inserts it into its keyring, and sends the new key to the server, signed with the old key $PK_{u,j}$:

$$PK_{u,j} \mathrel{\dot{\in}} \mathsf{ring}(u).\mathsf{insert}(npk_{u,j}, \mathsf{ring}(u)).\mathsf{send}(\mathsf{sign}(\mathsf{inv}(PK_{u,j}), \langle u, npk_{u,j}\rangle)) \tag{T2}$$

for each user u and for each session $j \in \mathbb{N}$. (Note that we also parameterize the variables; later on, we will require that different strands have different variables.) Rule T2 is, for instance, applicable to our concrete state where key $pk_{a,1}$ has been registered: it gets us to a new state where $npk_{a,1}$ has been added to $\mathsf{ring}(a)$ and the message $\mathsf{sign}(\mathsf{inv}(pk_{a,1}), \langle a, npk_{a,1}\rangle)$ has now been sent.

Afterwards, in the second strand, it is the keyserver's turn to act and its actions are initiated by an incoming message of the form $\mathsf{sign}(\mathsf{inv}(PK_i), \langle U_i, NPK_i\rangle)$:

$$\begin{array}{l}\mathsf{receive}(\mathsf{sign}(\mathsf{inv}(PK_i), \langle U_i, NPK_i\rangle)).\\ (\forall A_i.\ NPK_i \mathrel{\dot{\notin}} \mathsf{valid}(A_i)).(\forall A_i.\ NPK_i \mathrel{\dot{\notin}} \mathsf{revoked}(A_i)).\\ PK_i \mathrel{\dot{\in}} \mathsf{valid}(U_i).\mathsf{insert}(NPK_i, \mathsf{valid}(U_i)).\\ \mathsf{insert}(PK_i, \mathsf{revoked}(U_i)).\mathsf{delete}(PK_i, \mathsf{valid}(U_i)).\\ \mathsf{send}(\mathsf{inv}(PK_i)) \qquad\qquad\qquad\qquad\qquad \text{(T3)}\end{array}$$

for each $i \in \mathbb{N}$. Again, this rule is applicable to the concrete state reached above, moves the value $pk_{a,1}$ from $\mathsf{valid}(a)$ to $\mathsf{revoked}(a)$, and inserts $npk_{a,1}$ into $\mathsf{valid}(a)$. Finally, the server discloses the private key $\mathsf{inv}(pk_{a,1})$; while this is of course not done in an actual implementation, it expresses that this protocol is secure even if the intruder learns the private key to an old revoked key.

4.2.3 Transaction Strands

One may wonder about the execution model for the strands from the previous example, in particular if that could cause race conditions on the checks and modifications of the sets if parallel execution of several strands leads to some interleaving of the respective set operations. Suppose for instance, in our keyserver example, that we register the key $pk_{a,1}$ using strand T1, and then send out the messages $\mathsf{sign}(\mathsf{inv}(pk_{a,1}), \langle a, npk_{a,i}\rangle)$ for $i \in \{1, 2\}$ using T2. Then $pk_{a,1}$ is in $\mathsf{valid}(a)$ and $\mathsf{ring}(a)$ contains the keys $pk_{a,1}$, $npk_{a,1}$, and $npk_{a,2}$. If we now run two instances of the strand T3, one for each of the signatures, and we assume that they are executed step-by-step instead of one atomic block, then we could end up in a state where both $npk_{a,1}$ and $npk_{a,2}$ have been registered at the keyserver (i.e., inserted into $\mathsf{valid}(a)$) but only one public key, $pk_{a,1}$, has been revoked, because both instances of T3 can perform all their checks before updating their databases. In fact, as we will define formally in the next subsection, we adopt a *transaction semantics*: a *transaction strand* (or just transaction) is defined to be a strand of the form $\mathsf{receive}(T).L.\mathsf{send}(T')$ where T and T' are finite sets of terms, L is a strand that does not contain any send or receive steps, and where we write $\mathsf{receive}(\{t_1, \dots, t_n\})$ as an abbreviation for $\mathsf{receive}(t_1). \cdots .$ $\mathsf{receive}(t_n)$ (and similarly for send steps). The idea is that such a transaction is always performed atomically, i.e., as a single transition. This reflects, in our opinion, very well the normal work-flow of a web server with a database: the server receives an incoming request, performs some lookups and checks on its database (possibly aborting the transaction), then performs some modifications on its database, and sends a reply (which may be also a request to another server). The key is that the server serializes the handling of such transactions (to avoid said race conditions). A transaction semantics allows us to abstract from the implementation of such serialization mechanisms and thus focus on the verification of a larger system. Another example are crypto APIs, where a token receives an API command, performs some lookups and checks in its memory (possibly aborting the transaction), performs some updates to its memory and then gives out a result. Also here, we typically do not want to reason about race conditions from several API calls in parallel.

This is indeed slightly different from the "philosophy" of many process calculus approaches (e.g., StatVerif [ARR11] and Set-π [BMNN15]) where one would have to introduce explicit locking mechanisms. Also, the original notion of strand spaces by Guttman [THG99] is actually based on a notion of only a partial (instead of a total) order on send and receive steps in an execution; if we regard however set operations as interactions with a database with locking, then we obtain the partial order that our transaction semantics defines.

4.2.4 Transition Systems

Now that we have introduced the elements of our protocols we define a *protocol* $\mathcal{P}$ to be a countable set of transaction strands where no variable occurs in two different strands. The set of terms $trms(\mathcal{P})$ occurring in $\mathcal{P}$ is defined as expected.

Before giving the formal definition of the transition system we will first define the notion of a *database mapping* D to be a finite set of pairs (t, s) of terms, and for closed strands $\mathcal{S}$ we define the ground database mapping $db(\mathcal{S})$ as

$$db(\mathcal{S}) = \{(t,s) \mid \mathsf{insert}(t,s).\mathcal{S}' \text{ is a suffix of } \mathcal{S} \text{ and } \mathsf{delete}(t,s) \text{ does not occur in } \mathcal{S}'\}$$

Let $D = \{(t_1, s_1), \ldots, (t_n, s_n)\}$ be a database mapping and $\mathcal{S}$ a closed strand, then we may write $db(\mathsf{insert}(D).\mathcal{S})$ as a shorthand for the database mapping $db(\mathsf{insert}(t_1, s_1). \cdots .\mathsf{insert}(t_n, s_n).\mathcal{S})$.

States in the (ground) transition system are of the form $(\mathcal{P}; M, D)$ where $\mathcal{P}$ is a protocol, M is the set of messages that has been sent over the network and that we also refer to as the intruder knowledge, and D is a database mapping representing the state of all databases (including the events that have occurred). The initial state is $(\mathcal{P}_0; \emptyset, \emptyset)$ for a protocol $\mathcal{P}_0$.

Definition 4.1 A transition relation on states is defined as:

$$(\mathcal{P}; M, D) \overset{\sigma,\mathcal{S}}{\Longrightarrow} (\mathcal{P} \setminus \{\mathcal{S}\}; M \cup \sigma(T'), db(\mathsf{insert}(D).\sigma(L)))$$

if the following conditions are met:

C1: $\mathcal{S} = \mathsf{receive}(T).L.\mathsf{send}(T') \in \mathcal{P}$ is a transaction strand,

C2: σ is a ground substitution with domain $fv(\mathcal{S})$,

C3: $M \vdash \sigma(t)$ for all terms $t \in T$,

C4: $\sigma(t) = \sigma(t')$ for all steps $t \doteq t'$ occurring in L,

C5: $\sigma(\delta(t)) \neq \sigma(\delta(t'))$ for all steps $\forall\bar{x}.\ t \mathrel{\dot{\neq}} t'$ occurring in L and all ground substitutions δ with domain $\bar{x}$,

C6: $\sigma((t, s)) \in db(\mathsf{insert}(D).\sigma(L'))$ for all prefixes $L'.(t \mathrel{\dot{\in}} s)$ of L,

C7: $\sigma(\delta((t, s))) \notin db(\mathsf{insert}(D).\sigma(L'))$ for all prefixes $L'.(\forall\bar{x}.\ t \mathrel{\dot{\notin}} s)$ of L and all ground substitutions δ with domain $\bar{x}$,

Here the first side-condition C1 simply ensures that $\mathcal{S}$ is actually a transaction strand of the protocol, and the second condition C2 ensures that σ is actually an assignment of the free variables in $\mathcal{S}$ to concrete values. Condition C3 states that the intruder must be able to derive the messages that $\mathcal{S}$ expects to receive. The conditions C4 to C7 state that all checks and set updates performed by $\mathcal{S}$ are satisfied under σ. As the effect of a transition the strand $\mathcal{S}$ is removed from $\mathcal{P}$, the intruder learns $\sigma(T')$, and the databases are updated according to the set operations of $\sigma(L)$.

Note that the whole transaction strand $\mathcal{S}$ is "consumed" in each transition because we want the strands of protocols to be atomic transactions. This is different from other strand-based approaches in which a transition only eliminates one step of a strand and in which strands might contain multiple transactions (e.g., from a state containing the protocol $\{PK \doteq pk_{a,1}.\mathsf{receive}(npk_{a,1}).\mathsf{send}(PK)\}$ we can reach a state containing $\{\mathsf{receive}(npk_{a,1}).\mathsf{send}(PK)\}$ and where PK must be mapped to $pk_{a,1}$). Defining our protocol semantics on a transactional level, however, is without loss of generality: it is always possible to break a strand into smaller transaction strands while preserving the causal relationship of the original strand (i.e., transaction $i+1$ of a strand with n transactions can only be performed after transaction i, for any $i \in \{1, \dots, n-1\}$). For instance, one can insert additional message-transmissions between steps, e.g., the strand $PK \doteq pk_{a,1}.\mathsf{receive}(npk_{a,1}).\mathsf{send}(PK)$ can be split into two transactions, namely $PK \doteq pk_{a,1}.\mathsf{send}(f(PK))$ and $\mathsf{receive}(f(PK)).\mathsf{receive}(npk_{a,1}).\mathsf{send}(PK)$ where f is a fresh private symbol of arity one that we here use to preserve the causal relationship and to carry state information. In general, to split a strand $\mathcal{S}_1.\cdots.\mathcal{S}_n$ containing transaction strands $\mathcal{S}_i$ we can add additional steps that carry state information from $\mathcal{S}_i$ to $\mathcal{S}_{i+1}$ and which ensure that $\mathcal{S}_{i+1}$ can only be performed after $\mathcal{S}_i$: $\mathcal{S}_i.\mathsf{send}(\mathsf{state}_{\mathcal{S}_i}(x_1, \dots, x_m))$ and $\mathsf{receive}(\mathsf{state}_{\mathcal{S}_i}(x_1, \dots, x_m)).\mathcal{S}_{i+1}$, where $\mathsf{state}_{\mathcal{S}_i} \in \Sigma^m_{priv}$ is private and unique to $\mathcal{S}_i$ and where $fv(\mathcal{S}_i) = \{x_1, \dots, x_m\}$.

Such a transformation can also be used to link transactions $\mathcal{S}_1, \ldots, \mathcal{S}_n$ together, or to split a transaction strand into smaller transactions if one wishes to have greater granularity in state transitions. For tools based on transaction strands such an encoding would be useful; it would be convenient for users if they are allowed to specify strands containing multiple transactions. In this chapter, however, we will not provide such an input language for a tool—rather, we have decided to keep the protocol model simple by only allowing single-transaction strands. This decision is legitimate, in our opinion, since the above encoding for linking transactions can easily be automated and be transparent to end-users.

Finally, we note that protocol goals such as secrecy can also be encoded as strands. For instance, we can extend our running keyserver example with strands

$$\mathsf{receive}(\mathsf{inv}(PK'_i)).PK'_i \mathrel{\dot\in} \mathsf{valid}(h).\mathsf{assert}(\mathsf{attack})$$

for each honest user h and $i \in \mathbb{N}$, and an event attack that denotes when an attack has happened. Hence, if the private key of a valid public key for an honest agent is leaked then there is a violation of secrecy, and in those cases we emit the event attack using the construct assert. In other words, if there is a reachable state $(\mathcal{P}; M, D)$ in which $(\mathsf{attack}, \mathsf{events}) \in D$ then the protocol has a vulnerability. In principle we support all properties expressible in the geometric fragment [AMMV15] over events. This includes many reachability goals like authentication.

4.3 Symbolic Constraints

At the core of all typing results is a sound and complete constraint reduction system. It was originally used as an efficient procedure for model-checking of security protocols [MS01, RT03, BMV05], but is also used as a proof technique when proving relative soundness results such as [CD09, Möd12, AD14, AMMV15, HM17]. The constraints themselves arise from the symbolic exploration of the protocol state space where each symbolic state contains a constraint that represents the steps taken in the protocol so far. Any solution to a reachable constraint then represents (one or several) concrete runs of the protocol. In this section we consider constraints for stateful protocols.

4.3.1 Syntax and Semantics

The most basic parts of a symbolic constraint are requirements on the intruder to produce messages that honest agents expect to receive. For instance, if the messages $m_1, \dots, m_n$ (where each m_i might contain variables) have been sent out and some agent expects to receive a message pattern t it is standard to represent as a constraint the requirement on the intruder to produce t given the m_i. Any solution $\mathcal{I}$ to such a constraint is an assignment of the variables $fv(\{m_1, \dots, m_n, t\})$ to ground terms such that $\mathcal{I}(\{m_1, \dots, m_n\}) \vdash \mathcal{I}(t)$ holds. As shown in the previous chapter we can represent a (finite) set of such constraints by a strand, with send steps for messages the intruder has to generate, and receive steps for messages that the intruder learns (all in the order this happens), e.g., the constraint we just explained can be represented as the strand $\mathsf{receive}(m_1).\cdots.\mathsf{receive}(m_n).\mathsf{send}(t)$. We additionally want to handle strands with sets, and so we also just insert all the set operations (and similarly the checks and event assertions) into the intruder strands in the order they happen in a concrete execution. With this, our constraints are just like the strands for honest agents but with the direction of send and receive steps inverted, i.e., a send step from an honest agent becomes a receive step in our constraints and vice versa. For these reasons we define the syntax of our constraints to range over strands, i.e., they are defined as finite sequences of *steps* and are built from the following grammar where t and t' ranges over terms and $\bar{x}$ over finite variable sequences $x_1, \dots, x_n$:

$$\begin{array}{rcl} \mathcal{A} & ::= & \mathsf{send}(t).\mathcal{A} \mid \mathsf{receive}(t).\mathcal{A} \mid t \doteq t'.\mathcal{A} \mid (\forall \bar{x}.\ t \not\doteq t').\mathcal{A} \mid \\ & & \mathsf{insert}(t,t').\mathcal{A} \mid \mathsf{delete}(t,t').\mathcal{A} \mid t \mathrel{\dot\in} t'.\mathcal{A} \mid (\forall \bar{x}.\ t \mathrel{\dot\notin} t').\mathcal{A} \mid 0 \end{array}$$

Similarly to the ordinary strands we call constraints that only contain receive, send, equalities, and inequalities for *ordinary* or *stateless constraints*. We will often reuse the operations defined on strands for symbolic constraints, since they share the same syntax, and we also make the assumption that the bound variables occurring in a constraint are disjoint from its free variables. Moreover, we define the *intruder knowledge* $ik(\mathcal{A})$ of a constraint $\mathcal{A}$ as the set of received messages: $ik(\mathcal{A}) = \{t \mid \mathsf{receive}(t) \text{ occurs in } \mathcal{A}\}$.

An *interpretation* $\mathcal{I}$ *for a constraint* $\mathcal{A}$ (or just an *interpretation* if $dom(\mathcal{I}) = \mathcal{V}$) is now defined to be a substitution such that $fv(\mathcal{A}) \subseteq dom(\mathcal{I})$ and $ran(\mathcal{I})$ is ground. We then inductively define a model relation $[\![M, D; \cdot]\!]_s$ between interpretations and constraints where M and D are respectively the initial intruder knowledge and the state of databases:

Definition 4.2 (The semantics of stateful constraints)

$$\begin{array}{lll}
[\![M, D; 0]\!]_s\ \mathcal{I} & \text{iff} & \mathit{true} \\
[\![M, D; \mathsf{send}(t).\mathcal{A}]\!]_s\ \mathcal{I} & \text{iff} & M \vdash \mathcal{I}(t) \text{ and } [\![M, D; \mathcal{A}]\!]_s\ \mathcal{I} \\
[\![M, D; \mathsf{receive}(t).\mathcal{A}]\!]_s\ \mathcal{I} & \text{iff} & [\![M \cup \{\mathcal{I}(t)\}, D; \mathcal{A}]\!]_s\ \mathcal{I} \\
[\![M, D; t \doteq t'.\mathcal{A}]\!]_s\ \mathcal{I} & \text{iff} & \mathcal{I}(t) = \mathcal{I}(t') \text{ and } [\![M, D; \mathcal{A}]\!]_s\ \mathcal{I} \\
[\![M, D; (\forall \bar{x}.\ t \not\doteq t').\mathcal{A}]\!]_s\ \mathcal{I} & \text{iff} & [\![M, D; \mathcal{A}]\!]_s\ \mathcal{I} \text{ and } \mathcal{I}(\delta(t)) \neq \mathcal{I}(\delta(t')) \\
& & \text{for all ground substitutions } \delta \\
& & \text{with domain } \bar{x} \\
[\![M, D; \mathsf{insert}(t, s).\mathcal{A}]\!]_s\ \mathcal{I} & \text{iff} & [\![M, D \cup \{\mathcal{I}((t, s))\}; \mathcal{A}]\!]_s\ \mathcal{I} \\
[\![M, D; \mathsf{delete}(t, s).\mathcal{A}]\!]_s\ \mathcal{I} & \text{iff} & [\![M, D \setminus \{\mathcal{I}((t, s))\}; \mathcal{A}]\!]_s\ \mathcal{I} \\
[\![M, D; t \dot{\in} s.\mathcal{A}]\!]_s\ \mathcal{I} & \text{iff} & \mathcal{I}((t, s)) \in D \text{ and } [\![M, D; \mathcal{A}]\!]_s\ \mathcal{I} \\
[\![M, D; (\forall \bar{x}.\ t \dot{\notin} s).\mathcal{A}]\!]_s\ \mathcal{I} & \text{iff} & [\![M, D; \mathcal{A}]\!]_s\ \mathcal{I} \text{ and } \mathcal{I}(\delta((t, s))) \notin D \\
& & \text{for all ground substitutions } \delta \\
& & \text{with domain } \bar{x}
\end{array}$$

Finally, we say that an interpretation $\mathcal{I}$ is a *model of (or solution to) a constraint* $\mathcal{A}$, written $\mathcal{I} \models_s \mathcal{A}$, iff $[\![\emptyset, \emptyset; \mathcal{A}]\!]_s\ \mathcal{I}$.

We can now prove some useful lemmas about the constraint semantics. First, we have a lemma that we frequently apply in proofs (without explicitly referencing it) that allows us to split and merge constraints:

Lemma 4.3 *Given a ground set of terms M, a ground database mapping D, an interpretation $\mathcal{I}$, and symbolic constraints $\mathcal{A}$ and $\mathcal{A}'$, the following holds:*

$[\![M, D; \mathcal{A}.\mathcal{A}']\!]_s\ \mathcal{I}$ *if and only if* $[\![M, D; \mathcal{A}]\!]_s\ \mathcal{I}$ *and* $[\![M', D'; \mathcal{A}']\!]_s\ \mathcal{I}$
where $M' = M \cup ik(\mathcal{I}(\mathcal{A}))$ *and* $D' = db(\mathsf{insert}(D).\mathcal{I}(\mathcal{A}))$

Secondly, we can prove a useful relationship between the side-conditions C1 to C7 of the ground transition system and the constraint semantics. First we define the notion of the *dual* of a strand S by "swapping" the direction of receive and send steps. Formally, $dual(\mathfrak{s})$ denotes the dual of the strand step $\mathfrak{s}$ defined such that $dual(\mathsf{receive}(t)) = \mathsf{send}(t)$, $dual(\mathsf{send}(t)) = \mathsf{receive}(t)$, and $dual(\mathfrak{s}) = \mathfrak{s}$ for any other step $\mathfrak{s}$. It is then extended homomorphically to strands as expected. We will interpret dual strands as symbolic constraints and under this interpretation we can prove the following relationship:

Lemma 4.4 *Given a ground state $(\mathcal{P}; M, D)$, a transaction strand $\mathcal{S} \in \mathcal{P}$ (condition C1), and a ground substitution σ with domain $fv(\mathcal{S})$ (condition C2), then the conditions C3 to C7 hold if and only if $[\![M, D; dual(\mathcal{S})]\!]_s\ \sigma$.*

4.3.2 Symbolic Transition System

Now that we have defined the syntax and semantics of constraints we can construct a protocol transition system in which we build up constraints during transitions. In this symbolic transition system a symbolic state $(\mathcal{P}; \mathcal{A})$ consists of a protocol $\mathcal{P}$ and a constraint $\mathcal{A}$, and the initial state $(\mathcal{P}_0; 0)$ then consists of the initial protocol $\mathcal{P}_0$ and the empty constraint 0. During transitions we then build up a constraint by interpreting dual honest-agent strands as constraints:

Definition 4.5 A transition relation on symbolic states is defined as:

$$(\mathcal{P}; \mathcal{A}) \overset{\mathcal{S}}{\Longrightarrow}^{\bullet} (\mathcal{P} \setminus \{\mathcal{S}\}; \mathcal{A}.\mathit{dual}(\mathcal{S})) \text{ if } \mathcal{S} \in \mathcal{P}$$

We will now impose a well-formedness requirement on protocols; variables in honest-agent strands must either originate from a received message or in a positive check (e.g., a set query). In ordinary protocols there is nothing non-deterministic in the behavior of honest agents, so all free variables in their strands shall first occur in messages they receive. Now that we add set operations, we extend well-formedness naturally to set comprehensions: a set-membership check like $x \mathrel{\dot\in} s$ allows the agent to non-deterministically choose any element from s for x—unless x is already constrained before, thus limiting the choice accordingly.

We also require that reachable constraints in the symbolic transition system are of a well-formed kind that is dual to the well-formedness of protocols; every free variable of a constraint represents either a message that depends on choices the intruder can make (e.g., variables originating from send steps), or originates from a positive check. To that end we formally define constraint well-formedness first and then use this definition to define protocol well-formedness:

Definition 4.6 A constraint $\mathcal{A}$ is *well-formed w.r.t. variables* X (or simply *well-formed* when $X = \emptyset$), written $\mathit{wf}_X(\mathcal{A})$, where

$$\begin{array}{lll}
\mathit{wf}_X(0) & \text{iff} & \mathit{true} \\
\mathit{wf}_X(\mathsf{send}(t).\mathcal{A}) & \text{iff} & \mathit{wf}_{X \cup \mathit{fv}(t)}(\mathcal{A}) \\
\mathit{wf}_X(\mathsf{receive}(t).\mathcal{A}) & \text{iff} & \mathit{fv}(t) \subseteq X \text{ and } \mathit{wf}_X(\mathcal{A}) \\
\mathit{wf}_X(t \doteq t'.\mathcal{A}) & \text{iff} & \mathit{fv}(t') \subseteq X \text{ and } \mathit{wf}_{X \cup \mathit{fv}(t)}(\mathcal{A}) \\
\mathit{wf}_X(\mathsf{insert}(t,s).\mathcal{A}) & \text{iff} & \mathit{fv}(t) \cup \mathit{fv}(s) \subseteq X \text{ and } \mathit{wf}_X(\mathcal{A}) \\
\mathit{wf}_X(t \mathrel{\dot\in} s.\mathcal{A}) & \text{iff} & \mathit{wf}_{X \cup \mathit{fv}(t) \cup \mathit{fv}(s)}(\mathcal{A}) \\
\mathit{wf}_X(\mathfrak{a}.\mathcal{A}) & \text{iff} & \mathit{wf}_X(\mathcal{A}) \text{ otherwise}
\end{array}$$

Here the set X collects the variables that have occurred in send steps or positive checks. In other words, every free variable of a well-formed constraint originates from either a send step, a $\dot{\in}$ step, an event step, or at the left-hand side of a $\doteq$ step (or a delete or a negative check such as an inequality, but in those cases the new variables cannot be used elsewhere). We can then reuse the definition of well-formedness of constraints to formally define a notion of well-formedness of protocols:

Definition 4.7 A protocol $\mathcal{P}$ is *well-formed* iff for all strands $\mathcal{S} \in \mathcal{P}$ the symbolic constraint $dual(\mathcal{S})$ is well-formed.

Note that the well-formedness requirement on $t \mathrel{\dot{\in}} s$ allows us to model protocols where we pick arbitrary elements from sets—as in the keyserver example.

Well-formedness of reachable constraints is now easy to prove. We write $\overset{w}{\Longrightarrow}^*$ here to denote the reflexive-transitive closure of $\overset{\cdot}{\Longrightarrow}$ where the label $w = (\sigma_1, \mathcal{S}_1), \ldots, (\sigma_n, \mathcal{S}_n)$ denotes a sequence of transition labels, and similarly $\overset{w'}{\Longrightarrow}^{\bullet *}$ denotes the reflexive-transitive closure of $\overset{\cdot}{\Longrightarrow}^{\bullet}$ where $w' = \mathcal{S}_1, \ldots, \mathcal{S}_n$.

Lemma 4.8 (Well-formedness of reachable constraints) [1] *If $\mathcal{P}_0$ is a well-formed protocol and $(\mathcal{P}_0; 0) \overset{w}{\Longrightarrow}^{\bullet *} (\mathcal{P}; \mathcal{A})$ then $\mathcal{A}$ is a well-formed symbolic constraint and $\mathcal{P}$ is a well-formed protocol.*

We now prove that the symbolic and ground transition systems are equivalent. Essentially, if we consider for every reachable symbolic state $(\mathcal{P}; \mathcal{A})$ and every model $\mathcal{I}$ of $\mathcal{A}$ the corresponding ground state $(\mathcal{P}; ik(\mathcal{I}(\mathcal{A})), db(\mathcal{I}(\mathcal{A})))$, then we obtain exactly the reachable states of the ground transition system:

Theorem 4.9 (Equivalence of transition systems) [2] *For any protocol $\mathcal{P}_0$,*

$$\begin{aligned}&\{(\mathcal{P}; M, D) \mid \exists w.\ (\mathcal{P}_0; \emptyset, \emptyset) \overset{w}{\Longrightarrow}^* (\mathcal{P}; M, D)\} = \\ &\{(\mathcal{P}; ik(\mathcal{I}(\mathcal{A})), db(\mathcal{I}(\mathcal{A}))) \mid \exists w.\ (\mathcal{P}_0; 0) \overset{w}{\Longrightarrow}^{\bullet *} (\mathcal{P}; \mathcal{A}) \text{ and } \mathcal{I} \models_s \mathcal{A}\}\end{aligned}$$

4.3.3 Reduction to Ordinary Constraints

The key to our typing result—that allows us to benefit from existing typing results—is to first reduce the problem of solving general intruder constraints (with set operations) to solving ordinary intruder constraints (without set operations). To that end we introduce a sound and complete translation mechanism

[1]Not proven in Isabelle.
[2]Not proven in Isabelle.

that removes the stateful parts of constraints, for instance those reachable in $\overset{\cdot}{\Longrightarrow}^{\bullet *}$. The translation $tr(\cdot)$ is then defined as follows, where D is a database mapping that records what has occurred in the constraint so far:

Definition 4.10 (Translation of symbolic constraints) Given a constraint $\mathcal{A}$ its translation into ordinary constraints is denoted by $tr(\mathcal{A}) = tr_{\emptyset}(\mathcal{A})$ where:

$$
\begin{array}{rcl}
tr_D(0) & = & \{0\} \\
tr_D(\mathsf{insert}(t,s).\mathcal{A}) & = & tr_{D\cup\{(t,s)\}}(\mathcal{A}) \\
tr_D(\mathsf{delete}(t,s).\mathcal{A}) & = & \{(t,s) \doteq d_1.\cdots.(t,s) \doteq d_i. \\
& & (t,s) \not\doteq d_{i+1}.\cdots.(t,s) \not\doteq d_n.\mathcal{A}' \mid \\
& & \;D = \{d_1,\dots,d_i,\dots,d_n\}, 0 \le i \le n, \\
& & \;\mathcal{A}' \in tr_{D\setminus\{d_1,\dots,d_i\}}(\mathcal{A})\} \\
tr_D(t \mathrel{\dot\in} s.\mathcal{A}) & = & \{(t,s) \doteq d.\mathcal{A}' \mid d \in D, \mathcal{A}' \in tr_D(\mathcal{A})\} \\
tr_D((\forall \bar{x}.\ t \mathrel{\dot\notin} s).\mathcal{A}) & = & \{(\forall \bar{x}.\ (t,s) \not\doteq d_1).\cdots.(\forall \bar{x}.\ (t,s) \not\doteq d_n).\mathcal{A}' \mid \\
& & \;D = \{d_1,\dots,d_n\}, 0 \le n, \mathcal{A}' \in tr_D(\mathcal{A})\} \\
tr_D(\mathfrak{a}.\mathcal{A}) & = & \{\mathfrak{a}.\mathcal{A}' \mid \mathcal{A}' \in tr_D(\mathcal{A})\} \text{ otherwise}
\end{array}
$$

Intuitively, the set $tr_D(\cdot)$ of reduced constraints represents a disjunction of ordinary constraints, and since we cannot represent disjunctions in our constraints we use sets instead. Note also that D will always be finite and that this does not mean that we are restricting ourselves to only finitely many sessions. Rather, in each protocol execution only finitely many things have happened at any given point and D then represents the state of the sets and events. Hence the translation always produces a finite set and for this reason we can interpret the set as a finite disjunction of constraints.

We will now explain how each set operation is translated. The purpose of the translation $tr(\mathcal{A})$ is to capture precisely the models of $\mathcal{A}$ using only a finite number of ordinary constraints, so we will proceed with the explanation with this in mind.

The simplest case is the $\mathsf{insert}(t,s)$ case, and here we record the insertion for the remaining translation. Now consider the $t \mathrel{\dot\in} s$ case. For any model $\mathcal{I}$ of $t \mathrel{\dot\in} s$ with a given database mapping $D = \{(t_1,s_1),\dots,(t_n,s_n)\}$ (where each entry of D might contain variables) we know that $\mathcal{I}((t,s)) \in \mathcal{I}(D)$. In other words, some check $(t,s) \doteq d$ for some d in D has $\mathcal{I}$ as a model if and only if $t \mathrel{\dot\in} s$ has $\mathcal{I}$ as a model, and by then constructing one constraint for each $d_i \in D$ where we require $(t,s) \doteq d_i$ we get the desired result.

For the $\forall \bar{x}.\ t \mathrel{\dot\notin} s$ case we know that $\mathcal{I}(\delta(t)) \neq \mathcal{I}(\delta(t'))$ *or* $\mathcal{I}(\delta(s)) \neq \mathcal{I}(\delta(s'))$ for any $(t',s') \in D$ and ground substitution δ with domain $\bar{x}$. In other words,

$\mathcal{I}(\delta((t,s))) \neq \mathcal{I}(\delta((t',s')))$ for all $(t',s') \in D$ and this is exactly what the translation expresses. We also have to make sure that the newly introduced quantified constraints do not capture any variables of D. This is, in fact, the case for all constraints reachable in our symbolic transition system, since we have previously assumed all strands of protocols to have disjoint variables from each other and also that the bound and free variables of strands are disjoint. Thus this property also holds for the reachable constraints.

The most interesting case is the translation of $\mathsf{delete}(t,s)$ steps. Since terms may contain variables we do not know a priori which insertions to remove from D, but we still need to ensure that t has actually been removed from the set s in the remaining constraint translation—otherwise the translation would be unsound. We accomplish this by partitioning the insertions D into those $\{d_1,\dots,d_i\}$ that must be equal to (t,s) in the remaining translation and the remaining $D \setminus \{d_1,\dots,d_i\}$ that are unequal to (t,s), and we thus add equality and inequality constraints to express this partitioning. Consequently, we then remove $\{d_1,\dots,d_i\}$ from D for the remaining translation. Note that there will in general be cases where the choice of partitioning results in an unsatisfiable constraint, but since we construct constraints for *all* possibilities the translation still captures exactly the models of the original constraint. Note also that this partitioning of D implies that an exponential number of constraints are constructed in this case, namely one for each subset of D. The translation is meant to be used purely as a problem reduction—in a verification procedure one could ensure that trivially unsatisfiable translations are ignored to reduce the number of produced constraints.

