Karlsruhe Institute of Technology

Symbolic Analysis of Cryptographic Protocols

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Florian Böhl

aus Bielefeld

Tag der mündlichen Prüfung:	03.06.2014
Erster Gutachter:	Jun.-Prof. Dr. Dennis Hofheinz
Zweiter Gutachter:	Prof. Dr. Hubert Comon-Lundh

Zusammenfassung

Kryptographische Protokolle sind allgegenwärtig und täglich verlassen wir uns auf ihre Sicherheitseigenschaften. Oft sichern sie die Integrität und Vertraulichkeit von Kommunikation zu. Beispielsweise sollen sie Dritte davon abhalten, unsere Telefongespräche zu belauschen (GSM/DECT) oder verhindern, dass jemand Banktransaktionen fälscht, die wir bequem von zu Hause aus durchführen (HTTPS). Wir erwarten von ihnen Authentifikation beim Öffnen von Türen, Bezahlvorgängen und digitalen Unterschriften von Dokumenten (Funk-Autoschlüssel, KITCard, EC-Karte, neuer Personalausweis). Ihre Bedeutung wird weiter zunehmen, beispielsweise im Rahmen der Automation von Gebäuden oder bei intelligenten Stromnetzen.

Gemessen an der Bedeutung kryptographischer Protokolle wird ihre Sicherheit vor der Einführung oft nur unzureichend untersucht. Beispielsweise sind mittlerweile in fast allen der oben genannten Anwendungen Fehler auf Protokollebene bekannt, die die erwarteten Sicherheitseigenschaften in den schlimmsten Fällen vollständig unterminieren (so z.B. bei DECT). Solche Fehler sind nach der Verbreitung eines Protokolls oft nur mit hohem Aufwand zu korrigieren.

Warum kommt es dennoch selten zu einer rigorose Sicherheitsanalyse (ggf. mit Sicherheitsbeweis) vor dem Einsatz eines Protokolls? Eine Ursache hierfür ist sicherlich, dass ein Sicherheitsbeweis selbst für relativ kleine kryptographische Protokolle schon sehr aufwändig und mühsam ist; für jeden Zwischenzustand des Protokolls muss jede mögliche Angreifer-Aktion berücksichtigt werden. Abhilfe können hier maschinengestützte Beweistechniken schaffen.

In meine Dissertation beschäftige ich mich mit dem Rahmen für eine solche maschinengestützte Analyse. Das Fernziel über den Horizont der Arbeit hinaus ist, dass die Sicherheitseigenschaften eines Protokolls möglichst vollautomatisch vor seinem Einsatz überprüft werden können.

Abstraktion von kryptographischen Details.

Um die Komplexität der Analyse zu reduzieren, wird zunächst von kryptographischen Details abstrahiert (das entstehende abstrakte Modell heißt auch „symbolisches Modell“, woraus sich der Titel der Arbeit ableitet). Während in der tatsächlichen Implementierung ein konkretes kryptographisches Verfahren eingesetzt werden muss, z.B. AES-CBC mit MACs, wird die Analyse anhand der abstrakten Eigenschaften des kryptographischen Bausteins durchgeführt. Für Verschlüsselung kann man beispielsweise mit den folgenden zwei Ableitungsregeln arbeiten:

$$\frac{m \quad k}{enc(k,m)}, \qquad \frac{enc(k,m) \quad k}{m}$$

Diese entsprechen der Intuition, dass sich nur mit Kenntnis des Schlüssels Chiffrate erzeugen und entschlüsseln lassen. Die in der Realität nur mit vernachlässigbarer

Wahrscheinlichkeit auftretende Möglichkeit, dass ein Angreifer den Schlüssel korrekt rät, wird beispielsweise auf dieser Ebene ignoriert, um die Analyse zu vereinfachen. Später findet sich diese Abstraktion in Anforderungen an die Implementierung wieder (siehe „Modellgetreue Implementierung“).

Modulare Analyse von Protokollen.

Große Protokolle, wie z.B. TLS, sind selbst auf der gerade erläuterten abstrakten Ebene schwer „am Stück“zu untersuchen. Daher beschäftige ich mich in meiner Dissertation auch damit, wie man Beweise für Teile des Protokolls separat führen und zu einem Sicherheitsbeweis für das gesamte Protokoll zusammenfügen kann. Der zu diesem Zweck eingeführte Mechanismus erlaubt interessanter- und günstigerweise auch die Definition von komplexen Sicherheitseigenschaften.

Modellgetreue Implementierung.

Nach dem Nachweis von Sicherheitseigenschaften im symbolischen Modell bleibt die Frage offen, welche kryptographischen Anforderungen hinreichend für eine Implementierung sind, damit sie tatsächlich die abstrakt nachgewiesenen Eigenschaften hat. Eine Aussage hierüber machen sogenannte „Computational Soundness Theoreme“. Ein solches Theorem ist Teil meiner Arbeit. Es deckt die symbolische Verwendung der wichtigsten kryptographischen Bausteine ab und ist modular erweiterbar.

Acknowledgements

First of all, I would like to thank my adviser Dennis Hofheinz. He already supervised my Diploma thesis, encouraged me to start a PhD and therewith laid the foundation for this work. Throughout the years we spent together he always took his time to supply me with technical or general wisdom and never got tired of enlightening me when I was stuck. He was the only adviser I ever had and is thus without competitors; however, that said, I find it very hard to imagine a better one.

I thank my co-adviser Huber Comon-Lundh for his interest in my work and for his willingness to take his time to read it thoroughly and write a review. Additionally, I would like to thank him for encouraging me in a time where encouragement was definitely needed.

I am very grateful to my co-authors Dominique Unruh, Véronique Cortier and Bogdan Warinschi. The results of the fruitful work with them are now the core of this thesis. It was always a pleasure, to work, think and discuss the riddles of theoretical cryptography with them; as it was with Daniel Kraschewski, Jessica Koch, Christoph Striecks, Tibor Jager, Gareth Davies, Simon Greiner, Sarah Grebing and Bernhard Beckert who worked with me on other papers. All of them taught me a lot and I am thankful for the many different ways in which they inspired me.

This does not only hold for my co-authors but for all of my current and former colleagues in the KIT crypto group (IKS/ITI crypto) in general. They always provided me with a wonderful and warm working atmosphere and broadened my horizon in endless interesting and sometimes far out discussions. I truly find it sad to leave.

I would like to thank the Bristol crypto group for six marvelous months full of crypto, pubs and music. Here, special thanks also go to Jörn Müller-Quade for supporting me with finding a scholarship and to the DAAD for making the wonderful and enriching time in Bristol possible for me.

I thank the Ministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg for supporting the project MoSeS which provided most of my funding.

Finally, and therefore prominently, I would to thank my parents who have always supported me, my patient and loving spouse Nikola Wachter who had to suffer the most under paper deadlines and the finishing of this thesis and my friends Barbara Lödermann, Simon Friedberger and Julia Hesse who helped proofreading this thesis.

To everyone who deserved my thanks but was forgotten above: Thank you!