Finally, we show that tr is indeed a reduction, i.e., that $tr(\mathcal{A})$ captures exactly the models of $\mathcal{A}$, and that tr preserves well-formedness:

Theorem 4.11 (Semantic Equivalence) *Let $\mathcal{A}$ be a constraint and $\mathcal{I}$ an interpretation. Then $\mathcal{A}$ is semantically equivalent to its translation $tr(\mathcal{A})$ in the following sense:*

$$\mathcal{I} \models_s \mathcal{A} \text{ if and only if there exists } \mathcal{A}' \in tr(\mathcal{A}) \text{ such that } \mathcal{I} \models \mathcal{A}'.$$

The translation furthermore preserves well-formedness:

$$\text{If } \mathcal{A} \text{ is well-formed and } \mathcal{A}' \in tr(\mathcal{A}) \text{ then } \mathcal{A}' \text{ is well-formed.}$$

4.4 Lifting Typing Results to Stateful Protocols

So far everything in this chapter has been untyped. We now again consider the type system of Section 3.2 in which we annotate terms with types. In particular, each message pattern that an honest agent in a protocol expects to receive will have an intended type, and in a typed model we restrict all substitutions to well-typed ones. In this typed model the intruder is therefore effectively restricted to only sending messages which conform to the types. For protocols that satisfy the syntactic type-flaw resistance requirement we then prove that this restriction is sound, and this result we call a typing result. For proving our result we use the reduction *tr* from constraints with sets to ordinary constraints, enabling us to use existing typing results for protocols without sets and "lift" them to stateful protocols.

Recall that Γ is a typing function that assigns a type to each term, and that $\mathfrak{T}_a$ is a set of atomic types. In our running example of this chapter we might define $\mathfrak{T}_a = \{\mathsf{Value}, \mathsf{Agent}, \mathsf{Attack}\}$ where $\Gamma(a) = \mathsf{Agent}$ for all users and servers a, $\Gamma(pk) = \mathsf{Value}$ for any element pk of a set, and $\Gamma(\mathsf{attack}) = \mathsf{Attack}$. Similarly, the variables U_i have type Agent and the variables $PK_{u,j}$, PK_i, and NPK_i have type Value. All short-term public keys have type Value and all short-term private keys have type $\mathsf{inv}(\mathsf{Value})$. Since we use terms to model families of sets we have as a consequence that, e.g., keyrings of the form $\mathsf{ring}(u)$, for users u, have type $\mathsf{ring}(\mathsf{Agent})$.

4.4.1 Type-Flaw Resistance for Stateful Protocols

In this subsection we will define a sufficient syntactical condition for stateful protocols (i.e., verifying the condition does not require an exploration of the state space of a protocol) that allows us to prove our typing result for protocols that have this property. This condition is a conservative extension of type-flaw resistance (Definition 3.17) from the previous chapter and so the following condition will again be named *type-flaw resistance* since no confusion can arise.

We again require that all pairs t, t' of sub-message patterns that are not variables (i.e., are non-variable) can only be unified if their types match, and this will be our main condition of type-flaw resistance. This is a sufficient requirement to distinguish terms of different types and it therefore enables us to argue that ill-typed choices are unnecessary. In a nutshell, the typing result works as follows: with the condition of type-flaw resistance we ensure that the intruder cannot take a message generated by an honest agent (or a non-variable subterm of it) and use it in a different "context" of the protocol, i.e., a non-variable subterm of a different type. The constraint-based representation then allows one to argue that no attack relies on an ill-typed choice by the intruder like in the previous chapter. We use again that there is a sound, complete, and terminating reduction procedure for (ordinary) intruder constraints that will instantiate variables only upon unification of two elements of SMP—and such a unifier is guaranteed to be well-typed for a type-flaw resistant protocol. All remaining uninstantiated variables can be instantiated arbitrarily by the intruder, in particular in a well-typed way. Thus one can conclude that there is a well-typed solution if there is one at all.

Definition 4.12 (Type-flaw resistance) First, let the *set operation tuples* of a constraint (or strand) $\mathcal{A}$ be defined as:

$$\begin{aligned} setops(\mathcal{A}) \equiv \{(t,s) \mid\ & \mathsf{insert}(t,s) \text{ or } \mathsf{delete}(t,s) \text{ or } t \mathrel{\dot\in} s \text{ or} \\ & (\forall \bar{x}.\ t \mathrel{\dot\notin} s) \text{ for some } \bar{x} \text{ occurs in } \mathcal{A}\} \end{aligned}$$

and extend this definition to protocols $\mathcal{P}$ as follows:

$$setops(\mathcal{P}) \equiv \bigcup_{\mathcal{S}\in\mathcal{P}} setops(\mathcal{S})$$

Then type-flaw resistance is defined as follows:

1. A set of terms M is *type-flaw resistant* iff for all $t, t' \in SMP(M) \setminus \mathcal{V}$ it holds that $\Gamma(t) = \Gamma(t')$ if t and t' are unifiable.
2. A strand (or constraint) $\mathcal{A}$ is *type-flaw resistant* iff $trms(\mathcal{A}) \cup setops(\mathcal{A})$ is type-flaw resistant, and for any terms t, t' and variable sequences $\bar{x}$:
   (a) If $t \doteq t'$ occurs in $\mathcal{A}$ then $\Gamma(t) = \Gamma(t')$ if t and t' are unifiable.
   (b) If $\forall \bar{x}.\ t \mathrel{\dot{\neq}} t'$ occurs in $\mathcal{A}$ then either
      i. $\Gamma((fv(t) \cup fv(t')) \setminus \bar{x}) \subseteq \mathfrak{T}_a$, or
      ii. there does not exist a subterm of t or t' of the form $f(x_1, \dots, x_n)$ where $n > 0$ and $x_1, \dots, x_n \in \bar{x}$.
   (c) If $\forall \bar{x}.\ t \mathrel{\dot\notin} t'$ occurs in $\mathcal{A}$ then there does not exist a subterm of (t, t') of the form $f(x_1, \dots, x_n)$ where $n > 0$ and $x_1, \dots, x_n \in \bar{x}$.
3. A protocol $\mathcal{P}$ is *type-flaw resistant* iff the set $trms(\mathcal{P}) \cup setops(\mathcal{P})$ is type-flaw resistant and for all $\mathcal{S} \in \mathcal{P}$ the strand $\mathcal{S}$ is type-flaw resistant.

The main type-flaw resistance condition is defined in Definition 4.12(1) and it states that matching pairs of messages that might occur in a protocol run must have the same type. For equality steps $t \doteq t'$ any solution $\mathcal{I}$ must be a unifier of t and t', and so they should have the same type. If $t \doteq t'$ is unsatisfiable (i.e., t and t' are not unifiable) then their types do not matter. Hence we can later prove that our reduction *tr* preserves type-flaw resistance, even if *tr* produces some unsatisfiable equality steps. For inequality steps $\forall \bar{x}.\ t \not\doteq t'$ we only need to require that the variables occurring in t and t' (but not $\bar{x}$) are atomic, or that there are no composed subterm whose immediate parameters are all bound variables. For the remaining constraint steps note that when we translate a set operation such as $\mathsf{delete}(t, s)$ we construct steps of the form $(t, s) \doteq (t', s')$ and $(t, s) \not\doteq (t', s')$. Thus we must require that (t, s) and (t', s') are unifiable if they have the same type and that the translated inequalities are type-flaw resistant (note that inequalities with no bound variables—such as those occurring in the translation of delete steps—satisfy type-flaw resistance). By requiring that the set $\mathit{trms}(\mathcal{A}) \cup \mathit{setops}(\mathcal{A})$ is type-flaw resistant we have that the translated set operations must have the same type if they are unifiable. Similar conditions are needed for the event steps, but we can here relax the requirements slightly since their translations are simpler. Finally, a protocol is type-flaw resistant whenever its strands are, and we must additionally require here that $\mathit{trms}(\mathcal{P}) \cup \mathit{setops}(\mathcal{P})$ is type-flaw resistant because terms from different strands might be unifiable.

To see that these restrictions are necessary, suppose for a moment that we would allow for composed types for variables in inequalities with bound variables without the restrictions imposed by our type-flaw resistance requirements. We can then easily construct constraints which only have ill-typed solutions. For instance, consider the inequality $\forall x.\ y \not\doteq f(x)$ where $\Gamma(y) = f(\Gamma(x))$. For any instance $f(c)$ of y where $\Gamma(f(c)) = \Gamma(y)$ there is an instance of x (namely c) that does not satisfy the inequality. Hence the constraint has no well-typed solution. However, there do exist ill-typed solutions; since we are working in the free algebra, terms are equal if and only if they are syntactically equal, and hence any instance of y that is not of the form $f(c)$ for some c would be a solution to the inequality. [Möd12] has no such restrictions on the type of universally quantified variables and we thus found a counter-example to its typing result (see Section 4.4.3). Thus it seems that a typing result for stateful protocols necessarily requires a carefully restricted setting like our set-based approach.

Note also that we forbid set operations of the form $\forall x_1, \dots, x_n.\ t \not\dot\in f(x_1, \dots, x_n)$ such as the negative membership checks of the keyserver example. This is because the set expression $f(x_1, \dots, x_n)$ is a term whose immediate parameters are all bound variables. One solution would be to allow each negative membership check to also be type-flaw resistant if all free variables of the check have atomic types, which is similar to the requirements for inequalities. However, in that case we would also need to require that variables occurring in insertions have atomic type. Otherwise translation *tr* of negative membership checks may not preserve type-flaw resistance. To avoid having such restrictions on the variables occurring in insertions we instead solve the problem in a manner similar to how we support private functions in the typed model: We consider each occurrence of a set expression term $f(t_1, \ldots, t_n)$ (e.g., $\mathsf{ring}(U)$, $\mathsf{valid}(U)$, and $\mathsf{revoked}(U)$ in the keyserver example) to be syntactic sugar for the expression $f'(\mathsf{sec}, t_1, \ldots, t_n)$ where f' here has arity $n + 1$ and sec is some distinguished constant.

Example 4.2 *As an example of type-flaw resistance we show that the keyserver protocol is type-flaw resistant. One approach to proving type-flaw resistance of a protocol $\mathcal{P}$ is to first find a set of strand steps M that subsumes the steps of $\mathcal{P}$ as well-typed instances. By proving type-flaw resistance of all steps in M, and of the set of terms occurring in M, we can conclude that $\mathcal{P}$ must be type-flaw resistant. For our example we can consider the following set, where $\Gamma(\{A, S, U\}) = \{\mathsf{Agent}\}$ and $\Gamma(PK) = \mathsf{Value}$:*

$$\begin{array}{rcl} M & = & \{\mathsf{assert}(\mathsf{attack}), \mathsf{delete}(PK, \mathsf{valid}(U)), \\ & & \forall A.\ PK \mathrel{\dot\notin} \mathsf{revoked}(A), \forall A.\ PK \mathrel{\dot\notin} \mathsf{valid}(A), \\ & & \mathsf{insert}(PK, \mathsf{valid}(U)), \mathsf{insert}(PK, \mathsf{ring}(U)), \\ & & \mathsf{insert}(PK, \mathsf{revoked}(U)), PK \mathrel{\dot\in} \mathsf{valid}(U), PK \mathrel{\dot\in} \mathsf{ring}(U), \\ & & \mathsf{receive}(\mathsf{inv}(PK)), \mathsf{receive}(\mathsf{sign}(\mathsf{inv}(PK), \langle U, PK\rangle)), \\ & & \mathsf{send}(\mathsf{inv}(PK)), \mathsf{send}(PK), \mathsf{send}(\mathsf{sign}(\mathsf{inv}(PK), \langle U, PK\rangle))\} \end{array}$$

Hence all variables have atomic type and so the non-constant, non-variable sub-message patterns of M consist of the composed terms and subterms closed under well-typed variable renaming and well-typed instantiation of the variables with constants. It is easy to see that each pair of non-variable terms among these composed sub-message patterns have the same type if they are unifiable. Thus the total set of terms of the protocol—and in each strand—is type-flaw resistant.

What remains to be shown is that each strand step in M satisfies requirement 2(c) of Definition 4.12 (the remaining requirement, 2(b), is trivially satisfied). The only event step occurring in M is $\mathsf{assert}(\mathsf{attack})$, which is syntactic sugar for an insertion operation, and so there is nothing that needs to be shown for the event steps. For the remaining set operations we only need to show that the negative checks satisfy type-flaw resistance. Recall that we encode set expression terms like $\mathsf{valid}(A)$ as the term $\mathsf{valid}'(\mathsf{sec}, A)$. With that in mind it is easy to see that all negative set membership checks satisfy type-flaw resistance: The terms $(PK, \mathsf{valid}'(\mathsf{sec}, A))$ and $(PK, \mathsf{revoked}'(\mathsf{sec}, A))$ are the only set expression terms occurring in the negative membership checks of M and they do not have any subterm of the form $f(x_1, \ldots, x_n)$ for some function symbol f and variables x_i. Thus the final requirement 2(c) is satisfied.

In general, type-flaw resistance is in our opinion a reasonable property to require from protocols and their implementations: Most importantly one should not have messages that encrypt raw data, like a nonce or a key, without any bit of information what the data means, because this opens the door for the intruder to reuse messages from honest agents that he cannot produce himself (and whose precise content he may not even know) in a different context. In fact, most concrete implementations satisfy this. Our result extends previous typing results in the scope of protocols that can be considered to stateful protocols; the type-flaw resistance requirement is thus also extended accordingly, however this is in some sense also conservative: all protocols that are type-flaw resistant according to the notion of Definition 3.17 are also type-flaw resistant according to our Definition 4.12. In a nutshell, the additional requirements for set operations and events are simply to exclude that sets and events can be used as an "unchecked side-channel" where type-flaws attacks can creep in. The requirements on set operations are, in fact, only as strict as the requirements on inequalities and the tuples $(\cdot,\cdot)$ that arise in the translation tr. In particular, we support arbitrary types for set elements—the only restrictions being that negative set-membership checks do not contain subterms of the form $f(x_1,\dots,x_n)$, where the x_i are bound variables, and that unifiable set elements in the same set have the same type. Thus we support set elements of atomic types, composed types, and even non-homogeneous sets (i.e., sets containing elements of different types).

Finally, we prove that reachable constraints $\mathcal{A}$, and their translations $tr(\mathcal{A})$, are type-flaw resistant whenever the initial protocol is:

Lemma 4.13 (Type-flaw resistance preservation) [3] *If $\mathcal{P}_0$ is a type-flaw resistant protocol and $(\mathcal{P}_0; 0) \overset{w}{\Longrightarrow}^{\bullet *} (\mathcal{P}; \mathcal{A})$ then both $\mathcal{P}$ and $\mathcal{A}$ are type-flaw resistant. Moreover, if $\mathcal{A}' \in tr(\mathcal{A})$ then $\mathcal{A}'$ is also type-flaw resistant.*

4.4.2 The Typing Result

All that remains is to prove the actual typing result for stateful protocols. Recall that we have formalized a typing result in Chapter 3 (more specifically Corollary 3.28) that we can use to obtain well-typed models of ordinary constraints. Note that all constraints and protocols we consider in this chapter (and the following chapter) are by definition analysis-invariant because of the requirement Ana_4' on the analysis interface Ana. Restated with the new notions introduced in this chapter the corollary is thus as follows:

Corollary 3.28 (Typing result: ordinary constraints) *If $\mathcal{A}$ is a well-formed and ordinary constraint, $\mathcal{I} \models_s \mathcal{A}$, and $\mathcal{A}$ is type-flaw resistant, then there exists a well-typed interpretation $\mathcal{I}_\tau$ such that $\mathcal{I}_\tau \models_s \mathcal{A}$.*

[3] Type-flaw resistance preservation for protocols is not proven in Isabelle. Type-flaw resistance preservation of the reduction tr is fully formalized in Isabelle.

By using our reduction tr together with Corollary 3.28 on ordinary constraints we can prove the following:

Theorem 4.14 (Typing result: symbolic constraints) *If $\mathcal{A}$ is a well-formed constraint, $\mathcal{I} \models_s \mathcal{A}$, and $\mathcal{A}$ is type-flaw resistant, then there exists a well-typed interpretation $\mathcal{I}_\tau$ such that $\mathcal{I}_\tau \models_s \mathcal{A}$.*

Proof. From Theorem 4.11, Lemma 4.13(1), and the assumptions we can obtain a type-flaw resistant ordinary constraint $\mathcal{A}'$ such that $\mathcal{A}' \in tr(\mathcal{A})$ and $\mathcal{I} \models_s \mathcal{A}$. Hence, we can obtain a well-typed interpretation $\mathcal{I}_\tau$ such that $\mathcal{I}_\tau \models_s \mathcal{A}'$ by Corollary 3.28. By applying Theorem 4.11 again we can conclude the proof. □

With this intermediate result we can prove the main result of the chapter:

Theorem 4.15 (Typing result: stateful protocols) [4] *If $\mathcal{P}_0$ is a type-flaw resistant protocol, and*

$$(\mathcal{P}_0; \emptyset, \emptyset) \overset{w}{\Longrightarrow}{}^* (\mathcal{P}; M, D) \text{ where } w = (\sigma_1, \mathcal{S}_1), \ldots, (\sigma_k, \mathcal{S}_k)$$

then there exists a state $(\mathcal{P}; M', D')$ such that

$$(\mathcal{P}_0; \emptyset, \emptyset) \overset{w'}{\Longrightarrow}{}^* (\mathcal{P}; M', D') \text{ where } w' = (\sigma'_1, \mathcal{S}_1), \ldots, (\sigma'_k, \mathcal{S}_k)$$

for some well-typed ground substitutions $\sigma'_1, \ldots, \sigma'_k$.

Proof. By using the equivalence between the ground and the symbolic transition system (Theorem 4.9) we need only to show that reachable constraints in the symbolic transition system have well-typed models. Since reachable constraints for type-flaw resistant protocols are also type-flaw resistant (Lemma 4.13(2)) we only need to apply Theorem 4.14 to the reachable constraints. Thus we obtain the desired result. □

Declarations:
ik : pred (untyped)
X, n : nonce
Y : f(nonce)
attack : pred ()

Initial state:
ik(n)

Transition rules:
ik(Y). $\neg\exists X : Y = f(X) \Rightarrow$ attack()

Horn clauses:
$\forall X$: ik(X) $\rightarrow$ ik($f(X)$)

Figure 4.1: A TASLan specification that illustrates the flaw in [Möd12].

4.4.3 A Mistake in a Related Work

Typing for stateful systems has also been considered in [Möd12]; some of its theorems have only proof sketches. Rigorously formalizing however the lazy intruder and the typing result (parts of which are now formalized in Isabelle), we have discovered several significant mistakes and we can demonstrate with counter-examples, that the result of [Möd12] does not hold in this generality as we explain in detail now.

[Möd12] allows a quite general specification of the intruder by a set of Horn clauses. There are restrictions of the form of these Horn clauses [Möd12, Sec. 2.1]: each clause expresses either that the intruder can generate new terms by applying a function symbol to known terms (this corresponds to public function symbols in our work) or how the intruder can analyze terms, somewhat corresponding to our specification of Ana. For that, the requirement on the term obtained by the analysis is only that it must be a proper subterm of the term being analyzed. This allows for instance for the following Horn clause (where the predicate ik represents messages known by the intruder):

$$\mathsf{ik}(f(g(x)) \rightarrow \mathsf{ik}(x)$$

Suppose now f is a public function and the intruder knows $g(s)$ for a secret s. Then he can with the above rule apply f to $g(s)$ to obtain s. Such a step is however not covered by the constraint reduction procedure in [Möd12], since analysis steps can only be applied to terms that the intruder directly knows, not ones he has to first compose. Now this leads to a counter-example for the typing

[4]Not proven in Isabelle.

result if we assume that f is not a public symbol, but there is an honest strand $\mathsf{receive}(x).\mathsf{send}(f(x))$ with variable x an atomic type, say, nonce. If the intruder knows $g(s)$ and s is a secret, then there is an ill-typed attack with $x = g(s)$, but no well-typed attack.

There is a second problem that there is no restriction on the type of universally quantified variables in [Möd12]. Indeed composed-typed variables can also break the typing result as we have shown before. For instance, consider the inequality $\forall x.\ y \not\doteq f(x)$ where $\Gamma(y) = f(\Gamma(x))$. For any instance $f(c)$ of y there is an instance of x (namely c) that does not satisfy the inequality. Hence the constraint has no well-typed solution. However, there does exist ill-typed solutions; since we are working in the free algebra terms are equal iff they are syntactically equal, and hence any instance of x that are not of the form $f(c)$ for some c would be a solution to the inequality. The ASLan specification in Figure 4.1 demonstrates this issue. Here the attack predicate cannot be derived if Y is instantiated with a well-typed instance in the transition rule.

It turns out that the result from the present chapter is sufficient to fix the mistakes of [Möd12] by applying the same restrictions on the intruder deduction and on composed-typed variables. A fixed version of [Möd12], highlighting the changes, is available at

https://people.compute.dtu.dk/samo/taslanv3.pdf

4.5 Case Studies

In this section we discuss how our typing result is applicable in practice on several protocols, in particular that many protocols already satisfy the requirements of type-flaw resistance or require only minor changes to do so.

As for examples of stateful verification, we consider the examples from the AIF and AIF-ω tools, since this is the closest match to our formalization, as discussed in Section 4.6.1. Note that some of the examples have also similarly been considered in SAPIC, in particular PKCS#11 and ASW.

4.5.1 Automatically Checking Type-Flaw Resistance

One crucial point of the typing result is that it is relatively easy to check, namely by statically looking at the format of messages rather than traversing the entire state space, and that this can also be done automatically as a static analysis of a user's specification before verification in a typed model.

Note that $SMP(M)$ is in general infinite, but it is sufficient to check the following finite representation SMP_0 for type-flaw resistance: starting with $SMP_0 = M$, we first ensure that for every message $t \in SMP_0$ that contains a variable x of a composed type $f(\tau_1, \ldots, \tau_n)$, we ensure that also $[x \mapsto f(x_1, \ldots, x_n)](t) \in SMP_0$ for some variables $x_1 : \tau_1, \ldots, x_n : \tau_n$ that do not occur in t. (Even if some τ_i are themselves composed types, this can be done by adding finitely many messages, since all type expressions are finite terms.) Next, we close SMP_0 under subterms and key terms of Ana. Finally, let us ensure by well-typed α-renaming that all terms in SMP_0 have pairwise disjoint variables. Note that SMP_0 is a representation of $SMP(M)$ in the sense that every $SMP(M)$ term is a well-typed instance of an SMP_0 term. Now the condition that every pair $s, t \in SMP_0 \setminus \mathcal{V}$ with $\Gamma(s) \neq \Gamma(t)$ has no unifier, is equivalent to the type-flaw resistance of M:

Lemma 4.16 *M is type-flaw resistant if and only if*

$$\forall s, t \in SMP_0 \setminus \mathcal{V}.\ (\exists \delta.\ \delta(s) = \delta(t)) \longrightarrow \Gamma(s) = \Gamma(t)$$

Proof. Note that $SMP_0 \subseteq SMP(M)$, giving us one direction of the equivalence. For the other direction, suppose there are any $s, t \in SMP(M) \setminus \mathcal{V}$ such that $\Gamma(s) \neq \Gamma(t)$. We need to show that s and t are not unifiable. Since SMP_0 represents $SMP(M)$, there exists terms $s_0, t_0 \in SMP_0$ and well-typed substitutions θ_1 and θ_2 such that $s = \theta_1(s_0)$ and $t = \theta_2(t_0)$. Hence also $\Gamma(s_0) \neq \Gamma(t_0)$, and so s_0 and t_0 are not unifiable by assumption. Thus s and t cannot be unified either because s_0 and t_0 do not share variables. □

Note that for protocols with an infinite number of strands the initial set M should be chosen carefully to prevent an infinite SMP_0. In our keyserver example, for instance, we can choose for M a more general and finite set where all terms and set operations of the protocol are well-typed instances of terms in M—such as in the type-flaw resistance example of Section 4.4.1. This is sufficient to ensure finiteness of SMP_0.

4.5.2 Extension of the Keyserver Example

We will now illustrate by a small example how type-flaw problems can arise in practice, how type-flaw resistance is violated in such a case, and how the situation can be fixed. Suppose for the keyserver example, we augment the protocol with an exchange where a user can prove to be alive, formalized by having for each user u and each session $j \in \mathbb{N}$ the following transaction strand:

$$\mathsf{receive}(N_j).PK_{u,j} \mathrel{\dot{\in}} \mathsf{ring}(u).\mathsf{send}(\mathsf{sign}(\mathsf{inv}(PK_{u,j}), N_j))$$

where all N_j and $PK_{u,j}$ have atomic types. The idea is that anybody can send the user a challenge N_j, and u answers with a signature on it. In this blunt form it is obviously a bad idea, since an intruder can send an arbitrary term instead of N_j. Indeed the protocol now violates type-flaw resistance: $\mathsf{sign}(\mathsf{inv}(PK_{u,j}), N_j)$ has a unifier with the normal update message $\mathsf{sign}(\mathsf{inv}(PK_i), \langle U_i, NPK_i\rangle)$, while they have different types. The general recommendation is thus to use some form of tag to indicate what the messages should mean. In fact, many protocol standards already describe a concrete message format, e.g., in this case that nonces and public keys have certain byte lengths, or even fields that indicate the length, if it is not fixed; in contrast many protocol models model only abstractly the exchanged information as tuples. It is thus recommended to model the concrete message formats by transparent functions, i.e., functions like pair that the intruder can compose and decompose, and check that the concrete formats of the protocol standard are disjoint so that a confusion is impossible. In this case we may have functions $\mathsf{update}(U, PK)$ that is used in the update message and functions $\mathsf{challenge}(N)$ and $\mathsf{response}(N)$ to model the challenge response protocol to have rather the following form:

$$\mathsf{receive}(\mathsf{challenge}(N_j)).PK_{u,j} \mathrel{\dot{\in}} \mathsf{ring}(u).\mathsf{send}(\mathsf{sign}(\mathsf{inv}(PK_{u,j}), \mathsf{response}(N_j)))$$

One may argue that the formatting of the challenge message is irrelevant since it is in cleartext. We suggest, however, to use formatting information also here, since it is in fact good practice for implementations anyway and does not really hurt.

With the change we now have again type-flaw resistance and our typing result is applicable.

4.5.3 Secure Vehicle Communication

A set of examples for AIF is a model of the secure vehicle communication of the SEVECOM project [SEV09, MM11]. These define a setup for hardware security modules in cars that store a number of keys that can only be used via a number of API commands. A main concern is the so-called root key update. Here we have the following message patterns of incoming and outgoing messages, where the variables K, $K1$, and $K2$ are of type Value: K, $\mathsf{sign}(\mathsf{inv}(K), K)$, and $\mathsf{sign}(\mathsf{inv}(K2), \mathsf{pair}(K1, K))$, where we have omitted some message patterns that can be obtained by well-typed substitutions. The corresponding set of sub-message patterns

$$SMP(\{K, \mathsf{sign}(\mathsf{inv}(K), K), \mathsf{sign}(\mathsf{inv}(K2), \mathsf{pair}(K1, K))\})$$

is the closure of the message patterns under subterms, term decomposition, and well-typed instantiation.

This is not directly type-flaw resistant: if we first consider a well-typed renaming of the first signature, say, $\mathsf{sign}(\mathsf{inv}(K3), K3)$ then there is a unifier with the other signature $\mathsf{sign}(\mathsf{inv}(K2), \mathsf{pair}(K1, K))$, namely identifying $K3$, $K2$, and $\mathsf{pair}(K1, K)$. Indeed the two signature messages here have different type and meaning (the first means key revocation, the second means key update), while they have nothing signaling in the signed text which message it is. Indeed, if we look at the standard [SEV09], it requires that the revocation and the update signature contain specific text namely "REVOKE ROOT PUBLIC KEY" and "LOAD ROOT PUBLIC KEY". This had not been modeled in [MM11]. Again, we recommend to use here transparent functions $\mathsf{revoke}/1$ and $\mathsf{update}/2$, to model the format of the revoke and update messages, respectively, and for which the intruder can directly extract the arguments, i.e., $\mathsf{Ana}(\mathsf{revoke}(K)) = (\emptyset, \{K\})$ and $\mathsf{Ana}(\mathsf{update}(K1, K2)) = (\emptyset, \{K1, K2\})$. Then we have as SMP the following set closed under subterms and well-typed substitutions (in this case study closing under term decomposition is unnecessary as it is subsumed by closing under subterms):

$$\{K, \mathsf{sign}(\mathsf{inv}(K), \mathsf{revoke}(K)), \mathsf{sign}(\mathsf{inv}(K2), \mathsf{update}(K1, K))\}$$

and indeed now type-flaw resistance is satisfied.

4.5.4 PKCS#11

The examples of AIF-ω contain a number of specifications of PKCS#11-based APIs following [FS09, BCFS10]. Again the model of a crypto-device is here by a number of transactions that consist of a command and arguments to the device, which performs some checks, possibly generates some encryptions and makes some notes and sends an output as a result. The question is if an intruder can obtain something by combining several API calls in a way that had not been anticipated. Again, the AIF-ω model of these calls is based on sets for describing the different flags associated to a key (e.g., whether it is a key that can be extracted). The specification is again that all elements of the sets are declared to have type Value, thus it only remains to check that the messages input and output to the device fulfill the type-flaw resistance. Since the commands themselves are not encrypted, the AIF models do not model opcodes and the like and just present the bare arguments (e.g., key-handles, encrypted messages etc.) to the device. We then obtain the following kinds of messages:

$$\mathsf{bind}(N, K, K), \quad \mathsf{h}(N, K), \quad K, \quad \mathsf{senc}(M, K)$$

Here K, N, and M are of type Value, and we have omitted some terms that are redundant under well-typed substitution again. Also this is type-flaw resistant, however there is an interesting point. As long as only the intruder is interacting with the token interface, the type-flaw resistance is guaranteeing that he has no gain from using ill-typed messages. However, when we consider extensions of these examples (e.g., a richer API or a network with other tokens or honest parties), then also more complex messages M in the symmetric encryption may be produced (or received by honest agents) and the type-flaw resistance breaks. It therefore seems like a good idea to not have raw encryptions of a key (like in $\mathsf{senc}(M, K)$) but to insert some more information into the encrypted message, like a format as in the SEVECOM example above. Indeed this solves some of the attacks that arise in the use of the API already, when we use different such formats (or tags) for keys of different intended use (e.g., wrap-unwrap attacks).

4.5.5 ASW

The fair contract signing protocol ASW is another example of a protocol that necessarily requires a global state. With AIF shipped a formalization of ASW that abbreviates some protocol messages drastically, for instance the function $\mathsf{msg1}(A, B, \mathsf{contract}(A, B), \mathsf{h}(NA))$ to abbreviate a message from A to B that is actually signed with the private key of A, and intruder rules that allow the intruder to compose such a $\mathsf{msg1}$-message if A is dishonest (and always decompose it). To use it with our typing result and the Ana functions, we need to use a more standard model, explicitly denoting the signature function, i.e., for $\mathsf{msg1}$ we rather have $\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(A)), \mathsf{m1}(A, B, \mathsf{contract}(A, B), \mathsf{h}(NA)))$ where $\mathsf{m1}$ is a transparent function to model the concrete format of the message content.

Note that when this message is received by B, it has the form

$$\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(A)), \mathsf{m1}(A, B, \mathsf{contract}(A, B), \mathit{HNA}))$$

with a variable HNA of the composed type $\mathsf{h}(\mathsf{nonce})$, since B cannot check at this point that this is indeed a hash of a nonce as it should be. The entire point of this fair exchange is in fact that the nonces are revealed only later.

The second message has the form $\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(B)), \mathsf{m2}(B, t, \mathsf{h}(NB)))$ where $t = \mathsf{sign}(\mathsf{inv}(\mathsf{pk}(A)), \mathsf{m1}(A, B, \mathsf{contract}(A, B), \mathit{HNA})$, i.e., the t is a message of the first form; note that here the variables A and B must all agree in these forms since this is part of what the participants check. When A receives this message, it has the form $\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(B)), \mathsf{m2}(B, t', \mathit{HNB}))$ with a composed-type variable HNB since A similarly cannot check that this is really a hash of a nonce. In contrast t' has the form $\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(A)), \mathsf{m1}(A, B, \mathsf{contract}(A, B), \mathsf{h}(\mathit{NA})))$ since here the nonce NA has been created by A herself earlier.