Contents

1 Introduction 1

2 Symbolic Universal Composability 13

2.1 Review of the applied pi calculus . . . 14

2.1.1 Syntactic sugar . . . 17

2.1.2 Additional concepts used in this work . . . 17

2.2 Useful properties of the pi calculus . . . 21

2.2.1 Relating events and observational equivalence . . . 30

2.2.2 Unpredictability of nonces . . . 37

2.3 Symbolic UC . . . 38

2.4 Composition . . . 42

2.5 Property preservation . . . 63

2.6 Relation to Delaune-Kremer-Pereira . . . 65

2.7 Example: Secure channels . . . 68

2.7.1 Key exchange using NSL . . . 69

2.7.2 Secure channel from key exchange. . . . 71

2.7.3 Generating many keys from one . . . 77

2.8 Virtual primitives . . . 81

2.8.1 Realizing commitments . . . 83

2.8.1.1 A note on adaptive corruption . . . 89

2.8.2 Removing the virtual primitives . . . 90

2.8.3 On removing the CRS . . . 94

2.9 Limits for composition and property preservation . . . 97

3 Composable Computational Soundness 103

3.1 Preliminaries . . . 105

3.2 The symbolic model . . . 105

3.2.1 Reconciling the notions for symbolic models . . . 106

3.3 Implementation . . . 108

3.3.1 Interpretations . . . 108

3.3.2 Generating function . . . 108

3.3.3 Parsing function . . . 109

3.3.4 Good implementation . . . 110

3.4 Transparent functions . . . 113

3.5 Composition . . . 113

3.6 Deduction soundness . . . 120

3.7 Composition theorems . . . 123

3.7.1 Public datastructures . . . 123

3.7.2 Public key encryption . . . 123
3.7.2.1 Computational preliminaries . . . 124
3.7.2.2 Symbolic model . . . 124
3.7.2.3 Implementation . . . 126
3.7.2.4 PKE composability . . . 127
3.7.3 Signatures . . . 134
3.7.3.1 Computational preliminaries . . . 134
3.7.3.2 Symbolic model . . . 135
3.7.3.3 Implementation . . . 135
3.7.3.4 Signature composability . . . 137
3.7.4 Secret key encryption . . . 143
3.7.4.1 Computational preliminaries . . . 143
3.7.4.2 Symbolic model . . . 144
3.7.4.3 Implementation . . . 145
3.7.4.4 SKE composability . . . 146
3.7.5 MACs . . . 149
3.7.5.1 Computational preliminaries . . . 149
3.7.5.2 Symbolic model . . . 150
3.7.5.3 Implementation . . . 150
3.7.5.4 MAC composability . . . 151
3.7.6 Hash functions . . . 152
3.7.6.1 Symbolic model . . . 152
3.7.6.2 Implementation . . . 152
3.7.6.3 Hash composability . . . 153
3.8 Forgetfulness . . . 154
3.8.1 Preliminaries . . . 155
3.8.2 Forgetful symbolic models and implementations . . . 156
3.8.3 Sending keys around . . . 159

4 Outlook 163

Symbol Index 171

Index 175

1. Introduction

Cryptography is Ubiquitous.

Nowadays, more and more aspects of everyday life depend on the security of cryptographic protocols. We rely on them to assert the confidentiality of calls using our mobile phones (GSM [77]) or our cordless phones at home (DECT [75]). They protect our online banking transactions and are necessary to prevent identity theft (TLS/HTTPS [78]). We expect them to authenticate us at borders whilst still guaranteeing a good level of privacy against eavesdroppers (e-Passports). The importance of cryptographic protocols is growing. With Bitcoin we have the first popular e-currency, people are automating their homes to a larger and larger extent, and smart grids are going to supply us with energy in the future. Where will we turn to if not cryptography to solve the underlying security and privacy problems? From being used mainly for military purposes just about half a century ago, cryptography has evolved to be one of the most important tools for protecting everything from individual rights to the society at large in the digital age.

A State of Uncertainty.

Obviously, the widespread use of cryptographic protocols and the level of trust necessary for their use calls for strong security guarantees. Disconcertingly, many of the aforementioned protocols were standardized and are now widely used despite a lack of such guarantees. For most of them, serious flaws in their design are known by now. The most striking example of this is the TLS protocol which sits at the heart of most trusted communications on the Internet and still suffers from important security issues.

From this state of uncertainty, two major problems emerge. For one, security problems may obviously constitute a direct threat. Additionally, the lack of trust itself is a problem. Recently, Bitcoin courses plummeted due to alleged weaknesses in the protocol. After it became clear that the problem was a local implementation issue, markets recovered again [61]. The incident shows that the trust in cryptographic protocols does not go very far. As long as the vice president of the United States goes through surgery to deactivate the WIFI of his heart pacemaker for security reasons [57], we know that there is still work to do.