Messages three and four of the protocol are simply the nonces NA and NB; even though we suggest not to have such raw data sent around (and rather wrap it in another transparent format), this is not a problem with type-flaw resistance.

Note that for that part of the verification we have now the equations $\mathit{HNA} = \mathsf{h}(\mathit{NA})$ and $\mathit{HNB} = \mathsf{h}(\mathit{NB})$ since after receiving the nonce from each other, the agents should check out with the respective HNA and HNB received earlier. Note also that if there was a continuation for the case that such a check fails, it could not be handled by our typing result, because that would imply composed-typed variables in inequalities.

The most interesting part of ASW is the communication with a server in case the above four-step contract signing goes wrong, i.e., if one of the agents does not receive an answer anymore, in particular if B has received message three from A and thus has a valid contract, and dishonestly refuses to reveal the final message four to A, so A does not have a contract. The protocol assumes that both agents have resilient channels to a trusted third party, i.e., they eventually get an answer. If A did not receive an answer to her message one, she can send an abort message to the server of the form $\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(A)), \mathsf{abortReq}(t))$ where t is the first message she had sent. If A or B at a later point in the protocol (i.e., after at least sending/receiving message two) do not obtain an answer, they can ask for a resolve, which is of the form $\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(X)), \mathsf{resolveReg}(t_1, t_2))$ where t_1 and t_2 are the first two messages of the protocol and X is the agent A or B asking for the resolve. The server should now look in his database of contracts, and if the contract does not occur in the database yet, grant the abort or resolve request, by the messages $\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(s)), \mathsf{abort}(t))$ or $\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(s)), \mathsf{resolve}(t_1, t_2))$, respectively, where $\mathsf{inv}(\mathsf{pk}(s))$ is the private key of the server. The result is of course also stored in the database, and this entry will be the reply to any agent who asks for an abort or resolve of that contract.

The AIF model has here several limitations: since resilient channels cannot be modeled directly, it models the interaction between users and servers as atomic transitions. The assumption of the real protocol is a bit weaker: an intruder cannot entirely block a request or the response, but he may be able to delay it, for instance observe a request and send a different request that arrives earlier at the server. Also the messages exchanged are not modeled, but only the effects on the users and servers database. We have thus here checked type-flaw resistance both for the restricted model that comes with AIF and for an extended model that includes all necessary steps and possible interleavings.

The server's database is actually modeled as a family of sets $\mathsf{scondb}(A, B, \mathit{Status})$ for each agent A, B and where Status is either valid or $\mathsf{aborted}$. However, instead of the contract, it stores only the nonce NA. This is due to AIF's limitations to sets of constants. It is sufficient to make a working model of ASW, since NA is sufficient to identify the concrete exchange.

In fact, satisfaction of the type-flaw resistance is easy to see, since every function symbol except sign is applied in all messages to terms of the same types and the message being signed is never directly a variable. Similar, for the sets, the contents have all type nonce, and the set terms have the form $\mathit{family}(A, B, \mathit{Status})$ where A and B are agents and Status ranges over a set of possible status messages.

4.6 Connections to Other Formalisms

We have introduced the formalism of transaction strands to have a simple and mathematically pure formalism as a protocol model for our result without the disturbance of the many technical details of various protocol models. We want to illustrate now that our result can nonetheless be used in various protocol models, but we only sketch the main ideas and discuss also limitations of our typing results.

Note that the core of our result is proved on symbolic constraints (intruder strands) of a symbolic transition system. Connecting another formalism with our typing result requires only two aspects. First, one needs to define the semantics for the formalism in terms of a symbolic transition system with constraints (including set operations, equalities, and inequalities). Second, one needs to transfer the notion of type-flaw resistance, so that a type-flaw resistant specification in the formalism will only produce type-flaw resistant constraints. We have done this for transaction strands with detailed proofs. Due to the variety of other formalisms and their technical details, we only sketch in the following the ideas for the most common constructions.

4.6.1 AIF-ω and Rewriting

Our transaction strands are in some sense a purified version of AIF-ω. In a nutshell, it describes protocols by a set of rewrite rules for a state transition system, where each state is a set of *facts* like $\mathsf{ik}(m)$ to denote that the intruder knows message m. It is thus also similar to other rewriting based languages like Maude-NPA or the AVANTSSAR ASLan.

One can translate each AIF-ω rule into transaction strands as follows. Every intruder knowledge fact $\mathsf{ik}(m)$ on the left hand side of a rule corresponds to receiving a message m, and on the right hand side to sending a message m. If the expression $t \ \mathsf{in}\ s$ occurs on the left-hand side, then the transaction strand must contain $t \mathrel{\dot{\in}} s$; if the same expression does not occur on the right-hand side, then the transaction must include $\mathsf{delete}(t, s)$. If the expression $t\ \mathsf{notin}\ s$ occurs on the left-hand side, then the transaction must contain $\forall \bar{x}.\ t \mathrel{\dot{\notin}} s$ where $\bar{x}$ are the variables that on the left-hand side only occur in notin expressions. Finally, if $t\ \mathsf{in}\ s$ occurs on the right-hand side but not on the left, then the transaction must include $\mathsf{insert}(t, s)$. All other facts of AIF-ω are persistent (i.e., once true, they remain true in all successor states), therefore we can model them as events in transaction strands, using $\mathsf{event}(e)$ for the left-hand side facts and $\mathsf{assert}(e)$ for right-hand side facts. Note that the order of all these actions in the transaction matters: first we should have all receiving messages, checking for events and set memberships, then modifying sets and sending the outgoing messages. Still one may wonder what happens in the following AIF-ω rule: $x\ \mathsf{in}\ s.y\ \mathsf{in}\ s \Rightarrow x\ \mathsf{in}\ s$. If $x = y$ then this rule is contradictory, and the semantics of AIF-ω excludes such substitutions. For that reason, we also have to include the inequality $x \neq y$ to the transaction to exactly follow the AIF-ω semantics. In all remaining cases the inner order of the actions is actually irrelevant, but these subtle points were one of the motivations to introduce transaction strands. Finally, note that the rules from AIF-ω may have variables that represent any value from a countable set of constants, as well as the creation of fresh values. Since transaction strands do not have a mechanism for creating fresh values and free variables are not allowed, one must instantiate these variables appropriately, producing a countable set of transaction strands from finitely many rules.

With this translation from AIF-ω rules to transaction strands, we also directly obtain a semantics using symbolic constraints and actually immediately transfer the notion of type-flaw resistance from transaction strands with the obvious adaptations. However, type-flaw resistance will not be directly satisfied for typical AIF-ω specifications immediately, because they would contain rules for the intruder that contain untyped variables. While for honest agents, it is not a restriction to declare the *intended* type for each variable, the intruder deduction rules should be applicable to messages of *any* type. Thus, we have to make the reservation that the intruder deduction of an AIF-ω specification must be within the bounds of the intruder model we have used here, namely composition with public functions and decomposition according to an Ana theory. This is indeed possible for all the standard operators like symmetric and asymmetric encryption, signatures, hashes, and transparent functions like pair; operators that require algebraic equations like xor are however not supported. We come back to this when discussing process calculi and reduction rules below.

Finally, note that other rewriting based formalisms like [CDL05, EMM07] (or the closely related linear logic rules) are not based on sets, but usually multi-sets of facts, and they are not persistent, i.e., facts can be removed by transitions, which cannot directly be modeled by our notion of events in transaction strands. There is however a way to encode this using sets: for each fact where we want to encode non-persistent behavior, we introduce a corresponding event with one more argument. For this argument we use a fresh constant whenever a fact is introduced by a transition and the argument becomes member of a special set active. Whenever the fact shall be removed, we simply remove the corresponding constant from the set active. This allows for modeling both the multi-set aspect as well as the non-persistent aspect.

4.6.2 Set-π and Process Calculi

Process calculi are a very popular way of specifying protocols. While they can immediately describe stateful systems (due to Turing completeness), this is usually not at a level that directly works well with existing verification methods. Therefore several extensions have been proposed, namely Set-π for set operations similar to AIF-ω, and SAPIC for adding a notion of maps. One gap to the rewriting formalisms above is that process calculi do not have the notion of an atomic transaction. Therefore both Set-π and SAPIC rely on the use of locks, i.e., in order to read and write on a set or (an element of) a map, one has to first lock it, and no other process can get a lock on the same item before it is unlocked. It is possible to give a translation to transaction strands, modeling explicitly the locks by an additional set that stores which of the other sets are locked. However, it is a bit more convenient to directly give a semantics as a symbolic transition system, i.e., producing symbolic constraints in each execution.

However, before we can do that, there is another obstacle to overcome: it is standard to model in process calculi decryption and checking of messages explicitly by a let construct and reduction rules. For instance if the public function crypt represents asymmetric encryption and inv the private function that maps from public to private keys, for decryption one would introduce a new operator dcrypt and have the reduction rule $\mathsf{dcrypt}(\mathsf{crypt}(x, y), \mathsf{inv}(x)) \to y$. Then receiving and decrypting a message for instance would be $\mathsf{in}(u).\mathsf{let}\ v = \mathsf{dcrypt}(u, \mathsf{inv}(k))\ \mathsf{in}\ P\ \mathsf{else}\ Q$. Thus process P is executed if the received message u is indeed encrypted with k (and binding v to the content of that message), otherwise Q is executed. Note that the destructor dcrypt does not occur as part of "normal" messages.

Our typing result can only support such destructors if we can express such decryption operations using an Ana theory. In the example we would have $\mathsf{Ana}(\mathsf{crypt}(x,y)) = (\{\mathsf{inv}(x)\},\{y\})$ and we would translate the above example process into $\mathsf{in}(u).\ \mathsf{if}\ (\mathsf{crypt}(k,?v) \doteq u)\ \mathsf{then}\ P\ \mathsf{else}\ Q$. Note that here we have actually made an extension of Set-π, namely adding the concept of equalities from transaction strands to the if construct, including that newly introduced variables on the left-hand side are binding, here v, and we mark this by a question mark as is standard. This is formally defined by the symbolic semantics below.

Besides destructors, process calculi also commonly use reduction rules for checks on messages, e.g., $\mathsf{verify}(\mathsf{sign}(\mathsf{inv}(x),y),x) \to \mathsf{true}$ that can be used to verify a signature, for instance: $\mathsf{in}(u).\ \mathsf{let}\ \mathsf{true} = \mathsf{verify}(u,k)\ \mathsf{in}\ P\ \mathsf{else}\ Q$. For this, we do not need to have a corresponding line in Ana, rather we can model this directly by an equality: $\mathsf{in}(u).\ \mathsf{if}\ \mathsf{sign}(\mathsf{inv}(k),?z) \doteq u\ \mathsf{then}\ P\ \mathsf{else}\ Q$. With this, all the standard operators can be supported, except those that require algebraic equations like xor.

If we now assume Set-π without let but instead with equations in if, we can define its semantics as a symbolic transition system as in Figure 4.2 (using notation and labels similar to the original ground semantics) where $\alpha(P)$ is a fresh renaming of all variables in P that are bound by an in statement, and the translation $\mathit{ctr}(b)$ of a condition b is defined as follows. Recall that we had used the notion of binding occurrences also in equations (and logically this can also be done in set membership checks) and marked the respective occurrence by a question mark, like $?x = \ldots$. Let in the following $\bar{x}$ be the set of variables of a condition that are marked with the question mark:

$$\begin{array}{rclcrcl} \mathit{ctr}(s \doteq t) & = & s \doteq t & \quad & \mathit{ctr}(s \not\doteq t) & = & \forall \bar{x}.\ s \not\doteq t \\ \mathit{ctr}(t \mathrel{\dot\in} s) & = & t \mathrel{\dot\in} s & \quad & \mathit{ctr}(t \mathrel{\dot\notin} s) & = & \forall \bar{x}.\ t \mathrel{\dot\notin} s \end{array}$$

Note that the locking is not checked upon set operations, as this is done statically in Set-π. Since in general sets can have terms with variables, we have formulated the check as inequalities in the LCK rule.

In order to check type-flaw resistance, one now needs to consider the translation from let-statements into equations (which can be done transparent to the user) and then the type-flaw resistance property is almost as before, only we need to consider each condition positively as well as its negation (unless the else case is empty).

NIL:	$\mathcal{P} \uplus \{(0, \emptyset)\}, \mathcal{A} \to \mathcal{P}, \mathcal{A}$
COM$_1$:	$\mathcal{P} \uplus \{(\mathsf{in}(x).P_1, L_1)\}, \mathcal{A} \to \mathcal{P} \uplus \{(P_1, L_1)\}, \mathcal{A}.\mathsf{send}(x)$
COM$_2$:	$\mathcal{P} \uplus \{(\mathsf{out}(N).P_2, L_2)\}, \mathcal{A} \to \mathcal{P} \uplus \{(P_2, L_2)\}, \mathcal{A}.\mathsf{receive}(N)$
PAR:	$\mathcal{P} \uplus \{(P_1 \mid P_2, \emptyset)\}, \mathcal{A} \to \mathcal{P} \uplus \{(P_1, \emptyset), (P_2, \emptyset)\}, \mathcal{A}$
REPL:	$\mathcal{P} \uplus \{(!P, \emptyset)\}, \mathcal{A} \to \mathcal{P} \uplus \{(\alpha(P) \mid !P, \emptyset)\}, \mathcal{A}$
NEW:	$\mathcal{P} \uplus \{(\mathsf{new}\ x.P, L)\}, \mathcal{A} \to \mathcal{P} \uplus \{(P[x \mapsto c], L)\}, \mathcal{A}$
	for some fresh name c
IF1:	$\mathcal{P} \uplus \{(\mathsf{if}\ b\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2, L)\}, \mathcal{A} \to \mathcal{P} \uplus \{(P_1, L)\}, \mathcal{A}.ctr(b)$
IF2:	$\mathcal{P} \uplus \{(\mathsf{if}\ b\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2, L)\}, \mathcal{A} \to \mathcal{P} \uplus \{(P_2, L)\}, \mathcal{A}.ctr(\neg b)$
SET$_+$:	$\mathcal{P} \uplus \{(\mathsf{insert}(t, s).P, L)\}, \mathcal{A} \to \mathcal{P} \uplus \{(P, L)\}, \mathcal{A}.\mathsf{insert}(t, s)$
SET$_-$:	$\mathcal{P} \uplus \{(\mathsf{delete}(t, s).P, L)\}, \mathcal{A} \to \mathcal{P} \uplus \{(P, L)\}, \mathcal{A}.\mathsf{delete}(t, s)$
LCK:	$\mathcal{P} \uplus \{(\mathsf{lock}(l).P, L)\}, \mathcal{A} \to \mathcal{P} \uplus \{(P, \{l\} \cup L)\}, \mathcal{A}.l \not\doteq l_1. \cdots .l \not\doteq l_n$
	where $\{l_1, \ldots, l_n\} = L \cup \bigcup_{(P', L') \in \mathcal{P}} L'$
ULCK:	$\mathcal{P} \uplus \{(\mathsf{unlock}(l).P, \{l\} \uplus L)\}, \mathcal{A} \to \mathcal{P} \uplus \{(P, L)\}, \mathcal{A}$

Figure 4.2: A symbolic transition system for Set-π.

4.6.3 SAPIC

Finally, let us consider the SAPIC tool that is also a process calculus, but instead of sets has a global map, i.e., one can insert key-value pairs into the map (where inserting multiple times with the same key is overwriting), delete pairs, and query what value is associated to a key.

For a restricted setting, we can indeed express this map with sets, namely if we can split the map into finitely many partitions where each key and value are of some atomic type. For instance, in the PKCS examples, the value type are actually tuples, but the second part ranges over finitely many values and thus one could represent this maps as a finite collection of maps with atomic value type.

The idea is of course to model map $m = [k_1 \mapsto v_1, \ldots, k_n \mapsto v_n]$ by a family of sets $m(\cdot)$ such that $v_1 \in m(k_1), \ldots, v_n \in m(k_n)$. Initially, all maps should contain one distinguished symbol $\bot$ to represent that for that key no value is in the map. Then to insert the tuple (k, v) translates to the set operations $x \mathrel{\dot\in} m(k).\mathsf{delete}(x, m(k)).\mathsf{insert}(v, m(k))$. To delete key k from the map is then like inserting $(k, \bot)$. Querying for key k is checking $x \mathrel{\dot\in} m(k)$ and $x \not\doteq \bot$.

4.7 Proofs

This section contains the pen-and-paper proofs of our technical results. Note that many of the theorems and lemmas are more general versions of the ones stated earlier in this chapter.

4.7.1 Constraint Semantics

Lemma 4.3. *Given a ground set of terms M, a ground database mapping D, a ground set E of asserted events, an interpretation $\mathcal{I}$, and symbolic constraints $\mathcal{A}$ and $\mathcal{A}'$, the following holds:*

$$\llbracket M, D; \mathcal{A}.\mathcal{A}' \rrbracket_s\ \mathcal{I} \textit{ if and only if}$$
$$\llbracket M, D; \mathcal{A} \rrbracket_s\ \mathcal{I} \textit{ and } \llbracket M \cup ik(\mathcal{I}(\mathcal{A})), db(\mathsf{insert}(D).\mathcal{I}(\mathcal{A})); \mathcal{A}' \rrbracket_s\ \mathcal{I}$$

PROOF. Each direction of the biconditional follows easily by an induction on the leftmost constraint $\mathcal{A}$. □

Lemma 4.4 *Given a ground state $(\mathcal{P}; M, D)$, a transaction strand $\mathcal{S} \in \mathcal{P}$ (condition C1), and a ground substitution σ with domain $fv(\mathcal{S})$ (condition C2), then the conditions C3 to C7 hold if and only if $\llbracket M, D; dual(\mathcal{S}) \rrbracket_s\ \sigma$.*

PROOF. Let $\mathcal{S} = \mathsf{receive}(T).S.\mathsf{send}(T')$ where S does not contain send and receive steps and observe that $dual(\mathcal{S}) = \mathsf{send}(T).dual(S).\mathsf{receive}(T')$. Condition C3 then corresponds to $\llbracket M, D; \mathsf{send}(T) \rrbracket_s\ \sigma$, condition C4 to C7 to $\llbracket M, D; dual(S) \rrbracket_s\ \sigma$, and the remaining part $\mathsf{receive}(T')$ is irrelevant as it is satisfied for any M, D, E, and σ. Thus C3 to C7 hold if and only if $\llbracket M, D; dual(\mathcal{S}) \rrbracket_s\ \sigma$. □

4.7.2 Transition Systems

Lemma 4.8 (Well-formedness of reachable symbolic constraints). *If $\mathcal{P}_0$ is a well-formed protocol and $(\mathcal{P}_0; 0) \overset{w}{\Longrightarrow}^{\bullet *} (\mathcal{P}; \mathcal{A})$ then $\mathcal{A}$ is a well-formed symbolic constraint and $\mathcal{P}$ is a well-formed protocol.*

PROOF. By induction on reachability. The base case (i.e., the symmetric case) is trivial. For the inductive case assume that $(\mathcal{P}_0; 0) \overset{w'}{\Longrightarrow}^{\bullet *} (\mathcal{P}; \mathcal{A}) \overset{\mathcal{S}}{\Longrightarrow}^{\bullet} (\mathcal{P} \setminus \{\mathcal{S}\}; \mathcal{A}.\mathcal{A}')$ where $\mathcal{A}$ and $\mathcal{P}$ are well-formed by the induction hypothesis. Let

$\mathcal{S} = \mathsf{receive}(T).S.\mathsf{send}(T')$, where S does not contain further receive and send steps, then $\mathcal{A}' = \mathit{dual}(\mathcal{S}) = \mathsf{send}(T).S.\mathsf{receive}(T')$ by definition. Since $\mathcal{P} \setminus \{\mathcal{S}\} \subseteq \mathcal{P}$ and $\mathcal{P}$ is well-formed we have that $\mathcal{P} \setminus \{\mathcal{S}\}$ must also be well-formed. Since all strands in $\mathcal{P}_0$ are variable-disjoint we also have that $\mathit{fv}(\mathcal{A}) \cap \mathit{fv}(\mathcal{A}') = \emptyset$. Hence $\mathcal{A}.\mathcal{A}'$ is well-formed if $\mathcal{A}$ is well-formed and $\mathcal{A}'$ is well-formed. The prefix $\mathcal{A}$ is well-formed by the induction hypothesis, and since $\mathcal{S} \in \mathcal{P}$ we know that $\mathit{dual}(\mathcal{S}) = \mathcal{A}'$ is well-formed as well. Thus we can conclude the case. $\square$

Theorem 4.9 (Equivalence of transition systems). *Let $\mathcal{P}_0$ be a protocol and let $\{\mathcal{S}_1, \dots, \mathcal{S}_k\} \subseteq \mathcal{P}_0$. Then:*

1. *If $(\mathcal{P}_0; \emptyset, \emptyset) \overset{w}{\Longrightarrow}^* (\mathcal{P}; M, D)$ where $w = (\sigma_1, \mathcal{S}_1), \dots, (\sigma_k, \mathcal{S}_k)$ then*
   (a) *$(\mathcal{P}_0; 0) \overset{w'}{\Longrightarrow}^{\bullet *} (\mathcal{P}; \mathit{dual}(\mathcal{S}_1). \cdots .\mathit{dual}(\mathcal{S}_k))$ where $w' = \mathcal{S}_1, \dots, \mathcal{S}_k$,*
   (b) *$\sigma_1 \cdot \ldots \cdot \sigma_k \models_s \mathit{dual}(\mathcal{S}_1). \cdots .\mathit{dual}(\mathcal{S}_k)$,*
   (c) *$M = \mathit{ik}((\sigma_1 \cdot \ldots \cdot \sigma_k)(\mathit{dual}(\mathcal{S}_1). \cdots .\mathit{dual}(\mathcal{S}_k)))$, and*
   (d) *$D = \mathit{db}((\sigma_1 \cdot \ldots \cdot \sigma_k)(\mathit{dual}(\mathcal{S}_1). \cdots .\mathit{dual}(\mathcal{S}_k)))$.*
2. *If $(\mathcal{P}_0; 0) \overset{w}{\Longrightarrow}^{\bullet *} (\mathcal{P}; \mathcal{A})$ and $\mathcal{I} \models_s \mathcal{A}$ where $w = \mathcal{S}_1, \dots, \mathcal{S}_k$, $\mathit{dom}(\mathcal{I}) = \mathit{fv}(\mathcal{A})$, and $\mathcal{I}$ is ground, then there exists substitutions $\sigma_1, \dots, \sigma_k$ such that*
   (a) *$(\mathcal{P}_0; \emptyset, \emptyset) \overset{w'}{\Longrightarrow}^* (\mathcal{P}; \mathit{ik}(\mathcal{I}(\mathcal{A})), \mathit{db}(\mathcal{I}(\mathcal{A})))$ where $w' = (\sigma_1, \mathcal{S}_1), \dots, (\sigma_k, \mathcal{S}_k)$,*
   (b) *$\mathcal{A} = \mathit{dual}(\mathcal{S}_1). \cdots .\mathit{dual}(\mathcal{S}_k)$,*
   (c) *$\mathit{dom}(\sigma_i) = \mathit{fv}(\mathcal{S}_i)$ for all $i \in \{1, \dots, k\}$, and*
   (d) *$\mathcal{I} = \sigma_1 \cdot \ldots \cdot \sigma_k$*

PROOF.

1. We prove the first implication by an induction on reachability.

   The base case (i.e., the symmetric case) follows easily from the assumptions.

   So, in the inductive case, assume that

$$\begin{aligned} (\mathcal{P}_0; \emptyset, \emptyset) &\overset{w}{\Longrightarrow}^* (\mathcal{P}; M, D) \\ &\overset{w_{k+1}}{\Longrightarrow} (\mathcal{P} \setminus \{\mathcal{S}_{k+1}\}; M \cup \sigma_{k+1}(T'), \mathit{db}(\mathsf{insert}(D).\sigma_{k+1}(S)) \end{aligned}$$

   where $\mathcal{S}_{k+1} = \mathsf{receive}(T).S.\mathsf{send}(T') \in \mathcal{P}$ and $w = (\sigma_1, \mathcal{S}_1), \dots, (\sigma_k, \mathcal{S}_k)$ and $w_{k+1} = (\sigma_{k+1}, \mathcal{S}_{k+1})$. We furthermore assume the induction hypothesis:

(H1) $(\mathcal{P}_0; 0) \overset{w'}{\Longrightarrow}^{\bullet *} (\mathcal{P}; \mathit{dual}(\mathcal{S}_1).\cdots.\mathit{dual}(\mathcal{S}_k))$ where $w' = \mathcal{S}_1, \ldots, \mathcal{S}_k$,

(H2) $\sigma_1 \cdot \ldots \cdot \sigma_k \models_s \mathit{dual}(\mathcal{S}_1).\cdots.\mathit{dual}(\mathcal{S}_k)$,

(H3) $M = \mathit{ik}((\sigma_1 \cdot \ldots \cdot \sigma_k)(\mathit{dual}(\mathcal{S}_1).\cdots.\mathit{dual}(\mathcal{S}_k)))$, and

(H4) $D = \mathit{db}((\sigma_1 \cdot \ldots \cdot \sigma_k)(\mathit{dual}(\mathcal{S}_1).\cdots.\mathit{dual}(\mathcal{S}_k)))$.

In the remaining proof for this case we will use the abbreviations $\mathcal{I} = \sigma_1 \cdot \ldots \cdot \sigma_{k+1}$ and $\mathcal{A} = \mathit{dual}(\mathcal{S}_1).\cdots.\mathit{dual}(\mathcal{S}_{k+1})$. We will now prove each of the five parts of the thesis for this case:

- From (H1) and $\mathcal{S}_{k+1} \in \mathcal{P}$ we can apply $\overset{\mathcal{S}_{k+1}}{\Longrightarrow}^{\bullet}$ and immediately conclude part (a) of the thesis, namely:

  $$(\mathcal{P}_0; 0) \overset{w', \mathcal{S}_{k+1}}{\Longrightarrow}^{\bullet *} \quad (\mathcal{P} \setminus \{\mathcal{S}_{k+1}\}; \mathcal{A})$$

- From the assumption and Lemma 4.4 we know that

  $$[\![M, D; \mathit{dual}(\mathcal{S}_{k+1})]\!]_s\ \sigma_{k+1}$$

  and since all strands of a protocol must be pairwise variable-disjoint we furthermore know that $(\sigma_1 \cdot \ldots \cdot \sigma_k)(\mathit{dual}(\mathcal{S}_{k+1})) = \mathit{dual}(\mathcal{S}_{k+1})$. Hence $[\![M, D; \mathit{dual}(\mathcal{S}_{k+1})]\!]_s\ \mathcal{I}$ because $\mathit{dom}(\sigma_i) \cap \mathit{dom}(\sigma_j) = \emptyset$ for all $i, j \in \{1, \ldots, k+1\}$ where $i \neq j$ since $\mathit{dom}(\sigma_i) = \mathit{fv}(\mathcal{S}_i)$ for all $i \in \{1, \ldots, k+1\}$. Likewise, we get $\mathcal{I} \models_s \mathit{dual}(\mathcal{S}_1).\cdots.\mathit{dual}(\mathcal{S}_k)$ from (H2). All together we can then conclude part (b), namely $\mathcal{I} \models_s \mathcal{A}$.

- Note that $\mathit{dual}(\mathsf{receive}(T).S.\mathsf{send}(T')) = \mathsf{send}(T).\mathit{dual}(S).\mathsf{receive}(T')$ and so $\mathit{ik}(\mathit{dual}(\mathcal{S}_{k+1})) = T'$. Together with the variable disjointness of the strands and (H3) we have:

  $$\begin{aligned}
  &M \cup \sigma_{k+1}(T') \\
  &= \mathit{ik}((\sigma_1 \cdot \ldots \cdot \sigma_k)(\mathit{dual}(\mathcal{S}_1).\cdots.\mathit{dual}(\mathcal{S}_k))) \cup \sigma_{k+1}(T') \\
  &= \mathit{ik}(\mathcal{I}(\mathit{dual}(\mathcal{S}_1).\cdots.\mathit{dual}(\mathcal{S}_k))) \cup \mathcal{I}(T') \\
  &= \mathit{ik}(\mathcal{I}(\mathit{dual}(\mathcal{S}_1).\cdots.\mathit{dual}(\mathcal{S}_k))) \cup \mathit{ik}(\mathcal{I}(\mathit{dual}(\mathcal{S}_{k+1}))) \\
  &= \mathit{ik}(\mathcal{I}(\mathit{dual}(\mathcal{S}_1).\cdots.\mathit{dual}(\mathcal{S}_k).\mathit{dual}(\mathcal{S}_{k+1})))
  \end{aligned}$$

  which proves the third part of the case.

- Note that $\mathit{dual}(S) = S$, since S does not contain message transmission steps, and therefore $\mathit{db}(\sigma_{k+1}(S)) = \mathit{db}(\sigma_{k+1}(\mathit{dual}(\mathcal{S}_{k+1})))$. Together with the variable disjointness and (H4) we then have:

  $$\begin{aligned}
  &\mathit{db}(\mathsf{insert}(D).\sigma_{k+1}(S)) \\
  &= \mathit{db}(\mathsf{insert}(D).\mathcal{I}(\mathit{dual}(\mathcal{S}_{k+1}))) \\
  &= \mathit{db}(\mathsf{insert}(\mathit{db}(\mathcal{I}(\mathit{dual}(\mathcal{S}_1).\cdots.\mathit{dual}(\mathcal{S}_k)))).\mathcal{I}(\mathit{dual}(\mathcal{S}_{k+1}))) \\
  &= \mathit{db}(\mathcal{I}(\mathit{dual}(\mathcal{S}_1).\cdots.\mathit{dual}(\mathcal{S}_k).\mathit{dual}(\mathcal{S}_{k+1})))
  \end{aligned}$$

  which proves the fourth part of the case.

Thus we have proven all parts of the thesis for the inductive case.

2. We also prove the second implication by an induction on reachability. Again, the base case is trivial.

   For the inductive case we assume that

$$\begin{aligned}(\mathcal{P}_0; 0) &\overset{w}{\Longrightarrow}\bullet^* \quad (\mathcal{P}; \mathcal{A})\\ &\overset{\mathcal{S}_{k+1}}{\Longrightarrow}\bullet \quad (\mathcal{P} \setminus \{\mathcal{S}_{k+1}\}; \mathcal{A}.dual(\mathcal{S}_{k+1}))\end{aligned}$$

   and $\mathcal{I} \models_s \mathcal{A}.dual(\mathcal{S}_{k+1})$ where $w = \mathcal{S}_1, \ldots, \mathcal{S}_k$, $dom(\mathcal{I}) = fv(\mathcal{A}.dual(\mathcal{S}_{k+1}))$, and $\mathcal{I}$ is ground. Now obtain the unique σ_{k+1} and $\mathcal{I}'$ such that $dom(\sigma_{k+1}) = fv(dual(\mathcal{S}_{k+1}))$, $dom(\mathcal{I}') = fv(\mathcal{A})$, and $\mathcal{I} = \mathcal{I}' \cdot \sigma_{k+1}$. This is always possible since the domain of $\mathcal{I}$ is exactly the free variables of $\mathcal{A}.dual(\mathcal{S}_{k+1})$ and $\mathcal{I}$ is ground and since all strands of $\mathcal{P}_0$ are pairwise-variable disjoint. Hence we have that

$$\mathcal{I}' \models_s \mathcal{A} \text{ and } [\![ik(\mathcal{I}'(\mathcal{A})), db(\mathcal{I}'(\mathcal{A})); dual(\mathcal{S}_{k+1})]\!]_s\ \sigma_{k+1},$$

   the former of which is the premise to the induction hypotheses for this case, and we can therefore apply the induction hypotheses to get

   (H1) $(\mathcal{P}_0; \emptyset, \emptyset) \overset{w'}{\Longrightarrow}{}^* (\mathcal{P}; ik(\mathcal{I}'(\mathcal{A})), db(\mathcal{I}'(\mathcal{A})))$ where $w' = (\sigma_1, \mathcal{S}_1), \ldots, (\sigma_k, \mathcal{S}_k)$,

   (H2) $\mathcal{A} = dual(\mathcal{S}_1).\cdots.dual(\mathcal{S}_k)$,

   (H3) $dom(\sigma_i) = fv(\mathcal{S}_i)$ for all $i \in \{1, \ldots, k\}$, and

   (H4) $\mathcal{I}' = \sigma_1 \cdot \ldots \cdot \sigma_k$

   This basically lets us immediately prove the second, third, and fourth part of the thesis for this case, namely

   - $\mathcal{A}.dual(\mathcal{S}_{k+1}) = dual(\mathcal{S}_1).\cdots.dual(\mathcal{S}_k).dual(\mathcal{S}_{k+1})$,
   - $dom(\sigma_i) = fv(\mathcal{S}_i)$ for all $i \in \{1, \ldots, k+1\}$, and
   - $\mathcal{I} = \sigma_1 \cdot \ldots \cdot \sigma_k \cdot \sigma_{k+1}$

   All that remains to be proven is

$$(\mathcal{P}_0; \emptyset, \emptyset) \overset{w''}{\Longrightarrow}{}^* (\mathcal{P} \setminus \{\mathcal{S}_{k+1}\}; ik(\mathcal{I}(\mathcal{A}.dual(\mathcal{S}_{k+1}))), db(\mathcal{I}(\mathcal{A}.dual(\mathcal{S}_{k+1}))))$$

   where $w'' = w', (\sigma_{k+1}, \mathcal{S}_{k+1})$.

   From the first induction hypothesis (H1), Lemma 4.4, and

$$[\![ik(\mathcal{I}'(\mathcal{A})), db(\mathcal{I}'(\mathcal{A})); dual(\mathcal{S}_{k+1})]\!]_s\ \sigma_{k+1}$$

we can apply $\overset{(\sigma_{k+1},\mathcal{S}_{k+1})}{\Longrightarrow}$ to get

$$(\mathcal{P}_0; \emptyset, \emptyset) \overset{w''}{\Longrightarrow}^* (\mathcal{P} \setminus \{\mathcal{S}_{k+1}\}; \mathit{ik}(\mathcal{I}'(\mathcal{A})) \cup \sigma_{k+1}(T'), \\ \mathit{db}(\mathsf{insert}(\mathit{db}(\mathcal{I}'(\mathcal{A}))).\sigma_{k+1}(S)))$$

where $\mathcal{S}_{k+1} = \mathsf{receive}(T).S.\mathsf{send}(T')$. Hence we only need to prove that

- $\mathit{ik}(\mathcal{I}'(\mathcal{A})) \cup \sigma_{k+1}(T') = \mathit{ik}(\mathcal{I}(\mathcal{A}.\mathit{dual}(\mathcal{S}_{k+1})))$ and
- $\mathit{db}(\mathsf{insert}(\mathit{db}(\mathcal{I}'(\mathcal{A}))).\sigma_{k+1}(S)) = \mathit{db}(\mathcal{I}(\mathcal{A}.\mathit{dual}(\mathcal{S}_{k+1})))$

and this is proven similarly to how we proved the third and fourth parts of the previous case. Thus we have proven all four conjuncts—(a), (b), (c), and (d)—of the conclusion for this inductive case.

$\square$

4.7.3 Constraint Reduction

We prove Theorem 4.11 in two steps. First we prove that the translation preserves well-formedness. Secondly we prove the semantic equivalence.

Theorem 4.11(2) (Well-formedness of translation). *Let X be a set of variables and D be a database mapping such that $\mathit{fv}(D) \subseteq X$. If $\mathcal{A}$ is well-formed w.r.t. the variables X and $\mathcal{A}' \in \mathit{tr}_D(\mathcal{A})$ then $\mathcal{A}'$ is well-formed w.r.t. X.*

Proof. We prove the statement by an induction on $\mathit{tr}_D(\mathcal{A})$:

- Case $\mathit{tr}_D(0)$: Trivially true by definition of well-formedness.
- Cases $\mathit{tr}_D(\mathsf{send}(t).\mathcal{A})$, $\mathit{tr}_D(\mathsf{receive}(t).\mathcal{A})$, $\mathit{tr}_D(t \doteq t'.\mathcal{A})$, and $\mathit{tr}_D((\forall \bar{x}.\ t \not\doteq t').\mathcal{A})$: Follows easily from the induction hypotheses.
- Case $\mathit{tr}_D(\mathsf{insert}(t,s).\mathcal{A})$: From the premises we have that $\mathit{fv}(t) \cup \mathit{fv}(s) \subseteq X$ because $\mathsf{insert}(t,s).\mathcal{A}$ is well-formed w.r.t. X. Hence $\mathcal{A}$ is well-formed w.r.t. X and $\mathit{fv}(D \cup \{(t,s)\}) = \mathit{fv}(D) \subseteq X$ as well. Thus we can apply the induction hypothesis to $\mathcal{A}' \in \mathit{tr}_{D\cup\{(t,s)\}}(\mathcal{A})$ and conclude the case.
- Case $\mathit{tr}_D(\mathsf{delete}(t,s).\mathcal{A})$: Note that $\mathit{fv}(D \setminus \{d_1,\dots,d_i\})) \subseteq \mathit{fv}(D) \subseteq X$. Hence equalities of the form $(t,s) \doteq d$ for $d \in D$ are always well-formed

w.r.t. X. Note also that tr_D translates $\mathsf{delete}(t, s)$ into equalities and inequalities of the form $(t, s) \doteq d$ and $(t, s) \not\doteq d$ for $d \in D$. Since inequalities do not have any well-formedness requirements we can now conclude the case using the induction hypothesis.

- Case $tr_D(t \mathrel{\dot\in} s.\mathcal{A})$: By the premises we have that $\mathcal{A}$ is well-formed w.r.t. $X \cup fv(t) \cup fv(s)$. From the induction hypothesis we then have that $\mathcal{A}' \in tr_D(\mathcal{A})$ is also well-formed w.r.t. $X \cup fv(t) \cup fv(s)$. Thus $(t, s) \doteq d.\mathcal{A}'$ is well-formed w.r.t. X.
- Case $tr_D((\forall \bar{y}.\ t \mathrel{\dot\notin} s).\mathcal{A})$: The constraints $\forall \bar{y}.\ t \mathrel{\dot\notin} s$ and $(\forall \bar{y}.\ (t, s) \not\doteq d_1). \cdots .(\forall \bar{y}.\ (t, s) \not\doteq d_n)$ have the same well-formedness requirement. Thus the case follows straightforwardly from the induction hypothesis.

□

Theorem 4.11(1) (Semantic equivalence of constraints and their translation). *Assume $fv(D)$ to be disjoint from the bound variables of $\mathcal{A}$. Then:*

$$[\![M, \mathcal{I}(D); \mathcal{A}]\!]_s\ \mathcal{I} \text{ iff there exists } \mathcal{A}' \in tr_D(\mathcal{A}) \text{ such that } [\![M; \mathcal{A}']\!]\ \mathcal{I}.$$

Proof. Consider the following two statements which together are equivalent to the original biconditional:

1. If $[\![M, \mathcal{I}(D); \mathcal{A}]\!]_s\ \mathcal{I}$ then $\exists \mathcal{A}' \in tr_D(\mathcal{A}).\ [\![M; \mathcal{A}']\!]\ \mathcal{I}$.
2. If $\mathcal{A}' \in tr_D(\mathcal{A})$ and $[\![M; \mathcal{A}']\!]\ \mathcal{I}$ then $[\![M, \mathcal{I}(D); \mathcal{A}]\!]_s\ \mathcal{I}$.

We prove the first of these implications by an induction on $\mathcal{A}$:

- Case 0: Trivially true.
- Cases $\mathsf{send}(t).\mathcal{A}$, $\mathsf{receive}(t).\mathcal{A}$, $t \doteq t'.\mathcal{A}$, and $(\forall \bar{x}.\ t \not\doteq t').\mathcal{A}$: Follows straightforwardly from the induction hypotheses.
- Case $\mathsf{insert}(t, s).\mathcal{A}$: So $[\![M, \mathcal{I}(D) \cup \{\mathcal{I}((t, s))\}; \mathcal{A}]\!]_s\ \mathcal{I}$. Since $\mathcal{I}(D) \cup \{\mathcal{I}((t, s))\} = \mathcal{I}(D \cup \{(t, s)\})$ we can apply the induction hypothesis to obtain $\mathcal{A}'$ where $\mathcal{A}' \in tr_{D \cup \{(t,s)\}}(\mathcal{A})$ and $[\![M; \mathcal{A}']\!]\ \mathcal{I}$. Thus we can conclude that $\mathcal{A}' \in tr_D(\mathsf{insert}(t, s).\mathcal{A})$ and $[\![M; \mathcal{A}']\!]\ \mathcal{I}$.

- Case $\mathsf{delete}(t,s).\mathcal{A}$: Hence $[\![M, \mathcal{I}(D) \setminus \{\mathcal{I}((t,s))\}; \mathcal{A}]\!]_s\ \mathcal{I}$. Now partition D into the sets $\{d_1,\dots,d_i\}$ and $\{d_{i+1},\dots,d_n\}$ for some $0 \le i \le n$ such that $\mathcal{I}(\{(t,s)\}) = \mathcal{I}(\{d_1,\dots,d_i\})$ and $\mathcal{I}((t,s)) \notin \mathcal{I}(\{d_{i+1},\dots,d_n\})$. Then

$$\begin{aligned}&\mathcal{I}(D) \setminus \mathcal{I}(\{(t,s)\})\\ &\quad = \mathcal{I}(D) \setminus \mathcal{I}(\{d_1,\dots,d_i\})\\ &\quad = \mathcal{I}(\{d_{i+1},\dots,d_n\})\\ &\quad = \mathcal{I}(D \setminus \{d_1,\dots,d_i\})\end{aligned}$$

  We can then apply the induction hypothesis to obtain an ordinary constraint $\mathcal{A}' \in tr_{D\setminus\{d_1,\dots,d_i\}}(\mathcal{A})$ such that $[\![M; \mathcal{A}']\!]\ \mathcal{I}$. Now let $\mathcal{B} = (t,s) \doteq d_1.\cdots.(t,s) \doteq d_i.(t,s) \not\doteq d_{i+1}.\cdots.(t,s) \not\doteq d_n$. Then we have that $\mathcal{B}.\mathcal{A}' \in tr_D(\mathcal{A})$ and $[\![M; \mathcal{B}.\mathcal{A}']\!]\ \mathcal{I}$ which concludes the case.

- Case $t \mathrel{\dot\in} s.\mathcal{A}$: Hence, by the premises of this case, $[\![M, \mathcal{I}(D); \mathcal{A}]\!]_s\ \mathcal{I}$ where $\mathcal{I}((t,s)) \in \mathcal{I}(D)$. So by the induction hypothesis we can obtain $\mathcal{A}'$ such that $\mathcal{A}' \in tr_D(\mathcal{A})$ and $[\![M; \mathcal{A}']\!]\ \mathcal{I}$. Because $\mathcal{I}((t,s)) \in \mathcal{I}(D)$ it must be the case that there exists some $d \in D$ such that $\mathcal{I}((t,s)) = \mathcal{I}(d)$. Thus we can conclude $(t,s) \doteq d.\mathcal{A}' \in tr_D(t \mathrel{\dot\in} s.\mathcal{A})$ for such a $d \in D$ and $[\![M; (t,s) \doteq d.\mathcal{A}']\!]\ \mathcal{I}$.

- Case $(\forall \bar{x}.\ t \mathrel{\dot\notin} s).\mathcal{A}$: Hence $\mathcal{I}(\delta((t,s))) \notin \mathcal{I}(D)$ for all ground substitutions δ with domain $\bar{x}$. Hence $\mathcal{I}(\delta((t,s))) \neq \mathcal{I}(d)$ for all $d \in D$ and all δ with domain $\bar{x}$. Since $\bar{x} \cap fv(D) = \emptyset$ we have that $\delta(D) = D$. Hence $\mathcal{I}(\delta((t,s))) \neq \mathcal{I}(\delta(d))$ for all $d \in D$ and all δ with domain $\bar{x}$. Therefore $[\![M; (\forall \bar{x}.\ (t,s) \not\doteq d_1).\cdots.(\forall \bar{x}.\ (t,s) \not\doteq d_n)]\!]\ \mathcal{I}$, where $D = \{d_1,\dots,d_n\}$, by definition of the constraint semantics. Thus the case follows by the induction hypothesis.

- The cases $\mathsf{assert}(t).\mathcal{A}$, $\mathsf{event}(t).\mathcal{A}$, and $(\forall \bar{x}.\ \neg\mathsf{event}(t)).\mathcal{A}$ are proven similarly to the cases $\mathsf{insert}(t,s).\mathcal{A}$, $t \mathrel{\dot\in} s.\mathcal{A}$, and $(\forall \bar{x}.\ t \mathrel{\dot\notin} s).\mathcal{A}$ respectively.

The other implication is also proven by an induction on $\mathcal{A}$:

- Case 0: Trivially true.

- Cases $\mathsf{send}(t).\mathcal{A}$, $\mathsf{receive}(t).\mathcal{A}$, $t \doteq t'.\mathcal{A}$, and $(\forall \bar{x}.\ t \not\doteq t').\mathcal{A}$: Follows straightforwardly from the induction hypotheses.

- Case $\mathsf{insert}(t,s).\mathcal{A}$: Hence $\mathcal{A}' \in tr_{D\cup\{(t,s)\}}(\mathcal{A})$. Since we already have that $[\![M, \emptyset; \mathcal{A}']\!]_s\ \mathcal{I}$ from the premises we can now apply the induction hypothesis to get that $[\![M, \mathcal{I}(D \cup \{(t,s)\}); \mathcal{A}]\!]_s\ \mathcal{I}$. Since we also have that $\mathcal{I}(D \cup \{(t,s)\}) = \mathcal{I}(D) \cup \{\mathcal{I}((t,s))\}$ we can conclude the following:

$$[\![M, \mathcal{I}(D); \mathsf{insert}(t,s).\mathcal{A}]\!]_s\ \mathcal{I}$$

- Case $\mathsf{delete}(t,s).\mathcal{A}$: Hence $\mathcal{A}' = \mathcal{B}.\mathcal{A}''$ for some $d_1,\dots,d_n$, i, n, $\mathcal{B}$, and $\mathcal{A}''$ where
  - $0 \le i \le n$,
  - $D = \{d_1,\dots,d_i \dots, d_n\}$,
  - $\mathcal{B} = (t,s) \doteq d_1.\cdots.(t,s) \doteq d_i.(t,s) \not\doteq d_{i+1}.\cdots.(t,s) \not\doteq d_n$, and
  - $\mathcal{A}'' \in tr_{D\setminus\{d_1,\dots,d_i\}}(\mathcal{A})$.

  We also have that $[\![M;\mathcal{A}'']\!]\ \mathcal{I}$ and $[\![M;\mathcal{B}]\!]\ \mathcal{I}$ because $[\![M;\mathcal{A}']\!]\ \mathcal{I}$. Note that $\mathcal{I}(D\setminus\{d_1,\dots,d_i\}) = \mathcal{I}(D)\setminus\{\mathcal{I}((t,s))\}$ because $\mathcal{I}((s,t)) \notin \mathcal{I}(\{d_{i+1},\dots,d_n\})$ and $\{\mathcal{I}((t,s))\} = \mathcal{I}(\{d_1,\dots,d_i\})$. Thus, we can apply the induction hypothesis and conclude the following:

$$[\![M,\mathcal{I}(D);\mathsf{delete}(t,s).\mathcal{A}]\!]_s\ \mathcal{I}$$

- Case $t \mathrel{\dot\in} s.\mathcal{A}$: Hence $\mathcal{A}' = (t,s) \doteq d.\mathcal{A}''$ for some $d \in D$ and $\mathcal{A}'' \in tr_D(\mathcal{A})$, and together with the premises we then have that $\mathcal{I}((t,s)) = \mathcal{I}(d)$ and $[\![M,\mathcal{I}(D);\mathcal{A}'']\!]_s\ \mathcal{I}$. We can now apply the induction hypothesis to get $[\![M,\mathcal{I}(D);\mathcal{A}]\!]_s\ \mathcal{I}$. Since $d \in D$ and $\mathcal{I}((t,s)) = \mathcal{I}(d)$ we also have that $\mathcal{I}((t,s))$ is in $\mathcal{I}(D)$. Thus we can conclude the following

$$[\![M,\mathcal{I}(D);t \mathrel{\dot\in} s.\mathcal{A}]\!]_s\ \mathcal{I}$$

- Case $(\forall\bar{x}.\ t \mathrel{\dot\notin} s).\mathcal{A}$: Hence $\mathcal{A}' = (\forall\bar{x}.\ (t,s) \not\doteq d_1).\cdots.(\forall\bar{x}.\ (t,s) \not\doteq d_n).\mathcal{A}''$ for some $\mathcal{A}'' \in tr_D(\mathcal{A})$ and $D = \{d_1,\dots,d_n\}$. Hence, using the premises, we know that $\mathcal{I}(\delta((t,s))) \neq \mathcal{I}(\delta(d_i))$ for all $i \in \{1,\dots,n\}$ and all ground δ with domain $\bar{x}$. Hence $\mathcal{I}(\delta((t,s))) \notin \mathcal{I}(D)$ for all ground δ with domain $\bar{x}$ because $\bar{x}$ and the free variables of D are disjoint. Since we also have $[\![M;\mathcal{A}'']\!]\ \mathcal{I}$ from the premises we can apply the induction hypothesis to get $[\![M,\mathcal{I}(D);\mathcal{A}]\!]_s\ \mathcal{I}$, and thus we can conclude the following:

$$[\![M,\mathcal{I}(D);(\forall\bar{x}.\ t \mathrel{\dot\notin} s).\mathcal{A}]\!]_s\ \mathcal{I}$$

□

4.7.4 The Typing Result

The remaining section contains the proof of our typing results and related lemmas.

Lemma 4.13 (Type-flaw resistance preservation).

1. *If $\mathcal{A}$ is type-flaw resistant, well-formed, and $\mathcal{A}' \in tr(\mathcal{A})$, then $\mathcal{A}'$ is type-flaw resistant.*
2. *If $\mathcal{P}_0$ is a type-flaw resistant protocol and $(\mathcal{P}_0; 0) \overset{w}{\Longrightarrow}^{\bullet *} (\mathcal{P}; \mathcal{A})$ then both $\mathcal{P}$ and $\mathcal{A}$ are type-flaw resistant.*

PROOF.

1. We first prove that $trms(\mathcal{A}') \cup setops(\mathcal{A}')$ is type-flaw resistant. Note that $trms(\mathcal{A}') \setminus trms(\mathcal{A}) \subseteq setops(\mathcal{A})$, and that $setops(\mathcal{A}') = \emptyset$. Hence $trms(\mathcal{A}') \cup setops(\mathcal{A}') \subseteq trms(\mathcal{A}) \cup setops(\mathcal{A})$. Since $trms(\mathcal{A}) \cup setops(\mathcal{A})$ is by assumption type-flaw resistant it follows that any subset is also type-flaw resistant. Thus $trms(\mathcal{A}') \cup setops(\mathcal{A}')$ is type-flaw resistant.

   The reduced constraint $\mathcal{A}'$ does not contain set operations, and the remaining constraint steps of $\mathcal{A}'$ either originate from $\mathcal{A}$ or are constructed during the translation $tr(\mathcal{A})$. Thus it is sufficient in the remaining proof to only consider those steps which are created during the translation $tr(\mathcal{A})$, and we do so by a case analysis:

   - During translation of a set operation of the form $t \mathrel{\dot{\in}} s$ the translation adds one step of the form $(t, s) \doteq (t', s')$ where $\mathsf{insert}(t', s')$ occurred somewhere in $\mathcal{A}$. Since both (t, s) and (t', s') occur in $setops(\mathcal{A})$, which is a subset of $SMP(\mathcal{A}) \setminus \mathcal{V}$, they must have the same type if they are unifiable. Thus type-flaw resistance is satisfied in this case.
   - During translation of a set operation of the form $\mathsf{delete}(t, s)$ or $\forall \bar{x}.\ t \mathrel{\dot{\notin}} s$ the translation adds new steps of the form $(t, s) \doteq (t', s')$ and $\forall \bar{x}.\ (t, s) \not\doteq (t', s')$ where $\mathsf{insert}(t', s')$ occurred somewhere in $\mathcal{A}$. As we argued earlier, if (t, s) and (t', s') are unifiable then they have the same type. Hence all the new equality steps $(t, s) \doteq (t', s')$ are type-flaw resistant. Also, from type-flaw resistance and well-formedness of $\mathcal{A}$ we have that

     (a) the variables of (t', s') are disjoint from the bound variables $\bar{x}$ (because of well-formedness), and

     (b) there is no composed term of the form $f(x_1, \dots, x_n)$, where $n > 0$ and $x_1, \dots, x_n \in \bar{x}$, occurring in (t, s).

     Hence the resulting inequality steps $\forall \bar{x}.\ (t, s) \not\doteq (t', s')$ are type-flaw resistant.

   Thus $\mathcal{A}'$ is type-flaw resistant.

2. This follows from the fact that $\mathcal{A} = dual(\mathcal{S}_1). \cdots .dual(\mathcal{S}_k)$ for some $\mathcal{S}_1, \dots, \mathcal{S}_k \in \mathcal{P}_0$, and since each $\mathcal{S}_i$ are type-flaw resistant (so each $dual(\mathcal{S}_i)$ is type-flaw resistant), $trms(\mathcal{P}_0) \cup setops(\mathcal{P}_0)$ is type-flaw resistant and $trms(\mathcal{A}) \cup$

$setops(\mathcal{A}) \subseteq trms(\mathcal{P}_0) \cup setops(\mathcal{P}_0)$ (so $trms(\mathcal{A}) \cup setops(\mathcal{A})$ is type-flaw resistant), we can conclude that $\mathcal{A}$ is type-flaw resistant. □

Theorem 4.14 (Typing result on symbolic constraints). *If $\mathcal{A}$ is well-formed, $\mathcal{I} \models_s \mathcal{A}$, and $\mathcal{A}$ is type-flaw resistant, then there exists a well-typed interpretation $\mathcal{I}_\tau$ such that $\mathcal{I}_\tau \models_s \mathcal{A}$.*

Proof. From Theorem 4.11, Lemma 4.13(1), and the assumptions we can obtain a type-flaw resistant ordinary constraint $\mathcal{A}'$ such that $\mathcal{A}' \in tr(\mathcal{A})$ and $\mathcal{I} \models \mathcal{A}'$. Hence, we can obtain a well-typed interpretation $\mathcal{I}_\tau$ such that $\mathcal{I}_\tau \models \mathcal{A}'$ by Corollary 3.28, and by then applying Theorem 4.11 again we can conclude the proof. □

Theorem 4.15 (Typing result for stateful protocols). *If $\mathcal{P}_0$ is a type-flaw resistant protocol, and*

$$(\mathcal{P}_0; \emptyset, \emptyset) \overset{w}{\Longrightarrow}{}^* (\mathcal{P}; M, D) \text{ where } w = (\sigma_1, \mathcal{S}_1), \dots, (\sigma_k, \mathcal{S}_k)$$

then there exists a state $(\mathcal{P}; M', D')$ such that

$$(\mathcal{P}_0; \emptyset, \emptyset) \overset{w'}{\Longrightarrow}{}^* (\mathcal{P}; M', D') \text{ where } w' = (\sigma'_1, \mathcal{S}_1), \dots, (\sigma'_k, \mathcal{S}_k)$$

for some well-typed ground substitutions $\sigma'_1, \dots, \sigma'_k$.

Proof. By first applying Theorem 4.9(1) and Lemma 4.13(2), then Theorem 4.14, and finally Theorem 4.9(2) we obtain the desired result. (Note that we in this proof need to use the general version of Theorem 4.9 given on page 96.) □

4.8 Isabelle/HOL Formalization

We give in this section an overview of the Isabelle-formalized stateful typing result and point out where it differs from the theory presented in this chapter. The formalization closely follows the pen-and-paper proofs, and so we will not go into detail with the actual Isabelle-formalized proofs but rather focus on how the main definitions and theorems are modeled in Isabelle.

4.8.1 Locale Assumptions

As in Chapter 3 the typed model is parameterized over a typing function Γ that must satisfy certain requirements. In Isabelle we again model this with a locale. For the stateful typing result we need a distinguished binary symbol *pair* that is used in the constraint translation, and so we extend the typed model locale of Chapter 3 accordingly:

$$\begin{array}{l}
\textbf{locale}\ \textit{stateful-typed-model} = \textit{typed-model arity public}\ \mathsf{Ana}\ \Gamma \\
\quad \textbf{for}\ \textit{arity} :: \Sigma \Rightarrow \textit{nat} \\
\quad\ \textbf{and}\ \textit{public} :: \Sigma \Rightarrow \textit{bool} \\
\quad\ \textbf{and}\ \mathsf{Ana} :: (\Sigma, \mathcal{V})\ \textit{term} \Rightarrow ((\Sigma, \mathcal{V})\ \textit{term set} \times (\Sigma, \mathcal{V})\ \textit{term set}) \\
\quad \textbf{and}\ \Gamma :: (\Sigma, \mathcal{V})\ \textit{term set} \Rightarrow (\Sigma, \mathfrak{T}_a)\ \textit{term-type} \\
+ \\
\quad \textbf{fixes}\ \textit{pair} :: \Sigma \\
\quad \textbf{assumes}\ \textit{arity pair} = 2 \\
\quad\ \textbf{and}\ \bigwedge f\ T\ \delta\ K\ M.\ \mathsf{Ana}\ (\textit{Fun}\ f\ T) = (K, M) \Longrightarrow \\
\qquad\qquad\qquad\qquad\quad \mathsf{Ana}\ (\delta(\textit{Fun}\ f\ T)) = (\delta(K), \delta(M))
\end{array}$$

The second assumption here formalizes the rule Ana_4' which was explained in Section 4.1.

4.8.2 Strand Definitions

Stateful strands and constraints are defined as lists of steps similarly to the "stateless" strands of Chapter 3 but here with new constructors for set updates (*Insert* and *Delete*) and membership checks (*InSet* and *NotInSet*):

$$\begin{array}{l}
\textbf{datatype}\ (\Sigma, \mathcal{V})\ \textit{stateful-strand-step} = \\
\quad\ \ \textit{Send}\ ((\Sigma, \mathcal{V})\ \textit{term}) \\
\mid\ \ \textit{Receive}\ ((\Sigma, \mathcal{V})\ \textit{term}) \\
\mid\ \ \textit{Equality}\ ((\Sigma, \mathcal{V})\ \textit{term})\ ((\Sigma, \mathcal{V})\ \textit{term}) \\
\mid\ \ \textit{Inequality}\ (\mathcal{V}\ \textit{list})\ ((\Sigma, \mathcal{V})\ \textit{term})\ ((\Sigma, \mathcal{V})\ \textit{term}) \\
\mid\ \ \textit{Insert}\ ((\Sigma, \mathcal{V})\ \textit{term})\ ((\Sigma, \mathcal{V})\ \textit{term}) \\
\mid\ \ \textit{Delete}\ ((\Sigma, \mathcal{V})\ \textit{term})\ ((\Sigma, \mathcal{V})\ \textit{term}) \\
\mid\ \ \textit{InSet}\ ((\Sigma, \mathcal{V})\ \textit{term})\ ((\Sigma, \mathcal{V})\ \textit{term}) \\
\mid\ \ \textit{NotInSet}\ (\mathcal{V}\ \textit{list})\ ((\Sigma, \mathcal{V})\ \textit{term})\ ((\Sigma, \mathcal{V})\ \textit{term})
\end{array}$$

$$\textbf{type-synonym}\ (\Sigma, \mathcal{V})\ \textit{stateful-strand} = (\Sigma, \mathcal{V})\ \textit{stateful-strand-step list}$$

For convenience we just use the notation defined in Section 4.3 for strands of type *stateful-strand*. Note that this will introduce some ambiguity in the following

$$
\begin{array}{lll}
\multicolumn{3}{l}{\textbf{fun } [\![\_;\_;\_]\!]_s \textbf{ where}} \\
\quad [\![M;D;[]]\!]_s & = & \lambda\mathcal{I}.\ \mathit{True} \\
\mid\ [\![M;D;\mathsf{send}(t).A]\!]_s & = & \lambda\mathcal{I}.\ M \vdash \mathcal{I}(t) \wedge [\![M;D;A]\!]_s\ \mathcal{I} \\
\mid\ [\![M;D;\mathsf{receive}(t).A]\!]_s & = & \lambda\mathcal{I}.\ [\![\mathit{insert}\ \mathcal{I}(t)\ M;D;A]\!]_s\ \mathcal{I} \\
\mid\ [\![M;D;t \doteq t'.A]\!]_s & = & \lambda\mathcal{I}.\ \mathcal{I}(t) = \mathcal{I}(t') \wedge [\![M;D;A]\!]_s\ \mathcal{I} \\
\mid\ [\![M;D;(\forall X.\ t \not\doteq t').A]\!]_s & = & \lambda\mathcal{I}. \\
\multicolumn{3}{l}{\qquad (\forall\delta.\ \mathit{subst\text{-}domain}\ \delta = \mathit{set}\ X \wedge \mathit{ground}\ (\mathit{subst\text{-}range}\ \delta)} \\
\multicolumn{3}{l}{\qquad\qquad \longrightarrow \mathcal{I}(\delta(t)) \neq \mathcal{I}(\delta(t')))} \\
\multicolumn{3}{l}{\qquad \wedge [\![M;D;A]\!]_s\ \mathcal{I}} \\
\mid\ [\![M;D;\mathsf{insert}(t,s).A]\!]_s & = & \lambda\mathcal{I}.\ [\![M;\mathit{insert}\ \mathcal{I}((t,s))\ D;A]\!]_s\ \mathcal{I} \\
\mid\ [\![M;D;\mathsf{delete}(t,s).A]\!]_s & = & \lambda\mathcal{I}.\ [\![M;D \setminus \{\mathcal{I}((t,s))\};A]\!]_s\ \mathcal{I} \\
\mid\ [\![M;D;t \mathrel{\dot\in} s.A]\!]_s & = & \lambda\mathcal{I}.\ \mathcal{I}((t,s)) \in D \wedge [\![M;D;A]\!]_s\ \mathcal{I} \\
\mid\ [\![M;D;(\forall X.\ t \mathrel{\dot\notin} s).A]\!]_s & = & \lambda\mathcal{I}. \\
\multicolumn{3}{l}{\qquad (\forall\delta.\ \mathit{subst\text{-}domain}\ \delta = \mathit{set}\ X \wedge \mathit{ground}\ (\mathit{subst\text{-}range}\ \delta)} \\
\multicolumn{3}{l}{\qquad\qquad \longrightarrow \mathcal{I}(\delta((t,s))) \notin D)} \\
\multicolumn{3}{l}{\qquad \wedge [\![M;D;A]\!]_s\ \mathcal{I}}
\end{array}
$$

$$\textbf{abbreviation } \mathcal{I} \models_s A \equiv [\![\emptyset;\emptyset;A]\!]_s\ \mathcal{I}$$

Figure 4.3: The constraint semantics formalized in Isabelle/HOL.

definitions, because the constructors of the stateless and the stateful strand datatypes overlap, but it should be obvious from the context which datatype is meant. This is of course only an ambiguity in this presentation, not the actual Isabelle-formalization.

Constraint well-formedness and semantics are defined similarly to Definition 4.6 respectively Definition 4.2, but we use a slightly different notation in the Isabelle-formalization. For well-formedness of constraints $\mathcal{A}$ we split the definition into two (see Figure 4.4): The function $\mathit{wf}'_{sst}\ V\ \mathcal{A}$ corresponds to $\mathit{wf}_V(\mathcal{A})$, and $\mathit{wf}_{sst}\ \mathcal{A}$ that corresponds to the full well-formedness criteria, i.e., that $\mathit{wf}_\emptyset(\mathcal{A})$ and that the free variables $\mathit{fv}(\mathcal{A})$ and bound variables $\mathit{bvars}(\mathcal{A})$ are disjoint. The Isabelle-formalized constraint semantics is listed in Figure 4.3 and closely follows Definition 4.2.