Security Problems are Hard to Find.

One key reason for the current situation is that security guarantees differ fundamentally from functional guarantees. Consider a system for which we would like to be certain that it has some functional property. There are several ways to tackle the problem. The most important and common one these days is testing. Furthermore, users of the system can usually assess whether it provides the promised functionality. This does not hold for security guarantees. E.g., secret information might leak without affecting the observable behavior of a system.

Errors are Hard to Fix.

Another important property of cryptographic protocols is that errors are often hard to fix. Here, the hardness rather stems from the distributed nature of protocols than from cryptographic aspects. For a widely-used protocol, we usually have many implementations running on many different devices under the control of many different parties. Obviously, deploying a fix for a flawed protocol is a non-trivial task in this setting. As a consequence, devices are usually not updated simultaneously but over time. Until the update process is complete, devices often have to use the oldest version of the protocol supported by communication partners. In some cases where the protocol involves a continuous process, e.g., Bitcoin, a flaw might even completely break the current state (e.g., all Bitcoins would be worthless). The only fix would then be to re-start from scratch.

A Need for Strong Security Guarantees.

To quickly recapitulate the setting most cryptographic protocols live in: Security problems are hard to find, hard to fix, consequences of errors might be dramatic, and even a lack of trust can lead to major problems. All of this calls for strong and rigorous security guarantees. There are multiple reasons why we do not already have these guarantees which we are going to shed some light on next.

The Dark Ages of Industrial Cryptography.

Before we judge the state of the art, we should remember that the widespread use of cryptographic protocols is a quite recent development. When the Domain Name System (DNS) was standardized in 1987, less than 30 years ago, security was not even considered to be an issue. Only as the Internet grew and spread, it became obvious that there were also parties with malicious intentions. This time, the mid 90s, was when a lot of the protocols we use today were devised (SSL in '94, SSH in '95). At that time decisions were made on the basis of what experts thought "looks good". This period is now sometimes referred to as "The Dark Ages of Industrial Cryptography" [60].

Unfortunately, protocols from that time do not only have a lot of flaws. Due to the design decisions made, they turn out to be inherently hard to prove correct. This is one of the reasons why we still do not have a proof of security for SSL/TLS – not even for the updated versions of it [60].

What can We Do Already?

Despite the current situation for cryptographic protocols there is hope for the future. One reason for this are developments in the area of cryptographic building blocks like encryption or hash functions. While the same secure-by-expert-opinion design has been prevalent for quite a while for those building blocks, the current standards (AES for symmetric encryption and SHA-3 for hash functions) were selected in an open competition and, among many other things, a proof of security was required. Even for older building blocks, e.g., RSA and ElGamal, we have provably secure variants these days. And indeed, until today, those provably secure schemes have withstood the test of time. This provides cause for hope that proofs of security are also going to be common practice for new cryptographic protocols in the future.

What Do We Need?

Obviously, the need for provably secure protocols can only be satisfied if we can provide the means to actually conduct appropriate proofs – say within the scope of a standardization process.

Security proofs for cryptographic protocols, due to the many different states a protocol can be in, are typically not complex but large. In contrast to the aforementioned security proofs for building blocks, conducting these proofs by hand would be too tedious and error-prone. Therefore, machine-assisted proofs are the only remedy.

Machine-assisted proofs in general have the stigma of being very time-consuming. However, in comparison to automated proofs for software, proofs for cryptographic protocols only need to be conducted once. The specification of a protocol is much less volatile than software. In that way, having provably secure protocols is a less lofty goal than having provably correct software in general, or, e.g., a provably correct operating system kernel [13].

Still, to make the security proofs for protocols as efficient as possible, we should not rely on machine assistance alone. To deal with large systems and protocols, we need modular proofs. That is, we need methods to break down a security proof into small easy-to-handle parts. Furthermore, we require ways to re-use these parts (and their proofs) for different systems.

While aiming for modular machine-assisted proofs, we still want security against the most powerful adversary possible.

Rigorous Proofs of Security for Cryptographic Protocols.