4.8.3 Constraint Reduction

The translation $\mathit{tr}_D(A)$ is defined as a function $\mathit{tr}\ A\ D$ in Isabelle where A is a constraint and D is a database mapping. The definition can be found in Figure 4.5.

$$
\begin{array}{lll}
\textbf{fun } \mathit{wf}'_{sst} \textbf{ where} & & \\
\quad \mathit{wf}'_{sst}\ V\ [] & = & \mathit{True} \\
\mid\ \mathit{wf}'_{sst}\ V\ (\mathsf{receive}(t).A) & = & (\mathit{fv}(t) \subseteq V \land \mathit{wf}'_{sst}\ VA) \\
\mid\ \mathit{wf}'_{sst}\ V\ (\mathsf{send}(t).A) & = & \mathit{wf}'_{sst}\ (V \cup \mathit{fv}(t))\ A \\
\mid\ \mathit{wf}'_{sst}\ V\ (t \doteq t'.A) & = & (\mathit{fv}(t') \subseteq V \land \mathit{wf}'_{sst}\ (V \cup \mathit{fv}(t))\ A) \\
\mid\ \mathit{wf}'_{sst}\ V\ ((\forall X.\ t \not\doteq t').A) & = & \mathit{wf}'_{sst}\ VA \\
\mid\ \mathit{wf}'_{sst}\ V\ (\mathsf{insert}(t,s).A) & = & (\mathit{fv}(t) \subseteq V \land \mathit{fv}(s) \subseteq V \land \mathit{wf}'_{sst}\ V\ A) \\
\mid\ \mathit{wf}'_{sst}\ V\ (\mathsf{delete}(t,s).A) & = & \mathit{wf}'_{sst}\ V\ A \\
\mid\ \mathit{wf}'_{sst}\ V\ (t \dot\in s.A) & = & \mathit{wf}'_{sst}\ (V \cup \mathit{fv}(t) \cup \mathit{fv}(s))\ A \\
\mid\ \mathit{wf}'_{sst}\ V\ ((\forall X.\ t \dot\notin s).A) & = & \mathit{wf}'_{sst}\ V\ A
\end{array}
$$

$$\textbf{abbreviation } \mathit{wf}_{sst}\ A \equiv \mathit{wf}'_{sst}\ \emptyset\ A \land \mathit{fv}(A) \cap \mathit{bvars}(A) = \emptyset$$

Figure 4.4: The constraint well-formedness requirements formalized in Isabelle/HOL.

```
fun $tr$ where
  $tr\ []\ D = [[]]$
| $tr\ (\mathsf{send}(t).A)\ D = map\ (\lambda B.\ \mathsf{send}(t).B)\ (tr\ A\ D)$
| $tr\ (\mathsf{receive}(t).A)\ D = map\ (\lambda B.\ \mathsf{receive}(t).B)\ (tr\ A\ D)$
| $tr\ (t \doteq t'.A)\ D = map\ (\lambda B.\ t \doteq t'.B)\ (tr\ A\ D)$
| $tr\ ((\forall X.\ t \not\doteq t').A)\ D = map\ (\lambda B.\ (\forall X.\ t \not\doteq t').B)\ (tr\ A\ D)$
| $tr\ (\mathsf{insert}(t,s).A)\ D = tr\ A\ (List.insert\ (t,s)\ D)$
| $tr\ (\mathsf{delete}(t,s).A)\ D = ($
     $concat\ (map\ (\lambda D_i.\ map\ (\lambda B.\ (map\ (\lambda d.\ to\text{-}pair\ (t,s) \doteq to\text{-}pair\ d)\ D_i).$
                                           $(map\ (\lambda d.\ to\text{-}pair\ (t,s) \not\doteq to\text{-}pair\ d)$
                                                $(filter\ (\lambda d.\ d \notin set\ D_i)\ D)).B)$
                                  $(tr\ A\ (filter\ (\lambda d.\ d \notin set\ D_i)\ D)))$
                      $(subseqs\ D)))$
| $tr\ (t \dot\in s.A)\ D =$
    $concat\ (map\ (\lambda B.\ map\ (\lambda d.\ to\text{-}pair\ (t,s) \doteq to\text{-}pair\ d.B)\ D)\ (tr\ A\ D))$
| $tr\ ((\forall X.\ t \dot\notin s).A)\ D =$
    $map\ (\lambda B.\ (map\ (\lambda d.\ (\forall X.\ to\text{-}pair\ (t,s) \not\doteq to\text{-}pair\ d))\ D).B)\ (tr\ A\ D)$
```

Figure 4.5: The constraint translation formalized in Isabelle/HOL.

There are some differences between the pen-and-paper version *tr* and the Isabelle version *tr*: We model the database mapping D and the result of the translation as lists instead of sets. This is for technical reasons more convenient: The *list* datatype in Isabelle/HOL is inductively defined and so all lists are finite. This ensures that D is always finite (*tr* is not defined for infinite D). The list representation moreover provides an ordering of the elements (namely their positions in D) and so we can easily iterate over the elements using the usual *map* and *filter* functions. For instance, in the translation of $\forall \bar{x}.\ t \mathrel{\dot{\notin}} s$ we need to construct a finite list of inequalities containing one inequality constraint for each element of D during translation, and we can do so with *map*.

As another consequence of the list representation we use subsequences instead of subsets. These subsequences are constructed using a function *subseqs*. We also use a function *List.insert* to insert new elements into lists. More precisely, *List.insert* d D appends d to the list D if d does not occur in D and *subseqs* D constructs all subsequences of D. We also use the abbreviation *to-pair* (t, t') to denote the term *Fun pair* $[t, t']$. The remaining functions *filter*, *map*, and *concat* are defined as usual.

The first equation of *tr* as defined in Figure 4.5 simply constructs the singleton list containing the empty constraint []: [[]]. This corresponds to the first equation of Definition 4.10 that returns a singleton set containing the empty constraint, i.e., $\{0\}$ in the pen-and-paper notation.

4.8.4 Type-Flaw Resistance

The Isabelle function $setops_{sst}$ corresponds to $setops(\cdot)$ while $trms_{sst}$ corresponds to $trms(\cdot)$. Substitution well-typedness is denoted by wt_{subst} in Isabelle. We define type-flaw resistance of stateful constraints in three steps. First, we formalize in Isabelle Definition 4.12(1) as follows:

definition $tfr_{set}\ M \equiv$
$$(\forall s \in (SMP\ M) \setminus \mathcal{V}.\ \forall t \in (SMP\ M) \setminus \mathcal{V}.\ (\exists \delta.\ \delta(s) = \delta(t)) \longrightarrow \Gamma(s) = \Gamma(t))$$

where $SMP\ M$ denotes the sub-message patterns of the set of terms M.

Secondly, to formalize Definition 4.12(2) we first define a predicate on the datatype *stateful-strand-step*:

$$
\begin{array}{lll}
\textbf{fun } \mathit{tfr}_{sstp} \textbf{ where} & & \\
\quad \mathit{tfr}_{sstp}\ (t \doteq t') & = & ((\exists \delta.\ \delta(t) = \delta(t')) \longrightarrow \Gamma(t) = \Gamma(t')) \\
| \quad \mathit{tfr}_{sstp}\ (\forall X.\ t \neq t') & = & (\\
& & \quad (\forall x \in (\mathit{fv}(t) \cup \mathit{fv}(t')) \setminus X.\ \exists a.\ \Gamma(\mathit{Var}\ x) = \mathit{TAtom}\ a) \lor \\
& & \quad (\forall f\ T.\ \mathit{Fun}\ f\ T \in \mathit{subterms}\ t \cup \mathit{subterms}\ t' \\
& & \qquad\qquad \longrightarrow T = [] \lor (\exists s \in T.\ s \notin \mathit{Var}\ `\ X) \\
\quad) & & \\
| \quad \mathit{tfr}_{sstp}\ (\forall X.\ t \dot{\notin} t') & = & (\\
& & \quad (\forall f\ T.\ \mathit{Fun}\ f\ T \in \mathit{subterms}\ (\mathit{to\text{-}pair}\ (t, t')) \\
& & \qquad\qquad \longrightarrow T = [] \lor (\exists s \in T.\ s \notin \mathit{Var}\ `\ X) \\
\quad) & & \\
| \quad \mathit{tfr}_{sstp}\ \_ & = & \mathit{True}
\end{array}
$$

Finally, we say that a stateful constraint $\mathcal{A}$ is type-flaw resistant if the set of terms $\mathit{trms}_{sst}\ \mathcal{A} \cup \mathit{setops}_{sst}\ \mathcal{A}$ satisfy tfr_{set} and if all steps of $\mathcal{A}$ satisfy tfr_{sstp}:

$$\textbf{definition}\ \mathit{tfr}_{sst}\ \mathcal{A} \equiv \mathit{tfr}_{set}\ (\mathit{trms}_{sst}\ \mathcal{A} \cup \mathit{setops}_{sst}\ \mathcal{A}) \land \mathit{list\text{-}all}\ \mathit{tfr}_{sstp}\ \mathcal{A}$$

where here *list-all* P L is true if and only if all elements of the list L satisfy the predicate P.

4.8.5 Lemmas and Theorems

The Isabelle-formalized proofs closely follows the pen-and-paper proofs and so we will not explain the proofs in this subsection.

Lemma 4.3 is formalized in Isabelle as follows. Here $\mathit{ik}_{sst}\ A$ denotes the intruder knowledge of A while $\mathit{dbupd}_{sst}\ A\ \mathcal{I}\ D$ corresponds to $\mathit{db}(\mathsf{insert}(D).\mathcal{I}(A))$:

lemma *strand-sem-append-stateful*:
$$[\![M; D; A.B]\!]_s\ \mathcal{I} \longleftrightarrow [\![M; D; A]\!]_s\ \mathcal{I} \land [\![M \cup \mathcal{I}(\mathit{ik}_{sst}\ A); \mathit{dbupd}_{sst}\ A\ \mathcal{I}\ D; B]\!]_s\ \mathcal{I}$$

The semantic equivalence theorem—Theorem 4.11—is as follows. The semantics for the ordinary constraints is here denoted by $\models$ while the semantics for the stateful constraints is denoted by $\models_s$:

lemma *tr-wf*:
assumes $A' \in \mathit{tr}\ A\ []$ **and** $\mathit{wf}_{sst}\ A$ **and** $\mathit{wf}_{trms}\ (\mathit{trms}_{sst}\ A)$
shows $\mathit{wf}_{st}\ \emptyset\ A'$ **and** $\mathit{wf}_{trms}\ (\mathit{trms}_{st}\ A')$ **and** $\mathit{fv}(A') \cap \mathit{bvars}(A') = \emptyset$

lemma *tr-sem-equiv*:
assumes $\mathit{fv}(A) \cap \mathit{bvars}(A) = \emptyset$ **and** $\mathit{interpretation}_{subst}\ \mathcal{I}$
shows $\mathcal{I} \models_s A \longleftrightarrow (\exists A' \in \mathit{tr}\ A\ [].\ \mathcal{I} \models A')$

Lemma 4.13 concerns type-flaw resistance. On the constraint-level the statement in Isabelle/HOL is as follows:

lemma *tr-tfr*:
assumes $A' \in tr\ A\ []$ **and** $tfr_{sst}\ A$ **and** $fv(A) \cap bvars(A) = \emptyset$
shows $tfr_{st}\ A'$

Finally we have the typing result on the constraint level, namely Theorem 4.14:

theorem *stateful-typing-result*:
assumes $wf_{sst}\ \mathcal{A}$ **and** $tfr_{sst}\ \mathcal{A}$ **and** $wf_{trms}\ (trms_{sst}\ \mathcal{A})$
and $interpretation_{subst}\ \mathcal{I}$ **and** $\mathcal{I} \models_s \mathcal{A}$
obtains $\mathcal{I}_\tau$ **where** $interpretation_{subst}\ \mathcal{I}_\tau$
and $\mathcal{I}_\tau \models_s \mathcal{A}$ **and** $wt_{subst}\ \mathcal{I}_\tau$ **and** $wf_{trms}\ (subst\text{-}range\ \mathcal{I}_\tau)$

4.9 Summary and Related Work

A relevant trend in protocol security is the support for stateful protocols, i.e., protocols in which participants can manipulate a global state that is shared among an unbounded number of sessions. This is for instance relevant to model security devices like key tokens or servers that maintain a database. There is only one typing result so far that supports stateful protocols, namely [Möd12]. We point out several mistakes of this paper in Section 4.4.3, showing that their results do not hold in this generality. A particular problem are variables of composed types in negative conditions, which illustrates that typing results for stateful systems are far more subtle than intuition suggests and rigorous proofs are necessary. Our main contribution of this chapter is to establish the first precise typing result for a class of stateful protocols—fixing also [Möd12]. Despite a meticulous formalization it is conceptually still quite simple, as it is based on a reduction to the existing typing results, in particular the formalization of Chapter 3.

Our typing result for stateful protocols conservatively extends existing ones, i.e., for stateless protocols we do not require any further restrictions. The restrictions on set operations are similar to those on messages and message checks. In fact, the condition of our typing result is satisfied by most examples distributed with the AIF-ω tool [MB16], and in the remaining cases a simple disambiguation of messages is sufficient.

Another closely related area are compositionality results that can often benefit from typing results. For instance, typing results can be used as a stepping stone for compositional reasoning, e.g., [AMMV15] prove that two protocols that are secure in isolation can also run securely on the same communication medium in parallel, if their messages do not interfere with each other, a requirement closely related to the typing result. Establishing compositionality for stateful protocols is the main objective of the next chapter.

Besides the trend towards the verification of more complex stateful protocols that this typing result focuses on, there are other crucial trends like the verification of privacy-type goals using equivalence properties, and typing results in this direction have been established [CCD14]. A question for future research is thus whether a typing result can be established for equivalence properties also in the presence of stateful protocols.

Chapter 5

Stateful Protocol Composition

In this chapter we prove a parallel compositionality result for stateful protocols, extending the Isabelle-formalization of the previous chapters. For protocols satisfying certain compositionality conditions our result shows that verifying the component protocols in isolation is sufficient to prove security of their composition. Our main contribution is an extension of the compositionality paradigm to stateful protocols where participants maintain shared databases. Because of the generality of our result we also cover many forms of sequential composition as a special case of stateful parallel composition. Moreover, we support declassification of shared secrets. Finally, we prove our result on the constraint-level in Isabelle/HOL providing a strong correctness guarantee of our proofs.

The proof of the main result is by a reduction to a problem of finding solutions for intruder constraints: given a satisfiable constraint representing an attack on the composition, we show that the projections of the constraints to the individual protocols are satisfiable. This particular tricky part of the proof has been formalized in the interactive theorem prover Isabelle/HOL. Last but not least, the formulation of the problem over intruder constraints allows us to apply our result with a variety of protocol formalisms such as applied-π calculus and multi-set rewriting.

The chapter is organized as follows:

- In Section 5.1 we define the stateful protocols we consider in this chapter.
- Afterwards we define protocol composition and introduce a keyserver protocol example in Section 5.2.
- We define our compositionality conditions and prove our main result in Section 5.3. In Subsection 5.3.6 we explain how sequential composition is a special case of stateful parallel composition.
- Finally, we give the proofs of our results in Section 5.4.

5.1 Stateful Protocols

We now introduce another strand-based protocol formalism for stateful protocols adapted from Chapter 4. This formalism is compact and reduced to the key concepts needed here, while more complex formalisms like process calculi can easily be fitted similarly. The semantics is defined by a symbolic transition system where constraints are built up during transitions. The models of the constraints then constitute the concrete protocol runs. We later use the typing result proven in Chapter 4 that shows that for a large class of protocols, it is without loss of attacks to restrict the constraints to well-typed models.

Example 5.1 *In the present chapter we use the following* Ana *theory to model asymmetric encryption and signatures:*

$$\begin{array}{rcl}\mathsf{Ana}(\mathsf{crypt}(k,m)) & = & (\{\mathsf{inv}(k)\},\{m\})\\ \mathsf{Ana}(\mathsf{sign}(k,m)) & = & (\emptyset,\{m\})\end{array}$$

We will also later use some transparent functions:

$$\begin{array}{rcl}\mathsf{Ana}(\mathsf{pair}(t,t')) & = & (\emptyset,\{t,t'\})\\ \mathsf{Ana}(\mathsf{update}(s,t,u,v)) & = & (\emptyset,\{s,t,u,v\})\end{array}$$

For all other terms t: $\mathsf{Ana}(t)=(\emptyset,\emptyset)$.

The constraints of this chapter are in fact identical to the constraints of Chapter 4. Hence all notions defined for constraints in Chapter 4 carry over to the present chapter. Also, since constraint well-formedness is such a crucial requirement we will only work with well-formed constraints throughout the chapter.

5.1.1 Protocol Syntax and Semantics

Let us now take the chance to define a slightly more detailed protocol model, based on the protocol model of Chapter 4. Protocols are defined as sets $\mathcal{P} = \{R_1, \ldots\}$ of *transaction rules* of the form:

$$R_i = \forall x_1 \in T_1, \ldots, x_n \in T_n.\ \mathsf{new}\ y_1, \ldots, y_m.\ S$$

where S is a *transaction strand*, i.e., of the form

$$\mathsf{receive}(t_1).\cdots.\mathsf{receive}(t_k).\phi_1 \cdots .\phi_{k'}.\mathsf{send}(t'_1).\cdots.\mathsf{send}(t'_{k''})$$

where

$$\phi \quad ::= \quad t \doteq t' \mid \forall \bar{x}.\ t \not\doteq t' \mid t \mathrel{\dot\in} t' \mid \forall \bar{x}.\ t \mathrel{\dot\notin} t' \mid \mathsf{insert}(t, t') \mid \mathsf{delete}(t, t')$$

The prefix $\forall x_1 \in T_1, \ldots, x_n \in T_n$ denotes that the transaction strand S is applicable for instantiations σ of the x_i variables where $\sigma(x_i) \in T_i$. The construct $\mathsf{new}\ y, \ldots, y_m$ represents that the occurrences of the variables y_i in the transaction strand S will be instantiated with fresh terms. We extend $\mathit{trms}(\cdot)$ and $\mathit{setops}(\cdot)$ to transactions strands, rules, and protocols as expected.

We define a transition relation $\Rightarrow^{\bullet}_{\mathcal{P}}$ for protocol $\mathcal{P}$ where states are constraints and the initial state is the empty constraint 0. First we define the *dual* of a transaction strand S, written $\mathit{dual}(S)$, as "swapping" the direction of the sent and received messages of S: $\mathit{dual}(\mathsf{send}(t).S) = \mathsf{receive}(t).\mathit{dual}(S)$, $\mathit{dual}(\mathsf{receive}(t).S) = \mathsf{send}(t).\mathit{dual}(S)$, and otherwise $\mathit{dual}(\mathfrak{s}.S) = \mathfrak{s}.\mathit{dual}(S)$. The transition $\mathcal{A} \Rightarrow^{\bullet}_{\mathcal{P}} \mathcal{A}.\mathit{dual}(\alpha(\sigma(S)))$ is then applicable if the following conditions are met:

1. $(\forall x_1 \in T_1, \ldots, x_n \in T_n.\ \mathsf{new}\ y_1, \ldots, y_m.\ S) \in \mathcal{P}$,
2. $\mathit{dom}(\sigma) = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$,
3. $\sigma(x_i) \in T_i$ for all $i \in \{1, \ldots, n\}$,
4. $\sigma(y_i)$ is a fresh ground term of type $\Gamma(y_i)$ for all $i \in \{1, \ldots, m\}$, and
5. α is a variable-renaming of the variables of $\sigma(S)$ where α is well-typed and the variables in $\mathit{ran}(\alpha)$ do not occur in $\sigma(S)$.

Hence transaction rules are processed atomically, and converted into constraints, during transitions. Note that each transaction rule can be executed arbitrarily often and so we support an unbounded number of "sessions". For instance, the transaction rule $\forall A \in \mathsf{Hon}.\ \mathsf{new}\ PK.\ \mathsf{insert}(PK, \mathsf{ring}(A))$ models that each honest agent $a \in \mathsf{Hon}$ can insert one fresh key into its keyring $\mathsf{ring}(a)$ during each application of the transaction rule. This rule can be executed any number of times with any agent $a \in \mathsf{Hon}$ and a fresh value for PK each time.

We say that a constraint $\mathcal{A}$ is *reachable* in protocol $\mathcal{P}$ if $0 \Rightarrow^{\bullet\star}_{\mathcal{P}} \mathcal{A}$ where $\Rightarrow^{\bullet\star}_{\mathcal{P}}$ denotes the transitive reflexive closure of $\Rightarrow^{\bullet}_{\mathcal{P}}$. We need to ensure that these constraints are well-formed and we will therefore always assume the following sufficient requirement on the protocols $\mathcal{P}$ that we work with: for any transaction strand S occurring in any rule $\forall x_1 \in T_1, \dots, x_n \in T_n.\ \mathsf{new}\ y_1, \dots, y_m.\ S$ of $\mathcal{P}$ the constraint $dual(S)$ is well-formed w.r.t. the variables $\{x_1, \dots, x_n, y_1, \dots, y_m\}$. In other words, the variables of S must first occur in either a $\mathsf{receive}$ step, a positive check ($\doteq$, $\dot{\in}$), or be part of $\{x_1, \dots, x_n, y_1, \dots, y_m\}$.

To model goal violations of a protocol $\mathcal{P}$ we first fix a special constant unique to $\mathcal{P}$, e.g., $\mathsf{attack}_{\mathcal{P}}$. Secondly, we add the rule $\mathsf{receive}(\mathsf{attack}_{\mathcal{P}})$ to $\mathcal{P}$ that we use as a signal for when an attack has occurred. The protocol then has a (well-typed) attack if there exists a (well-typed) satisfiable reachable constraint of the form $\mathcal{A}.\mathsf{send}(\mathsf{attack}_{\mathcal{P}})$. A protocol with no attacks is *secure*.

5.1.2 Protocol Goals in the Geometric Fragment

Recall that we can model events with sets, e.g., emitting an event e amounts to inserting e into a distinguished set of events while checking whether e has previously occurred (or not) corresponds to a positive (respectively negative) set-membership check. We therefore support essentially all security properties expressible in the geometric fragment [AMMV15, Gut14]. This covers many standard reachability goals such as authentication, and it seems that any significantly richer fragment of first-order logic would be incompatible with our result. Goals in the geometric fragment are not suited for expressing privacy-type properties, i.e., where goal violations occur if the observable behavior of protocols can be distinguished, and so we currently do not support such goals.

Goal formulae in the geometric fragment of first-order logic are formulas of the form $\forall \bar{x}.\ \phi \longrightarrow \psi$ where ϕ and ψ may contain $\wedge$, $\vee$, $\exists$, and events and checks (e.g., $\dot{\in}$, $\doteq$, and event), but not $\neg$, $\longrightarrow$, and $\forall$. It is also possible to express intruder derivation constraints in the antecedent ϕ as shown in [AMMV15].

A violation of a goal formula then occurs if the negation of a goal (i.e., a formula of the form $\exists \bar{x}.\ \phi \wedge \neg\psi$) is satisfied in some execution of the protocol. Note that a formula of the form $\exists \bar{x}.\ \phi \wedge \neg\psi$ can be transformed into a logically equivalent formula of the form $\exists \bar{x}.\ \phi'_1 \vee \cdots \vee \phi'_n$ where the ϕ'_i are of the form $\phi''_1 \wedge \cdots \wedge \phi''_m$ and the ϕ''_j do not contain disjunction, conjunction, or existential quantification. In our setting we can treat each ϕ'_i as a transaction rule, namely the rule $\phi''_1.\cdots.\phi''_m.\mathsf{send}(\mathsf{attack})$ where we "emit" the attack event to express a goal violation.

There are some issues, however: In a negated goal formula there might occur subformulae of the form $\forall\bar{x}.\ \bigvee_{i=1}^{n} t_i \mathrel{\dot{\notin}} s_i \vee \bigvee_{j=1}^{m} t'_j \mathrel{\dot{\neq}} s'_j$ and we currently cannot directly express such formulae as a constraint for the case where $n > 1$. The two cases where $n \neq m$ and $n, m \in \{0, 1\}$ are obviously supported. The case where $n = 0$ and $m > 1$ is also supported: Since we are working in the free algebra—i.e., terms are equal if they are syntactically equal—the unification problem $\bigwedge_{i=1}^{m} t_i \doteq s_i$ is equivalent to $f(t_1, \ldots, t_m) \doteq f(s_1, \ldots, s_m)$ for some $f \in \Sigma^m$ that does not occur anywhere else. Hence, the problem $\forall\bar{x}.\bigvee_{i=1}^{m} t_i \mathrel{\dot{\neq}} s_i$ has exactly the same solutions as $\forall\bar{x}.\ f(t_1, \ldots, t_m) \mathrel{\dot{\neq}} f(s_1, \ldots, s_m)$, which we can express as a symbolic constraint.

The case where $n > 1$ is more problematic, because here we have to translate negative set-membership checks into inequalities for our stateful typing result (see the translation *tr* in Definition 4.10). For instance, consider a goal formula $\forall\bar{x}.\ \phi \longrightarrow \psi$ where ψ contains a subformula of the form $\exists\bar{y}.\ \bigwedge_{i=1}^{n} t_i \mathrel{\dot{\in}} s_i$ where $n > 1$. The negation of this goal formula contains the subformula $\forall\bar{y}.\bigvee_{i=1}^{n} t_i \mathrel{\dot{\notin}} s_i$. For a given database mapping $D = \{d_1, \ldots, d_k\}$ each membership check $t_i \mathrel{\dot{\notin}} s_i$ is semantically equivalent to $\bigwedge_{d\in D}(t_i, s_i) \mathrel{\dot{\neq}} d$, which we can express as the constraint $(t_i, s_i) \mathrel{\dot{\neq}} d_1.\cdots.(t_i, s_i) \mathrel{\dot{\neq}} d_k$. Hence, for a given database mapping D, we can transform $\forall\bar{y}.\bigvee_{i=1}^{n} t_i \mathrel{\dot{\notin}} s_i$ into $\forall\bar{y}.\bigvee_{i=1}^{n} \bigwedge_{d\in D}(t_i, s_i) \mathrel{\dot{\neq}} d$. During translation *tr* one can then transform the inner formula into conjunctive normal form and push the quantifier $\forall\bar{x}$ inwards, resulting in a formula of the form $\bigwedge_i \forall\bar{y}.\bigvee_j u_{i,j} \mathrel{\dot{\neq}} v_{i,j}$ which we can express as a constraint. This would solve the problem, but at the cost of a much more complex translation from stateful to stateless constraints. A better approach would be to extend our constraint language (and all related definitions and proofs) to support negative checks of the form $\forall\bar{x}.\ \phi$ where $\phi ::= \phi_1 \vee \phi_2 \mid t \mathrel{\dot{\neq}} s \mid t \mathrel{\dot{\notin}} s$.

5.2 Composition and a Running Example

The core definition of this chapter is rather simple: We define the *parallel composition* $\mathcal{P}_1 \parallel \mathcal{P}_2$ of two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ as their union: $\mathcal{P}_1 \parallel \mathcal{P}_2 \equiv \mathcal{P}_1 \cup \mathcal{P}_2$. Protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ are also referred to as the *component protocols* of the composition $\mathcal{P}_1 \parallel \mathcal{P}_2$. To express composition of more than two protocols we parameterize our theory over a set $\mathcal{L}$ of indices. The only requirement on the set $\mathcal{L}$ is that it has at least two elements and we usually use natural numbers to denote the elements of $\mathcal{L}$. The parallel composition of all protocols indexed by $\mathcal{L}$ is then denoted by $\parallel_{i\in\mathcal{L}} \mathcal{P}_i$. If $\mathcal{L} = \{1, \ldots, N\}$ for some N (i.e., if $\mathcal{L}$ is finite) then we may also use the notation $\mathcal{P}_1 \parallel \cdots \parallel \mathcal{P}_N$.

For composed protocols $\|_{i\in\mathcal{L}}\, \mathcal{P}_i$ the reachable constraints will in general contain steps originating from multiple component protocols. To keep track of where a step in a constraint originated we assign to each step a *label* ℓ. The steps that are exclusive to the i-th component are marked with i. For that reason we also refer to the indices $\mathcal{L}$ as the *protocol-specific labels.* In addition to the protocol-specific labels $\mathcal{L}$ we also have a special label $\star$ that we explain later. Labels ℓ then range over $\mathcal{L} \cup \{\star\}$ unless otherwise specified.

Let $\mathcal{A}$ be a constraint with labels and ℓ be a label, we define $\mathcal{A}|_\ell$ to be the projection of $\mathcal{A}$ to the steps labeled ℓ or $\star$ (so the $\star$-steps are kept in every projection). We extend projections to transaction rules and protocols as expected. We may also write $\mathcal{P}^\star$ instead of $\mathcal{P}|_\star$.

5.2.1 A Keyserver Example

As a running example, Figure 5.1 and Figure 5.2 define two keyserver protocols that share the same databases of valid public keys registered at the keyserver. In a nutshell, the first protocol $\mathcal{P}_{ks,1} = \{R_1^1, \ldots, R_1^{10}\}$ allows users to register public keys out of band and to update an existing key with a new one (revoking the old key in the process), while the second protocol $\mathcal{P}_{ks,2} = \{R_2^1, \ldots, R_2^{10}\}$ uses a different mechanism to register new public keys. We write $t \mathrel{\dot{\notin}} f(\_)$ for $f \in \Sigma^n$ in this example as an abbreviation of $\forall x_1, \ldots, x_n.\ t \mathrel{\dot{\notin}} f(x_1, \ldots, x_n)$.

We use here three atomic types: the type of agents Agent, public keys PubKey, and the type Attack of the attack_i constants. We partition type Agent into the honest users Hon, the dishonest users Dis, and the keyservers Ser. There are sets for authentication goals begin_1, end_1, begin_2, and end_2, and all protocol steps related to these sets are highlighted in gray; let us first ignore these.

Protocol $\mathcal{P}_{ks,1}$ In the first protocol, rule R_1^5 models that an honest user registers a new public key PK out of band (e.g., by physically visiting a registration site); this is achieved by inserting PK (in the same transaction) both into a keyring $\mathsf{ring}(A)$ for user A and into a shared database $\mathsf{valid}(A, S)$ of the user's currently valid keys. There is also a corresponding rule for dishonest users: R_1^9. Dishonest users may register in their name any key they know (hence the $\mathsf{receive}(PK)$ step), so the key is not necessarily freshly created; also we do not model a keyring for them. (Rule R_i^4 gives the intruder access to arbitrarily many fresh key pairs.)

$$\boxed{\;1\;} \equiv \begin{array}{l} 1\colon \mathsf{receive}(\mathsf{sign}(\mathsf{inv}(\mathit{PK}), \mathsf{pair}(A, \mathit{NPK}))).\ \star\colon \mathit{PK} \mathrel{\dot\in} \mathsf{valid}(A, S). \\ \star\colon \mathit{NPK} \mathrel{\dot\notin} \mathsf{valid}(\_).\ 1\colon \mathit{NPK} \mathrel{\dot\notin} \mathsf{revoked}(\_) \end{array}$$

R_1^1	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $1\colon \mathsf{receive}(\mathsf{inv}(\mathit{PK})).\ \star\colon \mathit{PK} \mathrel{\dot\in} \mathsf{valid}(A,S).\ 1\colon \mathsf{send}(\mathsf{attack}_1)$
R_1^2	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $\boxed{\;1\;}.\ \star\colon \mathit{NPK} \mathrel{\dot\notin} \mathsf{begin}_1(A,S).\ 1\colon \mathsf{send}(\mathsf{attack}_1)$
R_1^3	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $\boxed{\;1\;}.\ \star\colon \mathit{NPK} \mathrel{\dot\in} \mathsf{begin}_1(A,S).\ \star\colon \mathit{NPK} \mathrel{\dot\in} \mathsf{end}_1(A,S).$ $1\colon \mathsf{send}(\mathsf{attack}_1)$
R_1^4	$\forall A \in \mathsf{Dis}.\ \mathsf{new}\ \mathit{PK}.$ $\star\colon \mathsf{send}(\mathit{PK}).\ \star\colon \mathsf{send}(\mathsf{inv}(\mathit{PK}))$
R_1^5	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.\ \mathsf{new}\ \mathit{PK}.$ $1\colon \mathsf{insert}(\mathit{PK}, \mathsf{ring}(A)).\ \star\colon \mathsf{insert}(\mathit{PK}, \mathsf{valid}(A,S)).$ $\star\colon \mathsf{insert}(\mathit{PK}, \mathsf{begin}_1(A,S)).\ \star\colon \mathsf{insert}(\mathit{PK}, \mathsf{end}_1(A,S)).$ $\star\colon \mathsf{send}(\mathit{PK})$
R_1^6	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.\ \mathsf{new}\ \mathit{NPK}.$ $1\colon \mathit{PK} \mathrel{\dot\in} \mathsf{ring}(A).\ 1\colon \mathsf{delete}(\mathit{PK}, \mathsf{ring}(A)).$ $1\colon \mathsf{insert}(\mathit{NPK}, \mathsf{ring}(A)).\ \star\colon \mathsf{insert}(\mathit{NPK}, \mathsf{begin}_1(A,S)).$ $\star\colon \mathsf{send}(\mathit{NPK}).\ 1\colon \mathsf{send}(\mathsf{sign}(\mathsf{inv}(\mathit{PK}), \mathsf{pair}(A, \mathit{NPK})))$
R_1^7	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $\boxed{\;1\;}.\ \star\colon \mathit{NPK} \mathrel{\dot\in} \mathsf{begin}_1(A,S).\ \star\colon \mathit{NPK} \mathrel{\dot\notin} \mathsf{end}_1(A,S).$ $\star\colon \mathsf{delete}(\mathit{PK}, \mathsf{valid}(A,S)).\ \star\colon \mathsf{insert}(\mathit{NPK}, \mathsf{valid}(A,S)).$ $1\colon \mathsf{insert}(\mathit{PK}, \mathsf{revoked}(A,S)).\ \star\colon \mathsf{insert}(\mathit{NPK}, \mathsf{end}_1(A,S)).$ $\star\colon \mathsf{send}(\mathsf{inv}(\mathit{PK}))$
R_1^8	$\forall A \in \mathsf{Dis}, S \in \mathsf{Ser}.$ $\boxed{\;1\;}.\ \star\colon \mathsf{delete}(\mathit{PK}, \mathsf{valid}(A,S)).\ \star\colon \mathsf{insert}(\mathit{NPK}, \mathsf{valid}(A,S)).$ $1\colon \mathsf{insert}(\mathit{PK}, \mathsf{revoked}(A,S))$
R_1^9	$\forall A \in \mathsf{Dis}, S \in \mathsf{Ser}.$ $1\colon \mathsf{receive}(\mathit{PK}).\ \star\colon \mathit{PK} \mathrel{\dot\notin} \mathsf{valid}(\_).\ \star\colon \mathsf{insert}(\mathit{PK}, \mathsf{valid}(A,S))$
R_1^{10}	$1\colon \mathsf{receive}(\mathsf{attack}_1)$

Figure 5.1: The transaction rules of the first keyserver protocol $\mathcal{P}_{ks,1}$.

Secondly, we model a key update with revocation of old keys. To request an update of key *PK* with a newly generated key *NPK* at server S, an honest user sends *NPK* signed with *PK* as in R_1^6. (For this rule there is no equivalent for the dishonest agents, since they may produce an arbitrary update request message.)

The rule R_1^7 shows how S receives the update message from an honest agent: it checks ($\boxed{\;1\;}$) that the key *PK* is currently valid, and that *NPK* is neither

$\boxed{2} \;\equiv\;$ $2\colon \mathsf{receive}(\mathsf{crypt}(PK, \mathsf{update}(A, S, NPK, \mathsf{pw}(A, S))))$.
$2\colon PK \mathrel{\dot\in} \mathsf{pubkeys}(S)$. $2\colon NPK \mathrel{\dot\notin} \mathsf{pubkeys}(\_)$. $2\colon NPK \mathrel{\dot\notin} \mathsf{seen}(\_)$

R_2^1	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $2\colon \mathsf{receive}(\mathsf{inv}(PK))$. $\star\colon PK \mathrel{\dot\in} \mathsf{valid}(A, S)$. $2\colon \mathsf{send}(\mathsf{attack}_2)$
R_2^2	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $\boxed{2}$. $\star\colon NPK \mathrel{\dot\notin} \mathsf{begin}_2(A, S)$. $2\colon \mathsf{send}(\mathsf{attack}_2)$
R_2^3	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $\boxed{2}$. $\star\colon NPK \mathrel{\dot\in} \mathsf{begin}_2(A, S)$. $\star\colon NPK \mathrel{\dot\in} \mathsf{end}_2(A, S)$. $2\colon \mathsf{send}(\mathsf{attack}_2)$
R_2^4	$\forall A \in \mathsf{Dis}.$ new PK. $\star\colon \mathsf{send}(PK)$. $\star\colon \mathsf{send}(\mathsf{inv}(PK))$
R_2^5	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ new NPK. $2\colon PK \mathrel{\dot\in} \mathsf{pubkeys}(S)$. $\star\colon \mathsf{insert}(NPK, \mathsf{begin}_2(A, S))$. $\star\colon \mathsf{send}(NPK)$. $2\colon \mathsf{send}(\mathsf{crypt}(PK, \mathsf{update}(A, S, NPK, \mathsf{pw}(A, S))))$
R_2^6	$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$ $\boxed{2}$. $\star\colon NPK \mathrel{\dot\in} \mathsf{begin}_2(A, S)$. $\star\colon NPK \mathrel{\dot\notin} \mathsf{end}_2(A, S)$. $\star\colon \mathsf{insert}(NPK, \mathsf{valid}(A, S))$. $\star\colon \mathsf{insert}(NPK, \mathsf{end}_2(A, S))$. $2\colon \mathsf{insert}(NPK, \mathsf{seen}(A))$
R_2^7	$\forall A \in \mathsf{Dis}, S \in \mathsf{Ser}.$ $2\colon \mathsf{send}(\mathsf{pw}(A, S))$
R_2^8	$\forall A \in \mathsf{Dis}, S \in \mathsf{Ser}.$ $\boxed{2}$. $\star\colon \mathsf{insert}(PK, \mathsf{valid}(A, S))$. $2\colon \mathsf{insert}(PK, \mathsf{seen}(A))$
R_2^9	$\forall S \in \mathsf{Ser}.$ new PK. $2\colon \mathsf{insert}(PK, \mathsf{pubkeys}(S))$. $\star\colon \mathsf{send}(PK)$
R_2^{10}	$2\colon \mathsf{receive}(\mathsf{attack}_2)$

Figure 5.2: The transaction rules of the second keyserver protocol $\mathcal{P}_{ks,2}$.

registered as valid or revoked. If so, it updates its databases accordingly: it moves the old key from $\mathsf{valid}(A, S)$ to $\mathsf{revoked}(A, S)$ and registers the new key NPK by inserting it into $\mathsf{valid}(A, S)$. Also, we reveal here $\mathsf{inv}(PK)$, in order to specify that the protocol must even be secure when old private keys are leaked. This is an example of declassification of a secret shared between two protocols: after intentionally revealing $\mathsf{inv}(PK)$ it should no longer count as a secret. The rule R_1^8 is the pendant for dishonest agents. The last rule R_1^{10} acts as a signal for when an attack has occurred in $\mathcal{P}_{ks,1}$.

Protocol $\mathcal{P}_{ks,2}$ The second protocol has another mechanism to register new keys: every user has a password $\mathsf{pw}(A, S)$ with the server (the dishonest agents reveal their password to the intruder with rule R_2^7). Instead of using a (possibly weak) password for an encryption, the registration message is encrypted with the public key of the server (rule R_2^5). For uniformity, we model the server's public keys in a set $\mathsf{pubkeys}(S)$ that is initialized with rule R_2^9 (in fact, the server may thus have multiple public keys). Rule R_2^6 models how the server receives a registration request (in case of honest users): to protect against replay, the server uses a set seen of seen keys (this may in a real implementation be a buffer-timestamp mechanism). Rule R_2^8 is the pendant for the dishonest users. Finally, the rule R_2^{10} acts as a signal for when an attack has occurred in $\mathcal{P}_{ks,2}$.

Authentication Besides the secrecy goal R_i^1 that no valid private key of an honest agent may ever be known by the intruder, the crucial authentication goal is that all insertions into $\mathsf{valid}(A, S)$ for honest A are authenticated. The classical injective agreement is modeled by the steps highlighted in gray: when an honest agent generates a fresh key for a server, it inserts it into a special set begin, and whenever a server accepts a key that appears to come from an honest agent A, then it inserts it into a special set end. (Note that these sets exist only in our model to specify the goals.) It is a violation of non-injective agreement if the server accepts a key that is not in begin (rule R_i^2), and of injective agreement if the server accepts a key that is already in end (rule R_i^3).

Such a specification is more declarative when one separates the protocol rules from the attack rules, but that has one drawback: if the protocol indeed had an attack, then one would allow the server to actually insert an unauthenticated key into its database and then in the next step the attack rule fires. For the composition result, however, we want that each protocol can rely on the other protocols to never insert unauthenticated keys into the database. This is why we integrate in rules R_i^6 of each protocol the checks that we are in an authenticated case (otherwise, the rules R_i^2 or R_i^3 fire). This is similar to a "lookahead" where we prevent the execution of a transition if it leads to an attack, and directly trigger an attack. This computation of the lookahead version of goals may of course be lifted from the user by verification tools.

5.3 The Compositionality Results

With stateful protocols and parallel composition defined we can now formally define the concepts underlying our results and state our compositionality theorems. We first provide a result on the level of constraints and afterwards show our main theorems for stateful protocols.

5.3.1 Protocol Abstraction

Note that all steps containing the valid set family in our keyserver example have been labeled with $\star$. Labeling operations on the shared sets with $\star$ is actually an important part of our compositionality result and we now explain why.

Essentially, compositionality results aim to prevent that attacks can arise from the composition itself, i.e., attacks that do not similarly work on the components in isolation. Thus we want to show that attacks on the composed system can be sufficiently decomposed into attacks on the components. This however cannot directly work if the components have shared sets like valid in the example: if one protocol inserts something to a set and the other protocol reads from the set, then this trace in general does not have a counter-part in the second protocol alone. We thus need a kind of *interface* to how the two protocols can influence their shared sets. In the keyserver example, both protocols can insert public keys into the shared set valid, the first protocol can even remove them. The idea is now that we develop from each protocol an *abstract* version that subsumes all the modifications that the concrete protocol can perform on the shared sets. This can be regarded as a "contract" for the composition: each protocol *guarantees* that it will not make any modifications that are not covered by its abstract protocol, and it will *assume* that the other protocol only makes modifications covered by the other protocol's abstraction. We will still have to verify that each individual protocol is also secure when running together with the other abstract protocol, but this is in general much simpler than the composition of the two concrete protocols. (In the special case that the protocols share no sets, i.e., like in all previous parallel composition results, the abstractions are empty, i.e., we have to verify only the individual components.)

In general, the abstraction of a component protocol $\mathcal{P}$ is defined by restriction to those steps that are labeled $\star$, i.e., $\mathcal{P}^\star$. We require that at least the modification of shared sets are labeled $\star$. In the keyserver example we have also labeled the operations on the authentication-related sets with a $\star$ (everything highlighted in gray): we need to ensure that we insert into the set of valid keys of an honest agent only those keys that really have been created by that agent and that have not been previously inserted. So the contract between the two protocols is that they only insert keys that are properly authenticated, but the abstraction ignores how each protocol achieves the authentication (e.g., signatures vs. passwords and seen-set). There are also some outgoing messages labeled with $\star$ which we discuss a little below.[1]

[1] We require also well-formedness of the $\star$-projected protocols. This is violated, for instance, if a protocol contains a rule where only outgoing messages are labeled $\star$ and these messages contain variables. However, given that the concrete protocol is already well-formed, this is easy to fix automatically, transparent to the user.

Example 5.2 *Consider the abstractions of rules R_2^5 and R_2^6:*

$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.\ \mathsf{new}\ \mathit{NPK}.$
$\quad \star\colon \mathsf{insert}(\mathit{NPK}, \mathsf{begin}_2(A,S)).$
$\quad \star\colon \mathsf{send}(\mathit{NPK})$

$\forall A \in \mathsf{Hon}, S \in \mathsf{Ser}.$
$\quad \star\colon \mathit{NPK} \mathrel{\dot\in} \mathsf{begin}_2(A,S).$
$\quad \star\colon \mathit{NPK} \mathrel{\dot\notin} \mathsf{end}_2(A,S).$
$\quad \star\colon \mathsf{insert}(\mathit{NPK}, \mathsf{valid}(A,S)).$
$\quad \star\colon \mathsf{insert}(\mathit{NPK}, \mathsf{end}_2(A,S))$

Notice that the gray steps prevent unauthenticated key registration because keys can only be registered if inserted into begin_2 *by an honest agent. If we* did not *ensure such authenticated key-registration then the intruder would be able to register arbitrary keys in $\mathcal{P}^\star_{ks,2}$. This would lead to an attack on secrecy in the protocol $\mathcal{P}_{ks,1} \parallel \mathcal{P}^\star_{ks,2}$.*

One may wonder why there is no similar specification for secrecy, i.e., that $\mathsf{inv}(\mathit{NPK})$ *is secret for every key NPK that is being inserted into valid. In fact, below we will declare all private keys to be secret by default. Thus, unless explicitly declassified, they are (implicitly) required to be secret.*

5.3.2 Shared Terms

Before giving the compositionality conditions we first formally define what terms can be shared: Every term t that occurs in multiple component protocols must be either a *basic public term* (meaning that the intruder can derive t without prior knowledge, i.e., $\emptyset \vdash t$) or a *shared secret*. If the intruder learns a shared secret (that has not been explicitly declassified) then it is considered a violation of secrecy in *all* component protocols. For instance, agent names are usually basic public terms whereas private keys are secrets. In fact, we will have that *all* shared terms (except basic public terms) are by default secrets—even public keys—before they are declassified.

Let *Sec* be a set of ground terms, representing the shared terms of the protocols that are initially all secret. Note that the set of shared secrets *Sec* is not a fixed predefined set of terms, but rather just another parameter to our compositionality conditions. To make matters smooth, we require that $\mathit{Sec} \cup \{t \mid \emptyset \vdash t\}$ is closed under subterms (which is trivially the case for the basic public terms). We require that all shared terms of the protocols are either in *Sec* or basic public terms. To precisely define this requirement, we first define the *ground sub-message patterns (GSMP)* of a set of terms M as follows:

$$\mathit{GSMP}(M) \equiv \{t \in \mathit{SMP}(M) \mid \mathit{fv}(t) = \emptyset\}$$

This definition is extended to constraints $\mathcal{A}$ as the following set:

$$\mathit{GSMP}(\mathcal{A}) \equiv \mathit{GSMP}(\mathit{trms}(\mathcal{A}) \cup \mathit{setops}(\mathcal{A}))$$

It is similarly extended to protocols.

Example 5.3 *We will typically study the ground subterms of each individual protocol in parallel with the abstraction of the other. For the example, the set* $\mathit{GSMP}(\mathcal{P}_{ks,1} \parallel \mathcal{P}^{\star}_{ks,2})$ *is the closure under subterms of the following set:*

$$\begin{aligned}&\{\mathsf{attack}_1, (pk, \mathsf{ring}(a)), (pk, \mathsf{valid}(a, s)), (pk, \mathsf{revoked}(a, s)), (pk, \mathsf{begin}_i(a, s)),\\ &\ (pk, \mathsf{end}_i(a, s)), \mathsf{sign}(\mathsf{inv}(pk), \mathsf{pair}(a, npk)) \mid i \in \{1, 2\}, pk, npk, a, s \in \mathcal{C},\\ &\ \ \Gamma(pk) = \mathsf{PubKey}, \Gamma(npk) = \mathsf{PubKey}, \Gamma(a) = \mathsf{Agent}, \Gamma(s) = \mathsf{Agent}\}\end{aligned}$$

and $\mathit{GSMP}(\mathcal{P}^{\star}_{ks,1} \parallel \mathcal{P}_{ks,2})$ *is the closure under subterms of the following set:*

$$\begin{aligned}&\{\mathsf{attack}_2, (pk, \mathsf{valid}(a, s)), (pk, \mathsf{seen}(a, s)), (pk, \mathsf{begin}_i(a, s)), (pk, \mathsf{end}_i(a, s)),\\ &\ (pk, \mathsf{pubkeys}(s)), \mathsf{inv}(pk), \mathsf{crypt}(pk, \mathsf{update}(a, s, npk, \mathsf{pw}(a, s))) \mid i \in \{1, 2\},\\ &\ \ pk, npk, a, s \in \mathcal{C}, \Gamma(\{pk, npk\}) = \{\mathsf{PubKey}\}, \Gamma(\{a, s\}) = \{\mathsf{Agent}\}\}\end{aligned}$$

For composition we will require that two protocols are disjoint in their ground sub-message patterns except for basic public terms and shared secrets:

Definition 5.1 (GSMP disjointness) Given two sets of terms M_1 and M_2, and a ground set of terms Sec (the shared secrets), we say that M_1 and M_2 are *Sec-GSMP disjoint* iff

$$\mathit{GSMP}(M_1) \cap \mathit{GSMP}(M_2) \subseteq \mathit{Sec} \cup \{t \mid \emptyset \vdash t\}$$

Furthermore, given two constraints $\mathcal{A}_1$ and $\mathcal{A}_2$ we say that they are Sec-GMSP disjoint iff the sets $\mathit{trms}(\mathcal{A}_1) \cup \mathit{setops}(\mathcal{A}_1)$ and $\mathit{trms}(\mathcal{A}_2) \cup \mathit{setops}(\mathcal{A}_2)$ are Sec-GSMP disjoint. This is similarly extended to protocols as expected.

5.3.3 Declassification and Leaking

Up until now the set of shared secrets has been static. We now remove this restriction by introducing a notion of declassification that will allow shared secrets to become public during protocol execution. For instance, in protocol $\mathcal{P}_{ks,1}$ we give revoked private keys of the form $\mathsf{inv}(PK)$ to the intruder by transmitting them over the network: $\mathsf{send}(\mathsf{inv}(PK))$. The transmitted key $\mathsf{inv}(PK)$ should no longer be secret after transmission and so we call such steps *declassification*. Since declassification involves shared secrets we require that they are declassified for all component protocols together. Thus we label them with $\star$.

For any constraint $\mathcal{A}$ with model $\mathcal{I}$ we can now formally define the set of secrets that has been declassified in $\mathcal{A}$ under $\mathcal{I}$:

Definition 5.2 (Declassification) Let $\mathcal{A}$ be a labeled constraint and $\mathcal{I}$ a model of $\mathcal{A}$. Then

$$\mathit{declassified}(\mathcal{A},\mathcal{I}) \equiv \mathcal{I}(\{t \mid \star\colon \mathsf{receive}(t) \text{ occurs in } \mathcal{A}\})$$

is the set of *declassified secrets of $\mathcal{A}$ under $\mathcal{I}$.*

Given a protocol $\mathcal{P}$, a reachable constraint $\mathcal{A}$ (i.e., $0 \Rightarrow^{\bullet\star}_{\mathcal{P}} \mathcal{A}$), and a model $\mathcal{I}$ of $\mathcal{A}$, then $\mathcal{I}(\mathcal{A})$ represents a concrete protocol run and the set $\mathit{declassified}(\mathcal{A},\mathcal{I})$ represents the messages that have been declassified by honest agents during the protocol run. Note that in this definition we have reversed the direction of the declassification transmission, because the send and $\mathsf{receive}$ steps of reachable constraints are duals of the transaction rules they originated from.

Declassification also allows us to share terms that have shared secrets as subterms but which are not themselves meant to be secret. For instance, public key certificates have as subterm the private key of the signing authority, and such certificates can be shared between protocols by modeling them as shared secrets that are declassified when first published.

Finally, if the intruder learns a secret that has not been declassified then it counts as an attack. We say that protocol $\mathcal{P}$ *leaks* a secret s if there is a reachable satisfiable constraint $\mathcal{A}$ where the intruder learns s before it is declassified:

Definition 5.3 (Leakage) Let Sec be a set of secrets and $\mathcal{I}$ be a model of the labeled constraint $\mathcal{A}$. *$\mathcal{A}$ leaks a secret from Sec under $\mathcal{I}$* iff there exists $s \in \mathit{Sec} \setminus \mathit{declassified}(\mathcal{A},\mathcal{I})$ and a protocol-specific label $\ell \in \mathcal{L}$ such that $\mathcal{I} \models_s \mathcal{A}|_\ell.\mathsf{send}(s)$.

Our notion of leakage requires that one of the components in isolation leaks a secret. This may seem like an undue restriction (that it counts only as a leakage if one protocol alone can leak), but we will make this as one of the prerequisites of composition, i.e., a quite weak requirement that can be checked for each protocol in isolation. Then the compositionality result ensures that the composition does not leak the shared secrets. Note also that the set $\mathit{declassified}(\mathcal{A},\mathcal{I})$ is unchanged during projection of $\mathcal{A}$, and so it suffices to pick the leaked s from the set $\mathit{Sec} \setminus \mathit{declassified}(\mathcal{A},\mathcal{I})$ instead of $\mathit{Sec} \setminus \mathit{declassified}(\mathcal{A}|_i,\mathcal{I})$.

Example 5.4 *The terms occurring in the GSMP intersection of the two key-server protocols are (a) public keys* pk*, (b) private keys of the form* $\mathsf{inv}(\mathsf{pk})$*, (c) agent names, and (d) operations on the shared set families* valid*,* begin_i*, and* end_i*. Agent names are basic public terms in our example, i.e.,* $\emptyset \vdash \mathsf{a}$ *for all constants* a *of type* Agent*. The public keys are initially secret, but we immediately*

Definition 5.4 (Parallel composability, for constraints) Let $\mathcal{A}$ be a constraint and let Sec be a ground set of terms. Define the set of labeled set operations of $\mathcal{A}$ as follows:

$$\begin{aligned} \mathit{labeledsetops}(\mathcal{A}) \equiv \{\ell\colon (t,s) \mid{} & \ell\colon \mathsf{insert}(t,s) \text{ or } \ell\colon \mathsf{delete}(t,s) \\ & \text{or } \ell\colon t \mathrel{\dot\in} s \text{ or } \ell\colon (\forall \bar{x}.\ t \mathrel{\dot{\notin}} s) \text{ occurs in } \mathcal{A}\} \end{aligned}$$

(This definition is extended to protocols as expected.)
Then $(\mathcal{A}, Sec)$ is *parallel composable* iff

1. if $\ell \neq \ell'$ then $\mathcal{A}|_\ell$ and $\mathcal{A}|_{\ell'}$ are Sec-GSMP disjoint, for all protocol-specific labels $\ell, \ell' \in \mathcal{L}$,
2. for all terms t the step $\star\colon \mathsf{send}(t)$ does not occur in $\mathcal{A}$,
3. for all $s \in Sec$ with $s' \sqsubseteq s$, it holds that either $\emptyset \vdash s'$ or $s' \in Sec$,
4. for all $\ell\colon (t,s), \ell'\colon (t',s') \in \mathit{labeledsetops}(\mathcal{A})$, if (t,s) and (t',s') are unifiable then $\ell = \ell'$,
5. $\mathcal{A}$ is type-flaw resistant and $\mathcal{A}$ and $\mathcal{A}|_\ell$ are well-formed, for every label $\ell \in \mathcal{L} \cup \{\star\}$.

Figure 5.3: The composability requirements for constraints.

declassify them whenever they are generated. To satisfy GSMP disjointness of $\mathcal{P}_{ks,1} \parallel \mathcal{P}^\star_{ks,2}$ and $\mathcal{P}^\star_{ks,1} \parallel \mathcal{P}_{ks,2}$ it thus suffices to choose the following set as the set of shared secrets (where the sec_f are special secret constants used in the encoding of the private function symbol f):

$$\begin{aligned} Sec = {} & \{pk, \mathsf{inv}(pk), (pk, f(a,s)), f(a,s), \mathsf{sec}_{\mathsf{inv}}, \mathsf{sec}_f \mid \Gamma(\{a,s\}) = \{\mathsf{Agent}\}, \\ & \Gamma(pk) = \mathsf{PubKey}, f \in \{\mathsf{valid}, \mathsf{begin}_1, \mathsf{end}_1, \mathsf{begin}_2, \mathsf{end}_2\}, pk, a, s \in \mathcal{C}\} \end{aligned}$$

Note that we want the set symbols like valid *to be private. This is because terms like* (pk, valid(a, s)) *occurs as a GSMP term in both component protocols and so we have to prevent the intruder from constructing such terms even after declassification of keys* pk*. Hence we model the set expressions like* valid(a, s) *as secrets to prevent the intruder from constructing* (pk, valid(a, s)) *when the intruder knows the constants* pk*,* a*, and* s.

5.3.4 Parallel Compositionality for Constraints

With these concepts defined we can list the requirements on constraints that are necessary to apply our result on the constraint level—see Definition 5.4.

The first requirement is at the core of our compositionality result and states that the protocols can only share basic public terms and shared secrets. The second requirement ensures that $\star$ steps are only used for declassification, checks, and stateful steps. The third condition is the requirement on the shared terms; it ensures that the set $Sec \cup \{t \mid \emptyset \vdash t\}$ is closed under subterms. The fourth condition is our requirement on stateful protocols; it implies that shared sets must be labeled with a $\star$. Finally, the last condition is needed to apply the typing result and it is orthogonal to the other conditions; it is indeed only necessary so that we can apply Theorem 4.14 and restrict ourselves to well-typed attacks. Typing results with different requirements could potentially be used instead.

Note that we require well-formedness of *all* projections of $\mathcal{A}$, including the $\star$-projection. This is because we usually consider constraints reachable in composed and augmented protocols (i.e., protocols composed with abstracted protocols), and we need well-formedness to apply the typing result to these constraints.

With the composability requirements defined we can state our main result on the constraint-level:

Theorem 5.5 *If $(\mathcal{A}, Sec)$ is parallel composable and $\mathcal{I} \models_s \mathcal{A}$ then there exists a well-typed interpretation $\mathcal{I}_\tau$ such that either $\mathcal{I}_\tau \models_s \mathcal{A}|_\ell$ for all protocol-specific labels $\ell \in \mathcal{L}$ or some prefix $\mathcal{A}'$ of $\mathcal{A}$ leaks a secret from Sec under $\mathcal{I}_\tau$.*

That is, we can obtain a well-typed model of all projections $\mathcal{A}|_\ell$, $\ell \in \mathcal{L}$, for satisfiable parallel composable constraints $\mathcal{A}$—or one of the projections has leaked a secret. In other words, if we can verify that a parallel composable constraint $\mathcal{A}$ does not have any well-typed model of all protocol-specific projections, and no prefix of $\mathcal{A}$ leaks a secret under any well-typed model, then it is unsatisfiable, i.e., there is no "attack".

5.3.5 Parallel Compositionality for Protocols

Until now our parallel compositionality result has been stated on the level of constraints. As a final step we now explain how we can use Theorem 5.5 to prove a parallel compositionality result for protocols.

First, we define the *traces* of a protocol $\mathcal{P}$ as the set of reachable constraints:

$$traces(\mathcal{P}) \equiv \{\mathcal{A} \mid 0 \Rightarrow^{\bullet\star}_{\mathcal{P}} \mathcal{A}\}$$

We then define a compositionality requirement on protocols that ensures that all traces are parallel composable: Definition 5.6. Similar to the conditions

Definition 5.6 (Parallel composability, for protocols) Let $\|_{i\in\mathcal{L}}\ \mathcal{P}_i$ be a composed protocol and let Sec be a ground set of terms. Then $(\|_{i\in\mathcal{L}}\ \mathcal{P}_i, \mathit{Sec})$ is *parallel composable* iff

1. for all protocol-specific labels $\ell, \ell' \in \mathcal{L}$, if $\ell \neq \ell'$ then $\|_{i\in\mathcal{L}}\ \mathcal{P}_i|_\ell$ and $\|_{i\in\mathcal{L}}\ \mathcal{P}_i|_{\ell'}$ are Sec-GSMP disjoint,
2. for all terms t the step $\star\colon \mathsf{receive}(t)$ does not occur in $\|_{i\in\mathcal{L}}\ \mathcal{P}_i$,
3. for all $s \in \mathit{Sec}$ with $s' \sqsubseteq s$, it holds that either $\emptyset \vdash s'$ or $s' \in \mathit{Sec}$,
4. for all $\ell\colon (t,s), \ell'\colon (t',s') \in \mathit{labeledsetops}(\|_{i\in\mathcal{L}}\ \mathcal{P}_i)$, if (t,s) and (t',s') are unifiable then $\ell = \ell'$,
5. $\|_{i\in\mathcal{L}}\ \mathcal{P}_i$ is type-flaw resistant and $\mathcal{P}_\ell$ and $\mathcal{P}_\ell^\star$ are well-formed, for every protocol-specific label $\ell \in \mathcal{L}$.

Figure 5.4: The composability requirements for protocols.

for constraints we require that each pair of different component protocols that are parallel composed with the abstraction of the others (i.e., $\|_{i\in\mathcal{L}}\ \mathcal{P}_i|_\ell$ is the component protocol ℓ composed with the abstraction of the other protocols) are Sec-GSMP disjoint. We also require type-flaw resistance and well-formedness.

Note that, for a protocol $\mathcal{P}$, the terms occurring in $\mathcal{P}^\star$ is a subset of the terms occurring in $\mathcal{P}$. So for composed protocols $\|_{i\in\mathcal{L}}\ \mathcal{P}_i$ and any protocol-specific label $\ell \in \mathcal{L}$ the terms occurring in the projected protocol $\|_{i\in\mathcal{L}}\ \mathcal{P}_i|_\ell$ is equal to the terms occurring in the protocol $\mathcal{P}_\ell \parallel \mathcal{P}_1^\star \parallel \cdots \parallel \mathcal{P}_{\ell-1}^\star \parallel \mathcal{P}_{\ell+1}^\star \parallel \mathcal{P}_{\ell+2}^\star \parallel \cdots$. When $\mathcal{L} = \{1,2\}$ the first condition of Definition 5.6 becomes the requirement that $\mathcal{P}_1 \parallel \mathcal{P}_2^\star$ and $\mathcal{P}_1^\star \parallel \mathcal{P}_2$ are Sec-GSMP disjoint.

For protocols we also need to require that their composition is type-flaw resistant. It is not sufficient to simply require it for the component protocols in isolation; unifiable messages from different protocols might break type-flaw resistance otherwise. Note also that type-flaw resistance of a protocol $\mathcal{P}$ implies that the traces of $\mathcal{P}$ are type-flaw resistant, because $\mathit{SMP}(\mathcal{A}) \subseteq \mathit{SMP}(\mathcal{P})$ for any $\mathcal{A} \in \mathit{traces}(\mathcal{P})$ and because the traces consists of the duals of the transaction strands occurring in the protocol; likewise for GSMP disjointness. Thus if $(\|_{i\in\mathcal{L}}\ \mathcal{P}_i, \mathit{Sec})$ is parallel composable then $(\mathcal{A}, \mathit{Sec})$ is parallel composable for any $\mathcal{A} \in \mathit{traces}(\|_{i\in\mathcal{L}}\ \mathcal{P}_i)$.