Finally, we are going to answer the question that remained open until now: How can we actually prove a cryptographic protocol secure? To that end, we will introduce two models that are potentially suitable to conduct such proofs next.

One is the *computational model* which is closely related to complexity theory. In this model we derive security guarantees from the fact that certain problems are believed to be computationally hard to solve (like factoring numbers which are composites of large primes).

An alternative model, the so called *symbolic model*, abstracts away from concrete computations. Here, a term algebra and deduction rules define the capabilities of an adversary. In this model we can prove a security property by showing, for example, that the adversary cannot learn a message that is supposed to be secret.

Before we dive into the details of the two models we would like to emphasize that both models are abstractions of reality and cannot give us security guarantees against all conceivable attacks. As soon as we are talking about a concrete implementation of a protocol on actual hardware, new problems can emerge – timing or tempest attacks would be two prominent examples. Models to capture these kinds of attacks are not covered here. In general, we feel that security of cryptographic protocols against a computational adversary as introduced in this work is definitely a necessary but not a sufficient criterion for a protocol that is ready to be deployed.

The Computational Model.

The foundation of the computational model of cryptography is the – in computer science widely accepted – conjecture that some problems cannot be solved in polynomial time (like the aforementioned factoring of large numbers). Theorems in this model basically are statements of the kind "X is secure if the problem Y is hard to solve". Proofs are actually complexity theoretic reductions. That is, we show that an adversary that breaks X could be used to solve Y. As long as Y is really hard, no successful adversary that runs in polynomial time can exist.

Security Properties in the Computational Model.

What does "X is secure" actually mean? There are two common approaches to model security properties in the computational model.

- **Experiments:** The idea of experiments goes back to Goldwasser and Micali [59]. Here, the adversary plays a game with a so called experiment. The concrete goal and the form of the game formalize a security property. E.g., one standard notion of security for public-key encryption schemes, IND-CPA security, can be captured by the following game: The experiment generates a keypair (pk, sk) and hands the public key pk to the adversary. The adversary replies with two message m_0 and m_1 (of equal length). Now the experiment picks a random bit $b \in \{0, 1\}$, computes the ciphertext c of m_b under pk and returns c to the adversary. Finally, the adversary has to guess the value of b and wins if it guessed correctly, i.e., it has to guess which of the messages is contained in c. We say that a public-key encryption scheme is IND-CPA secure if no efficient adversary exists that is significantly more successful than one that uniformly guesses a value.

- **Simulation-based notions:** The idea behind simulation-based notions of security, brought forward by Goldwasser and Micali [59] as well, is to stipulate the indistinguishability of two settings, often dubbed the *real* and the *ideal* setting. Intuitively, we have an adversary that interacts with a real implementation $\mathcal{R}$ of a system on the real side. On the ideal side we have an adversary that interacts with an idealization $\mathcal{F}$ of $\mathcal{R}$. The idea behind the idealization $\mathcal{F}$ is that it just describes the functionality of $\mathcal{R}$. E.g., while $\mathcal{R}$ might use an encryption scheme to distribute information among parties, in $\mathcal{F}$ the information is magically transferred to every party giving the adversary no attack surface. We now stipulate that for every adversary on the real side there is an adversary on the ideal side such that both sides are indistinguishable. This captures the intuition that for every attack on $\mathcal{R}$ there is an attack on $\mathcal{F}$. Or, in other words, the security of the system $\mathcal{R}$ is defined by the security of the ideal system $\mathcal{F}$.

The computational model features a strong adversary. It can capture all attacks based on the input/output behavior of honest parties[1] that can be carried out in polynomial time. However, it is not very suitable for automation although some approaches exist [18, 70]. Hence, it will not serve our needs when it comes to proving cryptographic protocols secure on a large scale.

From Cryptographic Building Blocks to Cryptographic Protocols.