Example 5.5 *Continuing Example 5.4 we now show that* $\mathcal{P}_{ks,1} \parallel \mathcal{P}_{ks,2}$ *is parallel composable, i.e., that it satisfies the conditions of Definition 5.6. We have previously shown type-flaw resistance and well-formedness for a similar key-*

server protocol (see Chapter 4) and so we focus on the remaining four conditions here. GSMP disjointness of the composed keyserver protocols was explained in Example 5.4. Hence the first condition of Definition 5.6 is satisfied. Conditions two and three are satisfied since $\mathcal{P}_{ks,1} \parallel \mathcal{P}_{ks,2}$ *does not contain any steps of the form* $\star\colon \mathsf{receive}(t)$ *and since any subterm of a term from Sec (as defined in the previous example) is either in Sec or an agent name (a basic public term). Note that* $labeledsetops(\mathcal{P}_{ks,1} \parallel \mathcal{P}_{ks,2})$ *consists of instances of labeled terms from the following set:*

$$\begin{aligned}\{&1\colon (PK_0, \mathsf{ring}(A_0)), 1\colon (PK_1, \mathsf{revoked}(A_1, S_1)),\\ &2\colon (PK_2, \mathsf{seen}(A_2, S_2)), \star\colon (PK_3, \mathsf{valid}(A_3, S_3)),\\ &\star\colon (PK_4^i, \mathsf{begin}_i(A_4^i, S_4^i)), \star\colon (PK_5^i, \mathsf{end}_i(A_5^i, S_5^i)) \mid i \in \{1, 2\}\}\end{aligned}$$

For all pairs $\ell\colon (t, s)$, $\ell'\colon (t', s')$ *in this set we have that* $\ell = \ell'$ *if* (t, s) *and* (t', s') *are unifiable. Hence condition 4 is satisfied.*

As a consequence of Theorem 5.5 we have that any protocol $\mathcal{P}_1$ can be safely composed with another protocol $\mathcal{P}_2$ provided that $\mathcal{P}_1 \parallel \mathcal{P}_2^\star$ is secure and that $\mathcal{P}_1^\star \parallel \mathcal{P}_2$ does not leak a secret:

Theorem 5.7 [2] *If* $(\mathcal{P}_1 \parallel \mathcal{P}_2, Sec)$ *is parallel composable,* $\mathcal{P}_1 \parallel \mathcal{P}_2^\star$ *is well-typed secure in isolation, and* $\mathcal{P}_1^\star \parallel \mathcal{P}_2$ *does not leak a secret under any well-typed model, then all goals of* $\mathcal{P}_1$ *hold in* $\mathcal{P}_1 \parallel \mathcal{P}_2$ *(even in the untyped model).*

Note that the only requirement on protocol $\mathcal{P}_2$ is that it does not leak any secrets (before declassifying), but we do not require that $\mathcal{P}_2$ is completely secure. This means, if we have a secure protocol $\mathcal{P}_1$, that the goals of $\mathcal{P}_1$ continue to hold in any composition with another protocol $\mathcal{P}_2$ that satisfies the composability conditions and does not leak secrets, even if $\mathcal{P}_2$ has some attacks. This is in particular interesting if we run a protocol $\mathcal{P}_1$ in composition with a large number of other protocols that are too complex to verify in all detail.

Finally, the composition of parallel composable and secure protocols is secure:

Corollary 5.8 [3] *If* $(\mathcal{P}_1 \parallel \mathcal{P}_2, Sec)$ *is parallel composable and* $\mathcal{P}_1 \parallel \mathcal{P}_2^\star$ *and* $\mathcal{P}_1^\star \parallel \mathcal{P}_2$ *are both secure in isolation then the composition* $\mathcal{P}_1 \parallel \mathcal{P}_2$ *is also secure (even in the untyped model).*

[2] Proven in Isabelle for stateless protocols. Not proven in Isabelle for stateful protocols.
[3] Proven in Isabelle for stateless protocols. Not proven in Isabelle for stateful protocols.

5.3.6 Sequential Composition

Until now we have focused entirely on parallel composition where protocols are run "side-by-side". Another type of protocol composition is sequential composition where protocols are run in sequence, e.g., most recently [CCW17] for PKIs. Thanks to the generality of our result, we can cover such sequential compositions as a parallel composition with sets dedicated to the hand-over between the protocols. Let us take a key-exchange protocols like TLS as an example, where the handshake protocol establishes a pair of shared keys between a client A and a server S, and then subsequently, the transport protocol uses these keys to encrypt communication between A and S. We illustrate how the last transition of the handshake and the first transition of the transport protocol look for A where t_1 and t_2 are terms representing the two shared keys established in the handshake (and there are similar rules for S):

```
∀A ∈ Hon, S ∈ Ser.
  1: ···
  ⋆: insert((t_1, t_2), keys(A, S))
```

```
∀A ∈ Hon, S ∈ Ser.
  ⋆: (K_1, K_2) ∈̇ keys(A, S).
  ⋆: delete((K_1, K_2), keys(A, S)).
  2: ···
```

Note that, like in the keyserver example, the set $\mathsf{keys}(A, S)$ does not represent a means of communication between two participants, but rather a buffer or glue between two protocols: one protocol is producing keys, the other protocol is consuming them. Of course, one needs to require here that the first protocol only inserts authenticated and secret keys into the set, which is similar to the assume-guarantee reasoning we have illustrated for our keyserver example.

In fact, our result allows for a generalization of existing sequential composition results: while all results like [CCW17] and the similar vertical result [GM11] are specialized to a particular set of data to be transferred from one protocol to another, our result does not prescribe a particular setup, but allows for any exchange of data through shared sets. This only requires one to specify sufficient assumptions on the shared-set operations for the assume-guarantee reasoning, but one does no longer need to establish a new composition theorem for each new form of sequential composition. In fact, the composition does not even need to be strictly sequential, e.g., if the first protocol establishes keys for the second protocol, one may well have that additionally the second protocol can also establish new keys for subsequent sessions.

5.4 Proving the Compositionality Theorem

The idea is to first prove Theorem 5.5 on the level of ordinary (or stateless) constraints. Using a variant of the reduction technique tr from Chapter 4 we then lift the theorem to stateful constraints. Finally, we apply Theorem 5.5 to prove our main results on the protocol-level, namely Theorem 5.7 and Corollary 5.8.

5.4.1 Proving the Result for Stateless Constraints

For Theorem 5.5 we need to show that for satisfiable parallel composable constraints $\mathcal{A}$ with shared secrets Sec we can obtain a well-typed model of projections $\mathcal{A}|_\ell$ for all labels $\ell \in \mathcal{L}$ or $\mathcal{A}$ has leaked a secret under some projection. In a nutshell we show that any term t occurring in a $\ell\colon \mathsf{send}(t)$ step of $\mathcal{A}$ need only to be constructed from terms of protocol ℓ, unless leakage has occurred previously. For that we first need a notion of terms belonging to a specific protocol:

Definition 5.9 Let $\mathcal{A}$ be a constraint and Sec be a set of shared secrets.

- A term t is *i-specific (w.r.t. $\mathcal{A}$ and Sec)* iff $t \in GSMP(\mathcal{A}|_i) \setminus (Sec \cup \{t \mid \emptyset \vdash t\})$ for a label i.
- A term t is *heterogeneous (w.r.t. $\mathcal{A}$ and Sec)* iff there exists protocol-specific labels $\ell_1, \ell_2 \in \mathcal{L}$ and subterms t_1 and t_2 of t such that $\ell_1 \neq \ell_2$ and each t_i is ℓ_i-specific w.r.t. $\mathcal{A}$ and Sec.
- A term t is *homogeneous (w.r.t. $\mathcal{A}$ and Sec)* if it is not heterogeneous w.r.t. $\mathcal{A}$ and Sec.

Then all ground sub-message patterns of parallel composable constraints are homogeneous:

Lemma 5.10 *If $(\mathcal{A}, Sec)$ is parallel composable and $t \in GSMP(\mathcal{A})$ then t is homogeneous.*

Proof. We can first of all obtain some protocol-specific label $i \in \mathcal{L}$ such that $t \in GSMP(\mathcal{A}|_i)$. Note also that the terms in $Sec \cup \{t \mid \emptyset \vdash t\}$ are by definition homogeneous since $Sec \cup \{t \mid \emptyset \vdash t\}$ is closed under subterms and so they cannot be protocol specific. Hence we simply need to show that if $t \in GSMP(\mathcal{A}|_i) \setminus (Sec \cup \{t \mid \emptyset \vdash t\})$ then t is homogeneous and we do so with a contradiction proof: Suppose that t is heterogeneous. Then we can obtain

Definition 5.11 (Homogeneous Intruder Deduction)

$$\frac{}{M \vdash_{\text{hom}}^{\mathcal{A},Sec} t} \;\begin{array}{c}(Axiom_{hom}),\\ t \in M\end{array}$$

$$\frac{M \vdash_{\text{hom}}^{\mathcal{A},Sec} t_1 \quad \cdots \quad M \vdash_{\text{hom}}^{\mathcal{A},Sec} t_n}{M \vdash_{\text{hom}}^{\mathcal{A},Sec} f(t_1,\dots,t_n)} \;\begin{array}{c}(Compose_{hom}), f \in \Sigma^n,\\ f \text{ public}, f(t_1,\dots,t_n) \text{ homogeneous},\\ f(t_1,\dots,t_n) \in GSMP(\mathcal{A})\end{array}$$

$$\frac{M \vdash_{\text{hom}}^{\mathcal{A},Sec} t \quad M \vdash_{\text{hom}}^{\mathcal{A},Sec} k_1 \quad \cdots \quad M \vdash_{\text{hom}}^{\mathcal{A},Sec} k_n}{M \vdash_{\text{hom}}^{\mathcal{A},Sec} t_i} \;\begin{array}{c}(Decompose_{hom}),\\ \mathsf{Ana}\ t = (K,T),\ t_i \in T,\\ K = \{k_1,\dots,k_n\}\end{array}$$

Figure 5.5: The homogeneous intruder deduction relation.

a subterm s of t and a label j where $j \neq i$ such that s is j-specific. Hence $s \in GSMP(\mathcal{A}|_j)$. Since GSMP is closed under subterms we also have that $s \in GSMP(\mathcal{A}|_i)$. Since $(\mathcal{A}, Sec)$ is parallel composable the sets $GSMP(\mathcal{A}|_i)$ and $GSMP(\mathcal{A}|_j)$ must be GSMP disjoint and so $s \in Sec \cup \{t \mid \emptyset \vdash t\}$ because it is in the intersection of the two aforementioned GSMP sets. However, s is j-specific and therefore cannot be a member of the set $Sec \cup \{t \mid \emptyset \vdash t\}$, a contradiction. □

Given a constraint $\mathcal{A}$ and a set of shared secrets Sec we now define a useful variant $\vdash_{\text{hom}}^{\mathcal{A},Sec}$ of the intruder deduction relation $\vdash$ as the restriction to homogeneous GSMP terms only—see Definition 5.11. This relation satisfies a useful property:

Lemma 5.12 *Let $(\mathcal{A}, Sec)$ be parallel composable and $t \in GSMP(\mathcal{A})$. Then $ik(\mathcal{A}) \vdash t$ iff $ik(\mathcal{A}) \vdash_{\text{hom}}^{\mathcal{A},Sec} t$.*

Proof. The idea is that deriving a term $f(t_1,\dots,t_n)$ that "falls outside of" the homogeneous GSMP terms is only possible by composition; if all the immediate subterms t_i are homogeneous GSMP terms then deriving $f(t_1,\dots,t_n)$ must have happened by an application of the $(Compose)$ rule. Usually, such proofs proceed by inspecting the derivation tree of the derivation of $f(t_1,\dots,t_n)$, and, in the case where $f(t_1,\dots,t_n)$ has been derived from decomposition, either transforming the tree to remove unnecessary decomposition steps or regress to the first decomposition step. Such arguments are cumbersome to formalize in

Isabelle/HOL since one would need a deep embedding of the derivation tree. For our purposes, however, it is sufficient to only encode the *height* of the derivation tree and so we equip the relation $\vdash$ with such a number: $M \vdash_k t$ iff k is the maximum number of applications of the (*Compose*) and (*Decompose*) rules in any path of the derivation tree for $M \vdash t$. Essentially, we prove that no matter how many steps occur in the derivation tree of $f(t_1, \ldots, t_n)$ the first time the term is derived (it might have been derived later on through decomposition) is always a composition step. □

This is useful because we can prove that all homogeneous GSMP terms can be derived purely through derivation of other homogeneous GSMP terms. In other words, for homogeneous GSMP terms such as those in parallel composable constraints we can reduce the intruder derivation problem to $\vdash^{\mathcal{A},Sec}_{\text{hom}}$.

Lemma 5.13 *Let $(\mathcal{A}, Sec)$ be parallel composable, $\mathcal{I}$ be a well-typed model of $\mathcal{A}$. Assume that no secret is homogeneously derivable in any of the projections of the constraint, i.e.,*

$$\forall i \in \mathcal{L}.\ \forall s \in Sec \setminus declassified(\mathcal{A},\mathcal{I}).\ \neg\left(ik(\mathcal{I}(\mathcal{A}|_i)) \vdash^{\mathcal{A},Sec}_{\text{hom}} s\right) \qquad (*)$$

If $ik(\mathcal{I}(\mathcal{A})) \vdash^{\mathcal{A},Sec}_{\text{hom}} t$ then

- *$t \notin Sec \setminus declassified(\mathcal{A},\mathcal{I})$, and*
- *for all $i \in \mathcal{L}$, if $t \in GSMP(\mathcal{A}|_i)$ then $ik(\mathcal{I}(\mathcal{A}|_i)) \vdash^{\mathcal{A},Sec}_{\text{hom}} t$.*

Proof. The lemma follows rather straightforwardly by induction on the homogeneous intruder deduction relation $\vdash^{\mathcal{A},Sec}_{\text{hom}}$ and using the fact that GSMP sets are closed under subterms and analysis. □

With Lemma 5.12 and Lemma 5.10 we can prove a useful consequence of Lemma 5.13:

Lemma 5.14 *Let $(\mathcal{A}, Sec)$ be parallel composable, $\mathcal{I}$ be a well-typed model of $\mathcal{A}$, $i \in \mathcal{L}$ be a protocol-specific label, and t a term such that $t \in GSMP(\mathcal{A}|_i)$. If $ik(\mathcal{I}(\mathcal{A})) \vdash t$ then either $ik(\mathcal{I}(\mathcal{A}|_i)) \vdash t$ or $\mathcal{A}$ leaks a secret from Sec.*

Proof. Proven by induction on the relation $\vdash^{\mathcal{A},Sec}_{\text{hom}}$. The only difficult case is ($Decompose_{hom}$) where we need to apply the aforementioned lemmata. □

Now we can use Lemma 5.14 to show that the models $\mathcal{I}$ of parallel composable constraints $\mathcal{A}$ are also models of the projections $\mathcal{A}|_i$, or some secret is leaked. The proof is by structural induction on the constraint $\mathcal{A}$. The only non-trivial case is where a step of the form $\mathsf{send}(t)$ occurs in $\mathcal{A}$, i.e., when a prefix of the form $\mathcal{A}'.\mathsf{send}(t)$ exists for $\mathcal{A}$. By the constraint semantics such a prefix corresponds to a derivation constraint $ik(\mathcal{I}(\mathcal{A}')) \vdash \mathcal{I}(t)$, and here we can apply Lemma 5.14. Thus:

Lemma 5.15 *Let $(\mathcal{A}, Sec)$ be parallel composable and let $\mathcal{A}$ be an ordinary constraint with a well-typed model $\mathcal{I}_\tau$. Then either $\mathcal{I}_\tau \models \mathcal{A}|_i$ for all $i \in \mathcal{L}$ or some prefix $\mathcal{A}'$ of $\mathcal{A}$ leaks a secret from Sec under $\mathcal{I}_\tau$.*

Finally, we can use the typing result Theorem 4.14 and Lemma 5.15 to relax the well-typedness assumption and prove our main result on the level of ordinary constraints:

Lemma 5.16 *Let $(\mathcal{A}, Sec)$ be parallel composable and let $\mathcal{A}$ be an ordinary constraint with model $\mathcal{I}$. Then there exists a well-typed interpretation $\mathcal{I}_\tau$ of $\mathcal{A}$ such that either $\mathcal{I}_\tau \models \mathcal{A}|_i$ for all $i \in \mathcal{L}$ or some prefix $\mathcal{A}'$ of $\mathcal{A}$ leaks a secret from Sec under $\mathcal{I}_\tau$.*

5.4.2 Proving the Result for Stateful Constraints

For stateful constraints the proof idea is to use a variant of the reduction technique introduced in Chapter 4 to reduce the compositionality problem for stateful constraints to the compositionality problem for ordinary constraints. We first make some definitions:

Definition 5.17 (Projections) Given a finite set D where

$$D = \{\ell_1 \colon (t_1, s_1), \dots, \ell_n \colon (t_n, s_n)\}$$

and where each t_i and s_i, are terms and $\ell_i \in \mathcal{L} \cup \{\star\}$ are labels, we define the *projection of D to ℓ*, written $|D|_\ell$, as follows:

$$|D|_\ell = \{\ell' \colon d \in D \mid \ell = \ell'\}$$

The constraint reduction tr^{pc} is now defined as follows:

Definition 5.18 (Translation of symbolic constraints) Given a labeled constraint $\mathcal{A}$ its translation into labeled ordinary constraints is denoted by $tr^{pc}(\mathcal{A}) = tr^{pc}_{\emptyset}(\mathcal{A})$ where:

$$
\begin{aligned}
&tr^{pc}_D(0) = \{0\}\\
&tr^{pc}_D(\ell\colon \mathsf{insert}(t,s).\mathcal{A}) = tr^{pc}_{D\cup\{\ell\colon (t,s)\}}(\mathcal{A})\\
&tr^{pc}_D(\ell\colon \mathsf{delete}(t,s).\mathcal{A}) = \{\\
&\quad \ell\colon (t,s) \doteq d_1.\cdots.\ell\colon (t,s) \doteq d_i.\ell\colon (t,s) \not\doteq d_{i+1}.\cdots.\ell\colon (t,s) \not\doteq d_n.\mathcal{A}' \mid\\
&\quad |D|_\ell = \{\ell\colon d_1,\dots,\ell\colon d_i,\dots,\ell\colon d_n\}, 0 \le i \le n, \mathcal{A}' \in tr^{pc}_{D\setminus\{\ell\colon d_1,\dots,\ell\colon d_i\}}(\mathcal{A})\}\\
&tr^{pc}_D(\ell\colon t \mathrel{\dot\in} s.\mathcal{A}) = \{\ell\colon (t,s) \doteq d.\mathcal{A}' \mid \ell\colon d \in |D|_\ell, \mathcal{A}' \in tr^{pc}_D(\mathcal{A})\}\\
&tr^{pc}_D(\ell\colon (\forall \bar{x}.\ t \mathrel{\dot\notin} s).\mathcal{A}) = \{\ell\colon (\forall \bar{x}.\ (t,s) \not\doteq d_1).\cdots.\ell\colon (\forall \bar{x}.\ (t,s) \not\doteq d_n).\mathcal{A}' \mid\\
&\quad |D|_\ell = \{\ell\colon d_1,\dots,\ell\colon d_n\}, 0 \le n, \mathcal{A}' \in tr^{pc}_D(\mathcal{A})\}\\
&tr^{pc}_D(\ell\colon \mathfrak{a}.\mathcal{A}) = \{\ell\colon \mathfrak{a}.\mathcal{A}' \mid \mathcal{A}' \in tr^{pc}_D(\mathcal{A})\} \text{ otherwise}
\end{aligned}
$$

Note that we apply projections $|D|_\ell$ when translating set operations labeled with ℓ. Hence we never "mix" two set operations with different labels in the reduction. A crucial point here is that parallel compositionality makes such mixing unnecessary, and this enables us to prove a strong relationship between translated constraints and projections:

Lemma 5.19 *Let $i \in \mathcal{L}$ be a protocol-specific label, $\mathcal{A}$ and $\mathcal{B}$ be two constraints, and let $D = \{\ell_1\colon (t_1,s_1),\dots,\ell_n\colon (t_n,s_n)\}$. If $\mathcal{B} \in tr^{pc}_D(\mathcal{A})$ then $\mathcal{B}|_i \in tr^{pc}_{|D|_i\cup|D|_\star}(\mathcal{A}|_i)$.*

Proof. The lemma follows from an induction over the structure of $\mathcal{A}$. We only show the $t \mathrel{\dot\in} s$ and $\mathsf{delete}(t,s)$ cases. All remaining cases are similarly proven.

- Case $\mathcal{A} = (\ell\colon t \mathrel{\dot\in} s).\mathcal{A}'$: In this case we know that $\mathcal{B}$ must be of the form $(\ell\colon (t,s) \doteq d).\mathcal{B}'$ for some $\ell\colon d \in |D|_\ell$ and $\mathcal{B}' \in tr^{pc}_D(\mathcal{A}')$. From the induction hypothesis we can now conclude that

  $$\mathcal{B}'|_i \in tr^{pc}_{|D|_i\cup|D|_\star}(\mathcal{A}'|_i) \tag{IH}$$

  We now show that $\mathcal{B}|_i \in tr^{pc}_{|D|_i\cup|D|_\star}(\mathcal{A}|_i)$ by a case analysis on ℓ:

  - $\ell = \star$ or $\ell = i$:

    In these cases we have that $\mathcal{A}|_i = (\ell\colon t\mathrel{\dot\in}s).(\mathcal{A}'|_i)$ and $\mathcal{B}|_i = (\ell\colon (t,s) \doteq d).(\mathcal{B}'|_i)$. We also have that $\ell\colon d \in |D|_i \cup |D|_\star$. Thus the case follows from (IH) and the definition of tr^{pc}.

  - $\ell \in \mathcal{L} \setminus \{i\}$:

    In this case we have that $\mathcal{A}|_i = \mathcal{A}'|_i$ and $\mathcal{B}|_i = \mathcal{B}'|_i$. Thus the case follows immediately from (IH).

- Case $\mathcal{A} = (\ell\colon \mathsf{delete}(t,s)).\mathcal{A}'$: In this case we know that $\mathcal{B}$ must be of the form $\ell\colon (t,s) \doteq d_1.\cdots.\ell\colon (t,s) \doteq d_j.\ell\colon (t,s) \not\doteq d_{j+1}.\cdots.\ell\colon (t,s) \not\doteq d_n.\mathcal{B}'$ for some $\mathcal{B}' \in tr^{pc}_{D'}(\mathcal{A}')$ and $0 \leq j \leq n$ where

  $$|D|_\ell = \{\ell\colon d_1,\ldots,\ell\colon d_j,\ldots,\ell\colon d_n\} \text{ and } D' = D \setminus \{\ell\colon d_1,\ldots,\ell\colon d_j\}$$

  From the induction hypothesis we can now conclude that

  $$\mathcal{B}'|_i \in tr^{pc}_{|D'|_i \cup |D'|_\star}(\mathcal{A}'|_i) \tag{IH}$$

  We now show that $\mathcal{B}|_i \in tr^{pc}_{|D|_i \cup |D|_\star}(\mathcal{A}|_i)$ by a case analysis on ℓ:

  - $\ell = \star$ or $\ell = i$:
    In these cases we have that $|D'|_i \cup |D'|_\star = (|D|_i \cup |D|_\star) \setminus \{\ell\colon d_1,\ldots,\ell\colon d_j\}$ and $\mathcal{B}|_i = (\ell\colon (t,s) \doteq d_1.\cdots.\ell\colon (t,s) \doteq d_j.\ell\colon (t,s) \not\doteq d_{j+1}.\cdots.\ell\colon (t,s) \not\doteq d_n).(\mathcal{B}'|_i)$ and $\mathcal{A}|_i = (\ell\colon \mathsf{delete}(t,s)).(\mathcal{A}|_i)$. Thus the case follows from (IH) and the definition of tr^{pc}.
  - $\ell \in \mathcal{L} \setminus \{i\}$:
    In this case we have that $\mathcal{A}|_i = \mathcal{A}'|_i$, $\mathcal{B}|_i = \mathcal{B}'|_i$, $|D'|_i = |D|_i$, and $|D'|_\star = |D|_\star$. Thus the case follows immediately from (IH).

□

Now the core idea is to reduce the compositionality problem for stateful constraints to ordinary constraints using the translation tr^{pc}. For that reason we need to show that the translation is correct, i.e., that the set of models of the input constraint is exactly the set of models of the translation:

Lemma 5.20 (Semantic equivalence of reduction) *Let $\mathcal{A}$ be a constraint and $D = \{\ell_1\colon (t_1,s_1),\ldots,\ell_n\colon (t_n,s_n)\}$. Assume that*

- *all unifiable set operations occurring in $\mathcal{A}$ and D carry the same label, i.e., if $\ell\colon (t,s), \ell'\colon (t',s') \in labeledsetops(\mathcal{A}) \cup D$ and $\exists\delta.\ \delta((t,s)) = \delta((t',s'))$ then $\ell = \ell'$, and*
- *the variables occurring in D do not occur in the bound variables of $\mathcal{A}$.*

Then the models of $\mathcal{A}$ are the same as the models of $tr^{pc}(\mathcal{A})$, i.e., $[\![M, \mathcal{I}(D); \mathcal{A}]\!]_s\ \mathcal{I}$ iff there exists $\mathcal{B} \in tr^{pc}_D(\mathcal{A})$ such that $[\![M; \mathcal{B}]\!]\ \mathcal{I}$.

Proof. This statement is very similar to Theorem 4.11. Note also that the first assumption of Lemma 5.20 is similar to Definition 5.4(4).

For this proof let us first define yet another variant of tr where we in the delete, $\dot{\in}$, and $\dot{\notin}$ cases do not project D to the current label ℓ (in contrast to tr^{pc}):

$\overline{tr}^{pc}_D(0) = \{0\}$
$\overline{tr}^{pc}_D(\ell\colon \mathsf{insert}(t,s).\mathcal{A}) = \overline{tr}^{pc}_{D\cup\{\ell\colon (t,s)\}}(\mathcal{A})$
$\overline{tr}^{pc}_D(\ell\colon \mathsf{delete}(t,s).\mathcal{A}) = \{$
$\ell_1\colon (t,s) \doteq d_1.\cdots.\ell_i\colon (t,s) \doteq d_i.\ell_{i+1}\colon (t,s) \not\doteq d_{i+1}.\cdots.\ell_n\colon (t,s) \not\doteq d_n.\mathcal{A}' \mid$
$D = \{\ell_1\colon d_1,\dots,\ell_i\colon d_i,\dots,\ell_n\colon d_n\}, 0 \le i \le n, \mathcal{A}' \in \overline{tr}^{pc}_{D\setminus\{\ell_1\colon d_1,\dots,\ell_i\colon d_i\}}(\mathcal{A})\}$
$\overline{tr}^{pc}_D(\ell\colon t \mathrel{\dot{\in}} s.\mathcal{A}) = \{\ell'\colon (t,s) \doteq d.\mathcal{A}' \mid \ell'\colon d \in D, \mathcal{A}' \in \overline{tr}^{pc}_D(\mathcal{A})\}$
$\overline{tr}^{pc}_D(\ell\colon (\forall\bar{x}.\ t \mathrel{\dot{\notin}} s).\mathcal{A}) = \{\ell_1\colon (\forall\bar{x}.\ (t,s) \not\doteq d_1).\cdots.\ell_n\colon (\forall\bar{x}.\ (t,s) \not\doteq d_n).\mathcal{A}' \mid$
$D = \{\ell_1\colon d_1,\dots,\ell_n\colon d_n\}, 0 \le n, \mathcal{A}' \in \overline{tr}^{pc}_D(\mathcal{A})\}$
$\overline{tr}^{pc}_D(\ell\colon \mathfrak{a}.\mathcal{A}) = \{\ell\colon \mathfrak{a}.\mathcal{A}' \mid \mathcal{A}' \in \overline{tr}^{pc}_D(\mathcal{A})\}$ otherwise

The theorem follows from the following two statements (the assumptions of this lemma still apply to D and $\mathcal{A}$):

$$[\![M,\mathcal{I}(D);\mathcal{A}]\!]_s\ \mathcal{I} \text{ iff } (\exists\mathcal{B}' \in \overline{tr}^{pc}_D(\mathcal{A}).\ [\![M;\mathcal{B}']\!]\ \mathcal{I}) \tag{1}$$

$$(\exists\mathcal{B} \in tr^{pc}_D(\mathcal{A}).\ [\![M;\mathcal{B}]\!]\ \mathcal{I}) \text{ iff } (\exists\mathcal{B}' \in \overline{tr}^{pc}_D(\mathcal{A}).\ [\![M;\mathcal{B}']\!]\ \mathcal{I}) \tag{2}$$

Statement (1) is actually a simple adaption of Theorem 4.11. The rest of this proof is to show statement (2) and we prove it by proving each direction of the bi-implication. Both proofs are by induction over the structure of $\mathcal{A}$ and we give the proof only for the most difficult case: delete. All remaining cases are similar. Note that the assumptions of this lemma still apply, but we skip showing the proofs of the antecedents of any induction hypothesis we use since those proofs are trivial.

1. To show:

   If $\mathcal{B} \in tr^{pc}_D(\mathcal{A})$ and $[\![M;\mathcal{B}]\!]\ \mathcal{I}$ then $[\![M;\mathcal{B}']\!]\ \mathcal{I}$ for some $\mathcal{B}' \in \overline{tr}^{pc}_D(\mathcal{A})$.

   Case $\mathcal{A} = (\ell\colon \mathsf{delete}(t,s)).\mathcal{A}_0$:
   In this case we know that $\mathcal{B}$ must be of the form:

   $$\mathcal{B} = \ell\colon (t,s) \doteq d_1.\cdots.\ell\colon (t,s) \doteq d_i.\ell\colon (t,s) \not\doteq d_{i+1}.\cdots.\ell\colon (t,s) \not\doteq d_n.\mathcal{B}_0$$

   for some $\mathcal{B}_0 \in tr^{pc}_{D\setminus\{\ell\colon d_1,\dots,\ell\colon d_i\}}(\mathcal{A}_0)$ where $|D|_\ell = \{\ell\colon d_1,\dots,\ell\colon d_n\}$ and $0 \le i \le n$. We also know that $[\![M;\mathcal{B}]\!]\ \mathcal{I}$ and therefore $[\![M;\mathcal{B}_0]\!]\ \mathcal{I}$.

From the induction hypothesis we can obtain $\mathcal{B}'_0 \in \overline{tr}^{pc}_{D\setminus\{\ell\colon d_1,\dots,\ell\colon d_i\}}(\mathcal{A}_0)$ such that $[\![M;\mathcal{B}'_0]\!]\ \mathcal{I}$. Now obtain $\ell_{k_1}, d_{k_1}, \dots, \ell_{k_m}, d_{k_m}$ such that $D \setminus |D|_\ell = \{\ell_{k_1}\colon d_{k_1}, \dots, \ell_{k_m}\colon d_{k_m}\}$. Hence $\ell \neq \ell_{k_j}$ for all $j \in \{1,\dots,m\}$ (because $|D|_\ell$ contains exactly the elements of D with label ℓ) and so $[\![M; \ell_{k_1}\colon (t,s) \mathrel{\dot{\neq}} d_{k_1} \cdots . \ell_{k_m}\colon (t,s) \mathrel{\dot{\neq}} d_{k_m}]\!]\ \mathcal{I}$ because of the unifiability assumption on the set operations of $\mathcal{A}$ and D. Let $\mathcal{B}' = \phi.\mathcal{B}'_0$ where

$$\begin{aligned}\phi \quad = \quad & \ell\colon (t,s) \doteq d_1.\cdots.\ell\colon (t,s) \doteq d_i.\ell\colon (t,s) \mathrel{\dot{\neq}} d_{i+1}.\cdots. \\ & \ell\colon (t,s) \mathrel{\dot{\neq}} d_n.\ell_{k_1}\colon (t,s) \mathrel{\dot{\neq}} d_{k_1}.\cdots.\ell_{k_m}\colon (t,s) \mathrel{\dot{\neq}} d_{k_m}\end{aligned}$$

We can then conclude that $\mathcal{B}' \in \overline{tr}^{pc}_D(\mathcal{A})$ and $[\![M;\mathcal{B}']\!]\ \mathcal{I}$.

2. To show:

If $\mathcal{B}' \in \overline{tr}^{pc}_D(\mathcal{A})$ and $[\![M;\mathcal{B}']\!]\ \mathcal{I}$ then $[\![M;\mathcal{B}]\!]\ \mathcal{I}$ for some $\mathcal{B} \in tr^{pc}_D(\mathcal{A})$.

Case $\mathcal{A} = (\ell\colon \mathsf{delete}(t,s)).\mathcal{A}_0$:
In this case we know that $\mathcal{B}'$ must be of the form:

$$\begin{aligned}\mathcal{B}' \quad = \quad & \ell_1\colon (t,s) \doteq d_1.\cdots.\ell_i\colon (t,s) \doteq d_i. \\ & \ell_{i+1}\colon (t,s) \mathrel{\dot{\neq}} d_{i+1}.\cdots.\ell_n\colon (t,s) \mathrel{\dot{\neq}} d_n.\mathcal{B}'_0\end{aligned}$$

for some $\mathcal{B}'_0 \in \overline{tr}^{pc}_{D\setminus\{\ell_1\colon d_1,\dots,\ell_i\colon d_i\}}(\mathcal{A}_0)$ where $D = \{\ell_1\colon d_1, \dots, \ell_n\colon d_n\}$ and $0 \leq i \leq n$. We also know that $[\![M;\mathcal{B}']\!]\ \mathcal{I}$ and therefore $[\![M;\mathcal{B}'_0]\!]\ \mathcal{I}$. Since (t,s) and d' are unifiable only if $\ell = \ell'$, for all $\ell'\colon d' \in D$, it must be the case that $\ell = \ell_j$ for all $j \in \{1,\dots,i\}$. We can thus apply the induction hypothesis to obtain $\mathcal{B}_0 \in tr^{pc}_{D\setminus\{\ell\colon d_1,\dots,\ell\colon d_i\}}(\mathcal{A}_0)$ where $[\![M;\mathcal{B}_0]\!]\ \mathcal{I}$. Now pick the largest subset $\{k_1,\dots,k_m\}$ of $\{i+1,\dots,n\}$ such that $\ell_{k_j} = \ell$ for all $0 \leq j \leq m$. Then $|D|_\ell = \{\ell\colon d_1, \dots \ell\colon d_i, \ell\colon d_{k_1}, \dots, \ell\colon d_{k_m}\}$. Let $\mathcal{B} = \ell\colon (t,s) \doteq d_1.\cdots.\ell\colon (t,s) \doteq d_i.\ell\colon (t,s) \mathrel{\dot{\neq}} d_{k_1}.\cdots.\ell\colon (t,s) \mathrel{\dot{\neq}} d_{k_m}.\mathcal{B}_0$. Thus $\mathcal{B} \in tr^{pc}_D(\mathcal{A})$ and $[\![M;\mathcal{B}]\!]\ \mathcal{I}$ which concludes the case.

□

By a straightforward induction proof over the structure of constraints we can prove that tr^{pc} preserves the properties we need for our compositionality result:

Lemma 5.21 *If $\mathcal{A}$ is well-formed and parallel composable, and if $\mathcal{B} \in tr^{pc}(\mathcal{A})$, then $\mathcal{B}$ is well-formed and parallel composable.*

For proving Theorem 5.5 we now only need to lift Lemma 5.16 to stateful constraints. That is, given $\mathcal{I} \models_s \mathcal{A}$ we obtain $\mathcal{B} \in tr^{pc}(\mathcal{A})$ such that $\mathcal{I} \models \mathcal{B}$. For

$\mathcal{B}$ we can apply Lemma 5.16; either $\mathcal{I}_\tau \models \mathcal{B}|_i$ for all $i \in \mathcal{L}$ or $\mathcal{B}$ leaks, for some well-typed interpretation $\mathcal{I}_\tau$. Finally, with Lemma 5.20 and Lemma 5.19 we can show that either $\mathcal{I}_\tau \models_s \mathcal{A}|_i$ for all $i \in \mathcal{L}$ or $\mathcal{A}$ leaks. Thus:

Theorem 5.5. *If $(\mathcal{A}, Sec)$ is parallel composable and $\mathcal{I} \models_s \mathcal{A}$ then there exists a well-typed interpretation $\mathcal{I}_\tau$ such that either $\mathcal{I}_\tau \models_s \mathcal{A}|_\ell$ for all protocol-specific labels $\ell \in \mathcal{L}$ or some prefix $\mathcal{A}'$ of $\mathcal{A}$ leaks a secret from Sec under $\mathcal{I}_\tau$.*

Proof. From the assumptions, Lemma 5.21, and Lemma 5.20 we can obtain a parallel composable $\mathcal{B}$ such that

$$\mathcal{B} \in tr^{pc}(\mathcal{A}) \text{ and } \mathcal{I} \models \mathcal{B} \qquad (*)$$

From Lemma 5.16 we can then obtain a well-typed interpretation $\mathcal{I}_\tau$ such that either

1. $\mathcal{I}_\tau \models \mathcal{B}|_\ell$ for all $\ell \in \mathcal{L}$, or
2. some prefix $\mathcal{B}'$ of $\mathcal{B}$ leaks a secret from Sec under $\mathcal{I}_\tau$.

In the former case it follows from Lemma 5.19, Lemma 5.20, and (*) that $\mathcal{I}_\tau \models_s \mathcal{A}|_\ell$ for all $\ell \in \mathcal{L}$ (note that the assumption of Lemma 5.20 follows from the fact that $\mathcal{B}$ is parallel composable and that the assumption is also preserved during projections). In the latter case we can obtain a secret $s \in Sec \setminus declassified(\mathcal{B}', \mathcal{I}_\tau)$ and a protocol-specific label $\ell_s \in \mathcal{L}$ such that $\mathcal{I}_\tau \models \mathcal{B}'|_{\ell_s}.\mathsf{send}(s)$. We need to prove that some prefix of $\mathcal{A}$ leaks the secret s and we will do so using the semantic equivalence of tr^{pc}. However, there is not necessarily a corresponding prefix of $\mathcal{A}$ with $\mathcal{B}'$ as a translation, and we need such a prefix to apply Lemma 5.20. Therefore we consider the longest prefix $\mathcal{B}''$ of $\mathcal{B}'$ that ends in a receive step (which must exist if s is not derivable from the empty intruder knowledge). For $\mathcal{B}''$ we can prove that there exists some prefix $\mathcal{A}''$ of $\mathcal{A}$ such that $\mathcal{B}'' \in tr^{pc}(\mathcal{A}'')$. We also know that $\mathcal{I}_\tau \models \mathcal{B}''|_{\ell_s}.\mathsf{send}(s)$ because $\mathcal{B}'$ and $\mathcal{B}''$ have the same intruder knowledges (also after projections). Moreover, $declassified(\mathcal{B}'', \mathcal{I}_\tau) = declassified(\mathcal{A}'', \mathcal{I}_\tau)$ and $ik(\mathcal{B}'') = ik(\mathcal{A}'')$ (also after projections). Thus we have that $\mathcal{A}''$ leaks a secret from Sec under $\mathcal{I}_\tau$ and we can therefore conclude the proof. □

5.4.3 Proving the Result for Protocols

Now that we have proven the result on the constraint level we can now prove Theorem 5.7 for stateful protocols.

The results in this section have so far only been proven in Isabelle for stateless protocols, but the pen-and-paper proofs are all for stateful protocols. In fact, the proofs for stateless and stateful protocols are very similar, and we intend to formalize it in Isabelle in the near future.

First we define the following abbreviations for arbitrary protocols $\mathcal{P}_1$ and $\mathcal{P}_2$:

1. $\mathcal{P}_1^\bullet \equiv \mathcal{P}_1 \parallel \mathcal{P}_2^\star$
2. $\mathcal{P}_2^\bullet \equiv \mathcal{P}_1^\star \parallel \mathcal{P}_2$
3. $\mathfrak{P}^\bullet \equiv \{\mathcal{A} \mid \mathcal{A}|_1 \in \mathit{traces}(\mathcal{P}_1^\bullet), \mathcal{A}|_2 \in \mathit{traces}(\mathcal{P}_2^\bullet)\}$

The main idea is now to prove the compositionality result for $\mathfrak{P}^\bullet$ (Lemma 5.24) from which the theorem follows. For that reason we first need to show that the traces of the composed protocol $\mathcal{P}_1 \parallel \mathcal{P}_2$ occur in $\mathfrak{P}^\bullet$ (Lemma 5.22) and that $(\mathcal{P}_1 \parallel \mathcal{P}_2, \mathit{Sec})$ is parallel composable iff $(\mathfrak{P}^\bullet, \mathit{Sec})$ is parallel composable (Lemma 5.23):

Lemma 5.22 $\mathit{traces}(\mathcal{P}_1 \parallel \mathcal{P}_2) \subseteq \mathfrak{P}^\bullet$

Proof. A constraint $\mathcal{A} \in \mathit{traces}(\mathcal{P}_1 \parallel \mathcal{P}_2)$ consists of an interleaving of two reachable constraints $\mathcal{A}_1 \in \mathit{traces}(\mathcal{P}_1)$ and $\mathcal{A}_2 \in \mathit{traces}(\mathcal{P}_2)$. Consider $\mathcal{A}|_1$. We need to prove that this constraint is in $\mathit{traces}(\mathcal{P}_1^\bullet)$. We have that $\mathcal{A}|_1$ consists of an interleaving of $\mathcal{A}_1|_1$ and $\mathcal{A}_2|_1$, and that $\mathcal{A}_1|_1 = \mathcal{A}_1 \in \mathit{traces}(\mathcal{P}_1)$ and $\mathcal{A}_2|_1 = \mathcal{A}_2|_\star \in \mathit{traces}(\mathcal{P}_2^\star)$. Thus $\mathcal{A}|_1 \in \mathit{traces}(\mathcal{P}_1^\bullet)$ because $\mathcal{P}_1^\bullet = \mathcal{P}_1 \cup \mathcal{P}_2^\star$. By a similar argument we can prove that $\mathcal{A}|_2 \in \mathit{traces}(\mathcal{P}_2^\bullet)$. $\square$

Lemma 5.23 *$(\mathcal{P}_1 \parallel \mathcal{P}_2, \mathit{Sec})$ is parallel composable if and only if $(\mathfrak{P}^\bullet, \mathit{Sec})$ is parallel composable.*

Proof. Note that all constraint steps that occur in $\mathit{traces}(\mathcal{P}_1 \parallel \mathcal{P}_2)$ also occur in $\mathfrak{P}^\bullet$, and vice versa. Since all but our well-formedness requirements universally quantifies over the terms and steps occurring in the protocols we have that these requirements are satisfied for $(\mathcal{P}_1 \parallel \mathcal{P}_2, \mathit{Sec})$ if and only if they are satisfied for $(\mathfrak{P}^\bullet, \mathit{Sec})$. For the well-formedness requirements note that we require all the reachable constraints plus all of the projections to be well-formed. Since $\mathfrak{P}^\bullet$ really only differs from $\mathit{traces}(\mathcal{P}_1 \parallel \mathcal{P}_2)$ by including $\star$-projections of (and interleavings of) constraints from $\mathit{traces}(\mathcal{P}_1 \parallel \mathcal{P}_2)$ we have that the well-formedness requirements for $\mathit{traces}(\mathcal{P}_1 \parallel \mathcal{P}_2)$ are satisfied if and only if they are satisfied for $\mathfrak{P}^\bullet$. $\square$

Lemma 5.24 *If $(\mathfrak{P}^\bullet, Sec)$ is parallel composable, and $\mathcal{P}_1^\bullet$ is well-typed secure in isolation (i.e., there does not exist a constraint in $traces(\mathcal{P}_1^\bullet)$ of the form $\mathcal{A}.(1\colon \mathsf{send}(\mathsf{attack}_1))$ such that either $\mathcal{A}.(1\colon \mathsf{send}(\mathsf{attack}_1))$ is well-typed satisfiable or some prefix $\mathcal{A}'$ of $\mathcal{A}|_1$ leaks a shared secret under a well-typed model), then for any attack $\mathcal{A}.(1\colon \mathsf{send}(\mathsf{attack}_1)) \in \mathfrak{P}^\bullet$ on $\mathcal{P}_1$, there exists some prefix $\mathcal{A}' \in traces(\mathcal{P}_2^\bullet)$ of $\mathcal{A}|_2$ that leaks a secret under a well-typed model.*

Proof. We first prove that any $\mathcal{A}.(1\colon \mathsf{send}(\mathsf{attack}_1)) \in \mathfrak{P}^\bullet$ is parallel composable. Then we can apply Theorem 5.5 since the constraint is satisfiable (otherwise it would not be an attack), and since $\mathcal{P}_1^\bullet$ is secure it must be the case that some prefix of $\mathcal{A}' \in traces(\mathcal{P}_2^\bullet)$ of $\mathcal{A}|_2$ leaks a secret under a well-typed model. □

From Lemma 5.22, Lemma 5.23, and Lemma 5.24 follows our main theorem:

Theorem 5.7. *If $(\mathcal{P}_1 \parallel \mathcal{P}_2, Sec)$ is parallel composable, $\mathcal{P}_1 \parallel \mathcal{P}_2^\star$ is well-typed secure in isolation, and $\mathcal{P}_1^\star \parallel \mathcal{P}_2$ does not leak a secret under any well-typed model, then all goals of $\mathcal{P}_1$ hold in $\mathcal{P}_1 \parallel \mathcal{P}_2$ (even in the untyped model).*

As a consequence of Theorem 5.7 we have the following corollary:

Corollary 5.8. *If $(\mathcal{P}_1 \parallel \mathcal{P}_2, Sec)$ is parallel composable and $\mathcal{P}_1 \parallel \mathcal{P}_2^\star$ and $\mathcal{P}_1^\star \parallel \mathcal{P}_2$ are both secure in isolation then the composition $\mathcal{P}_1 \parallel \mathcal{P}_2$ is also secure (even in the untyped model).*

Proof. Apply Theorem 5.7 twice: once to $\mathcal{P}_1^\bullet$ and once to $\mathcal{P}_2^\bullet$.

5.5 Summary and Related Work

Building on the Isabelle-formalized stateful typing result we established in this chapter a compositionality result for stateful protocols. Our composition theorem for parallel composition is the newest in a sequence of parallel composition results that are each pushing the boundaries of the class of protocols that can be composed [HT94, GT00, ACG+08, Gut09, CD09, cCC10, CDKR13, ACD15, AMMV15]. The first results simply require completely disjoint encryptions; subsequent results allowed the sharing of long-term keys, provided that wherever the common keys are used, the content messages of the different protocols are distinguished, for instance by tagging. Other aspects are which primitives are supported as well as what forms of negative conditions.

Our result lifts the common requirement that the component protocols only share a fixed set of long-term public and private constants. More interestingly, our result also allows for stateful protocols that maintain databases (such as a keyserver) and the databases may even be shared between these protocols. This includes the possibility to declassify long-term secrets, e.g., to verify that a protocol is even secure if the intruder learns all old private keys. Both databases, shared databases, and declassification are considerable generalizations over the existing results.

Like [AMMV15] our compositionality result links the parallel compositionality result with a typing result such as the result of Chapter 4, i.e., essentially requiring that all messages of different meaning have a distinguishable form. Under this requirement it is sound to restrict the intruder model to using only well-typed messages which greatly simplifies many related problems. While one may argue that such a typing result is not strictly necessary for composition, we believe it is good practice and also fits well with disjointness requirements of parallel composition. Moreover, many existing protocols already satisfy our typing requirement, since, unlike tagging schemes, this does not require a modification of a protocol as long as there is some way to distinguish messages of different meaning.

There are other types of compositionality results for sequential and vertical composition, where the protocols under composition do build upon each other, e.g., one protocol establishes a key that is then subsequently used by another protocol [ACG$^+$08, EMMS10, cCC10, CCW17, MV09, GM11]. This requires that one protocol satisfies certain properties (e.g., that the key exchange is authenticated and secret) for the other protocol to rely on. Our composition result allows for such sequential composition through shared databases: a key exchange protocol may enter keys into a shared set, and the other protocol consumes these keys. Thus our concept of sharing sets generalizes the interactions between otherwise independent protocols, and one only needs to think about the interface (e.g., only authenticated, fresh, secret keys can be entered into the database; they can only be used once). Moreover, we believe that sets are also a nice way to talk about this interaction.

Related to vertical composition is Sprenger and Basin's protocol refinement [SB18]. Protocol refinement is a method for iteratively designing protocols, starting with abstract security properties, and constructing increasingly more concrete protocols by replacing abstract concepts with concrete implementations while preserving the initial security properties. One of the abstraction levels assumes abstract secure channels over which communication happens, and the subsequent refinement implements those channels, resulting in concrete cryptographic protocols. Such a refinement has similarities to vertical composition (e.g., [GM11]) in which an abstract channel protocol establishes a secure channel over which an application protocol runs.

There are several interesting aspects of compositionality that our result does not cover, for instance, [CDKR13] discusses the requirements for composing password-based protocols, and [ACD15] investigates conditions under which privacy properties can be preserved under protocol composition.

Related to protocol compositionality with privacy properties is Gutmann's cut-blur principle [GR15] for information flow which establishes a result for safe information disclosure that is limited to within so-called blur operators. The cut principle is a method that can be used to effectively reduce verification of a larger system to a subset of the system under consideration, and it has a related compositionality principle for privacy-like properties. It is an interesting question whether one can adapt their results to our setting.

So far, compositionality results for security protocols are solely "paper-and-pencil" proofs. The proof arguments are often quite subtle, e.g., given an attack where the intruder learned a nonce from one protocol and uses it in another protocol, one has to prove that the attack does not rely on this, but would similarly work for distinct nonces. It is not uncommon that parts of such proofs are a bit sketchy with the danger of overlooking some subtle problems. For this reason, we have formalized the compositionality result—on the level of constraints—in the proof assistant Isabelle/HOL [NPW02], extending the formalization of Chapters 3 and 4, giving the extremely high correctness guarantee of machine-checked proofs. To our knowledge, this work is the first such formalization of a compositionality result in a proof assistant, with the notable exception of a study in Isabelle/HOL of compositional reasoning on concrete protocols [But12].

Finally, all the works on compositionality discussed so far are based on a black-box model of cryptography. There are several cryptographic frameworks for composition, most notably universal composability [Can01, KT11] and reactive simulatability [BPW07]. Considering the real cryptography makes compositional reasoning several orders of magnitude harder than abstract cryptography models. It is an intriguing question whether stateful protocol composition can be lifted to the full cryptographic level.

Chapter 6

Conclusion

In this thesis we established two kinds of relative soundness results for stateful security protocols, namely typing results and compositionality results. We moreover formalized them in the proof assistant Isabelle/HOL. We first formalized an existing typing result [AMMV15] in Isabelle/HOL and then built upon it to establish a typing result for stateful protocols where participants can maintain a global mutable state that span multiple sessions. Afterwards, we established a parallel compositionality result for stateful protocols, extending the compositionality paradigm to protocols that may share databases. Because of the generality of our compositionality result we are even able to show that sequential protocol composition is a special case of stateful parallel composition.

The main contributions of this work is a considerable extension of the compositionality paradigm to stateful protocols, and the formalization in Isabelle/HOL that provides us with strong correctness guarantees. In fact, we found mistakes in other works on relative soundness results during our Isabelle-formalization effort [Möd12, AMMV15]

To demonstrate the practical feasibility of our result we have shown that the TLS 1.2 handshake protocol satisfies the requirements of our typing result. Moreover, we have defined two stateful keyserver protocols and shown that they satisfy the conditions of our compositionality result. We have also illustrated how to connect other protocol formalisms with our symbolic constraints.

6.1 Future Work

There are several interesting directions for future research that are worthy of exploration. The most interesting of those can roughly be divided into two classes: strengthening the result by generalizing to larger classes of protocols and security properties, and making the results practically usable by integrating protocol verification methods.

Supporting the Full Geometric Fragment for Security Goals As explained in Subsection 5.1.2 our constraint language is not quite expressive enough to support all security goals expressible in the geometric fragment. The reason being is that we currently cannot express negative checks of the form $\forall \bar{x}.\ t_1 \mathrel{\dot{\notin}} s_1 \vee \cdots \vee t_n \mathrel{\dot{\notin}} s_n$. The most promising solution seems to be to simply extend our constraint language to support such constraints. This will require updates throughout our Isabelle formalization, but besides time it is unlikely that there are any major obstacles in the way.

Algebraic Properties Our term model is currently based on a free algebra in which terms are equal if they are syntactically equal. An interesting question for future research is thus if it is possible to support algebraic properties such as the properties of exponentiation needed to model Diffie-Hellman-based protocols. As already explain in Subsection 2.3.1 the ProVerif tool supports algebraic properties to some extent by a transformation into Horn clauses (which are in the free algebra). A similar transformation may also be possible in our setting. Another possibility would be to work directly with quotient algebras in Isabelle as in the work of [LS17]. Some of the challenges we face, however, is how to integrate unification modulo equational theories (since computing most-general unifiers is such an integral part of our constraint-based approach) and how to extend our typed model [Möd11].

Equivalence Properties As of now our result has focused entirely on reachability properties which includes confidentiality and authentication goals. There are other interesting properties like privacy-type goals using equivalence properties, and results in this direction have been established [CCD14]. Supporting such properties may be challenging as they are inherently different from reachability properties and the constraint-based approach that is used in this work. Another possibility could be to integrate our approach with α-β privacy [MGV13]. A question for future research is thus if statefulness and equivalence proofs can be combined.

Vertical Composition We demonstrated in Subsection 5.3.6 how sequential composition can be reduced to a stateful parallel composition problem. The idea is to use dedicated shared databases as a means to hand over information from one protocol to another. It is possible that vertical composition can also be recast as stateful parallel composition using a similar approach. For instance, consider a channel protocol that establishes a secure channel and an abstract application protocol which depends on a secure channel. The messages from the application protocol that need to be securely transmitted can be handed over to the channel protocol using shared sets. The challenge is then to find reasonable conditions for the channel and application protocols so that the reduction to the stateful parallel compositionality problem satisfies our composability conditions.

Verifier Integration The work presented in this thesis has focused on establishing compositionality results and formalizing them in Isabelle. As of now, to apply our results, one would need to manually model and prove protocols correct in Isabelle/HOL. Manually verifying protocols correct in Isabelle/HOL can be cumbersome and extremely time-consuming. Integrating protocol verification with our results is thus crucial to make them practically available. One approach would be to use existing Isabelle-frameworks for modeling and proving protocols manually, e.g., the work of Paulson and Bella [Pau98, Pau99, BMP06, Bel07], and link those to our typing and compositionality results. A more promising approach is to have automated tools generate proofs that Isabelle can check, e.g., by building on existing work in this direction [BM09, MCB13].

Bibliography

[AAA+12] Alessandro Armando, Wihem Arsac, Tigran Avanesov, Michele Barletta, Alberto Calvi, Alessandro Cappai, Roberto Carbone, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Gabriel Erzse, Simone Frau, Marius Minea, Sebastian Mödersheim, David von Oheimb, Giancarlo Pellegrino, Serena Elisa Ponta, Marco Rocchetto, Michaël Rusinowitch, Mohammad Torabi Dashti, Mathieu Turuani, and Luca Viganò. The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In *TACAS 2012*, pages 267–282, 2012.

[AC08] Alessandro Armando and Luca Compagna. SAT-based model-checking for security protocols analysis. *Int. J. Inf. Sec.*, 7(1):3–32, 2008.

[ACD15] Myrto Arapinis, Vincent Cheval, and Stéphanie Delaune. Composing security protocols: From confidentiality to privacy. In Riccardo Focardi and Andrew Myers, editors, *Principles of Security and Trust*, pages 324–343, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[ACG+08] Suzana Andova, Cas J. F. Cremers, Kristian Gjøsteen, Sjouke Mauw, Stig Fr. Mjølsnes, and Saša Radomirović. A framework for compositional verification of security protocols. *Inf. Comput.*, 206(2-4):425–459, 2008.

[AD14] Myrto Arapinis and Marie Duflot. Bounding messages for free in security protocols - extension to various security properties. *Inf. Comput.*, 239:182–215, 2014.

[AMMV15] Omar Almousa, Sebastian Mödersheim, Paolo Modesti, and Luca Viganò. Typing and compositionality for security protocols: A generalization to the geometric fragment. In *ESORICS 2015*, pages 209–229, 2015.

[ARR11] M. Arapinis, E. Ritter, and M.D. Ryan. StatVerif: Verification of stateful processes. In *CSF 2011*, pages 33–47. IEEE, 2011.

[BCFS10] Matteo Bortolozzo, Matteo Centenaro, Riccardo Focardi, and Graham Steel. Attacking and fixing PKCS#11 security tokens. In *CCS 2010*, pages 260–269, 2010.

[Bel07] Giampaolo Bella. *Formal Correctness of Security Protocols*. Information Security and Cryptography. Springer, 2007.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *CSFW 2001*, pages 82–96, 2001.

[BM09] Achim D. Brucker and Sebastian Mödersheim. Integrating automated and interactive protocol verification. In *FAST 2009*, 2009.

[BMNN15] Alessandro Bruni, Sebastian Mödersheim, Flemming Nielson, and Hanne Riis Nielson. Set-Pi: Set membership p-calculus. In *CSF 2015*, pages 185–198, 2015.

[BMP06] Giampaolo Bella, Fabio Massacci, and Lawrence C. Paulson. Verifying the SET purchase protocols. *J. Autom. Reasoning*, 36(1-2):5–37, 2006.

[BMV05] David A. Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *Int. J. Inf. Sec.*, 4(3):181–208, 2005.

[BP05] Bruno Blanchet and Andreas Podelski. Verification of cryptographic protocols: tagging enforces termination. *Theor. Comput. Sci.*, 333(1-2):67–90, 2005.

[BPW07] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Inf. Comput.*, 205(12):1685–1720, 2007.

[But12] Denis Frédéric Butin. *Inductive analysis of security protocols in Isabelle/HOL with applications to electronic voting*. PhD thesis, Dublin City University, November 2012.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

[cCC10] Ştefan Ciobâcă and Véronique Cortier. Protocol composition for arbitrary primitives. In *CSF*, pages 322–336. IEEE, 2010.

[CCD14] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. Typing messages for free in security protocols: The case of equivalence properties. In *CONCUR 2014*, pages 372–386, 2014.

[CCW17] Vincent Cheval, Véronique Cortier, and Bogdan Warinschi. Secure composition of PKIs with public key protocols. In Computer Security Foundations Symposium 2017 [Com17], pages 144–158.

[CD09] Véronique Cortier and Stéphanie Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, 2009.

[CDKR13] Céline Chevalier, Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Composition of password-based protocols. *Formal Methods in System Design*, 43(3):369–413, Dec 2013.

[CDL05] Iliano Cervesato, Nancy A. Durgin, and Patrick Lincoln. A comparison between strand spaces and multiset rewriting for security protocol analysis. *Journal of Computer Security*, 13(2):265–316, 2005.

[CM12] Cas Cremers and Sjouke Mauw. *Operational Semantics and Verification of Security Protocols*. Information Security and Cryptography. Springer, 2012.

[Com17] *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*. IEEE Computer Society, 2017.

[DR08] T. Dierks and E. Rescorla. RFC 5246: The Transport Layer Security (TLS) Protocol, Version 1.2, 2008. Available: `http://tools.ietf.org/rfc/rfc5246.txt`.

[EMM07] Santiago Escobar, Catherine A. Meadows, and José Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures*, volume 5705 of *Lecture Notes in Computer Science*, pages 1–50. Springer, 2007.

[EMMS10] Santiago Escobar, Catherine A. Meadows, José Meseguer, and Sonia Santiago. Sequential protocol composition in Maude-NPA. In *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings*, pages 303–318, 2010.

[FS09] Sibylle B. Fröschle and Graham Steel. Analysing PKCS#11 key management APIs with unbounded fresh data. In *ARSPA-WITS 2009*, pages 92–106, 2009.

[GM11] Thomas Groß and Sebastian Mödersheim. Vertical protocol composition. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*, pages 235–250. IEEE Computer Society, 2011.

[GN13] Georges Gonthier and Michael Norrish, editors. *Certified Programs and Proofs - Third International Conference, CPP 2013, Melbourne, VIC, Australia, December 11-13, 2013, Proceedings*, volume 8307 of *Lecture Notes in Computer Science*. Springer, 2013.

[Gou08] Jean Goubault-Larrecq. Towards producing formally checkable security proofs, automatically. In *Computer Security Foundations Symposium*, 2008.

[GR15] Joshua D. Guttman and Paul D. Rowe. A cut principle for information flow. In Cédric Fournet, Michael W. Hicks, and Luca Viganò, editors, *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 107–121. IEEE Computer Society, 2015.

[GT00] Joshua D. Guttman and F. Javier Thayer. Protocol independence through disjoint encryption. In *CSFW*, pages 24–34. IEEE, 2000.

[Gut09] Joshua D. Guttman. Cryptographic Protocol Composition via the Authentication Tests. In *FOSSACS'09*, pages 303–317. Springer, 2009.

[Gut12] Joshua D. Guttman. State and progress in strand spaces: Proving fair exchange. *J. Autom. Reasoning*, 48(2):159–195, 2012.

[Gut14] Joshua D. Guttman. Establishing and preserving protocol security goals. *Journal of Computer Security*, 22(2):203–267, 2014.

[HK13] Brian Huffman and Ondrej Kuncar. Lifting and transfer: A modular design for quotients in Isabelle/HOL. In Gonthier and Norrish [GN13], pages 131–146.

[HLS03] James Heather, Gavin Lowe, and Steve Schneider. How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217–244, 2003.

[HM17] Andreas Viktor Hess and Sebastian Mödersheim. Formalizing and Proving a Typing Result for Security Protocols in Isabelle/HOL. In Computer Security Foundations Symposium 2017 [Com17], pages 451–463.

[HM18] Andreas Viktor Hess and Sebastian Mödersheim. A Typing Result for Stateful Protocols. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 374–388. IEEE Computer Society, 2018.

[HMB18] Andreas Viktor Hess, Sebastian Alexander Mödersheim, and Achim D. Brucker. Stateful Protocol Composition. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, volume 11098 of *Lecture Notes in Computer Science*, pages 427–446. Springer, 2018.

[HT94] N. Heintze and J. D. Tygart. A model for secure protocols and their compositions. In *Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 2–13, May 1994.

[KK16] Steve Kremer and Robert Künnemann. Automated analysis of security protocols with global state. *Journal of Computer Security*, 24(5):583–616, 2016.

[KT09] Ralf Küsters and Tomasz Truderung. Using ProVerif to Analyze Protocols with Diffie-Hellman Exponentiation. In *CSF*, pages 157–171. IEEE, 2009.

[KT11] Ralf Küsters and Max Tuengerthal. Composition theorems without pre-established session identifiers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 41–50, New York, NY, USA, 2011. ACM.

[LS17] Joseph Lallemand and Christoph Sprenger. Refining authenticated key agreement with strong adversaries. *Archive of Formal Proofs*, 2017, 2017.

[MB16] Sebastian Mödersheim and Alessandro Bruni. AIF-ω: Set-based protocol abstraction with countable families. In *POST 2016*, 2016.

[MCB13] Simon Meier, Cas Cremers, and David A. Basin. Efficient construction of machine-checked symbolic protocol security proofs. *Journal of Computer Security*, 21(1):41–87, 2013.

[MGV13] Sebastian Mödersheim, Thomas Groß, and Luca Viganò. Defining privacy is supposed to be easy. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013.*

Proceedings, volume 8312 of *Lecture Notes in Computer Science*, pages 619–635. Springer, 2013.

[MM11] Sebastian Mödersheim and Paolo Modesti. Verifying SeVeCom using set-based abstraction. In *IWCMC 2011*, pages 1164–1169, 2011.

[Möd10] Sebastian Mödersheim. Abstraction by set-membership: verifying security protocols and web services with databases. In *CCS 2010*, 2010.

[Möd11] Sebastian Mödersheim. Diffie-Hellman without Difficulty. In *FAST 2011*, pages 214–229, 2011.

[Möd12] Sebastian Mödersheim. Deciding Security for a Fragment of ASLan. In *ESORICS*, pages 127–144. Springer, 2012.

[MS01] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS 2001*, 2001.

[MSCB13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *CAV 2013*, 2013.

[MV09] S. Mödersheim and L. Viganò. Secure pseudonymous channels. *ESORICS 2009*, pages 337–354, 2009.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.

[Pau99] Lawrence C. Paulson. Inductive analysis of the Internet protocol TLS. *ACM Trans. Inf. Syst. Secur.*, 2(3):332–351, 1999.

[RT03] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *Theor. Comput. Sci.*, 299, 2003.

[SB18] Christoph Sprenger and David A. Basin. Refining security protocols. *Journal of Computer Security*, 26(1):71–120, 2018.

[SEV09] SEVECOM project. Deliverable 2.1-App. A Baseline Security Specification, 2009. https://www.sevecom.eu/Deliverables/Sevecom_Deliverable_D2.1-App.A_v1.2.pdf.

[SP13] Andreas Schropp and Andrei Popescu. Nonfree Datatypes in Isabelle/HOL - Animating a Many-Sorted Metatheory. In Gonthier and Norrish [GN13], pages 114–130.

[ST18] Christian Sternagel and René Thiemann. First-order terms. *Archive of Formal Proofs*, February 2018. `http://isa-afp.org/entries/First_Order_Terms.html`, Formal proof development.

[THG99] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(1):191–230, 1999.

[TS09] René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *TPHOLs 2009*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009.