Before we proceed and introduce the symbolic model, we quickly explain the relation between cryptographic building blocks (commonly also called *cryptographic*

[1] That is, the adversary only gets the data without any additional information like power consumption, time consumption, or information about any internal states.

primitives) and cryptographic protocols. As the name suggests, cryptographic primitives are the foundation of cryptographic protocols. Examples of standard primitives are symmetric encryption, public-key encryption, digital signatures, or cryptographic hash functions. For each of these primitives there are several notions of security (either experiments or simulation-based notions) in the computational model. We also have numerous concrete constructions for each cryptographic primitive that meet the respective notions of security.

Even though we have secure building blocks, there are many ways to use them incorrectly in a protocol. This is where the real work begins if we want to prove a protocol secure.

The Symbolic Model.

In contrast to the computational model, where both cryptographic protocols and building blocks can be analyzed, the symbolic model, introduced by Dolev and Yao [56], focuses on the analysis of protocols. The behavior of parties (participants in a protocol) is usually modeled using a process calculus, like the applied pi calculus. Messages exchanged between the parties are terms built from function symbols which represent the cryptographic primitives used. To define the semantics of function symbols, we use deduction rules or rewrite rules. E.g., we could have a function symbol *enc* of arity two together with the deduction rules

$$\frac{m \quad k}{enc(k,m)}, \qquad\qquad \frac{enc(k,m) \quad k}{m}$$

to capture the semantics of symmetric encryption. That is, knowledge of a key k and a message m is required to derive a ciphertext $enc(k,m)$. Conversely, from a ciphertext $enc(k,m)$ and key k, the message m can be retrieved. Note that the only operations possible are those specified by the deduction rules. In particular, a key cannot be guessed, and no algebraic properties of concrete cryptographic instantiations can be used. The deduction rules are an idealization of the properties we expect from a cryptographic primitive.

Security Properties in the Symbolic Model.

How can we capture security properties in the symbolic model? As mentioned above, the model focuses on the analysis of protocols. One popular way to model the adversary in the symbolic world is to give it full control over public networks. I.e., it can receive, interrupt, and send messages. Every message it receives is added to the *adversarial knowledge*. When sending messages, the adversary is restricted to messages that are deducible from the adversarial knowledge. This type of adversary is commonly referred to as *Dolev-Yao adversary* – named after the authors of the seminal paper which laid the foundation of the symbolic model [56]. There are various ways to formalize security properties in the symbolic model. We quickly cover the two that are most important for this work.

- **Querying the adversarial knowledge:** We can usually analyze security properties of a protocol by querying the adversarial knowledge. That is, we prove that a certain message is never deducible from the knowledge of a Dolev-Yao adversary that can interact with (multiple runs of) a protocol. I.e., intuitively we proof that the adversary cannot learn a certain piece of information.

- **Indistinguishability-based security:** Analogously to simulation-based security in the computational setting we can stipulate that two settings have to be indistinguishable. However, we do not consider a real and an ideal side here, but rather two versions of the same protocol with different setups. For example considering EC-cards and e-passports we might want to require a property that is called *unlinkability* [6, 4]: Say that $\pi(A)$ denotes a session of a protocol initiated by party A, by proving that $\pi(A) \mid \pi(A)$ (two sessions of the protocol initiated by A in parallel) is indistinguishable from $\pi(A) \mid \pi(B)$, we can not only guarantee that a session cannot be linked to a concrete user but also that a user cannot be tracked (not even anonymously).

Mind the Gap.

One striking advantage of the symbolic model is that proofs are easier to automate than in the computational model. However, a caveat is that we only get security guarantees with respect to abstractions of our cryptographic primitives. That is, the operations of the adversary are restricted to the deduction rules we specified. How can we be sure that there are no computational attacks that are not captured by the deduction rules and would thus be overlooked? Or, in other words: Is there a way to back up a symbolic analysis with guarantees in the computational model? Fortunately, the answer is yes.

Reconciling the Computational and the Symbolic Model.

In their seminal work, Abadi and Rogaway [2] found a way to reconcile the computational and the symbolic model. A theorem that makes a statement about the relation of these models is called a *computational soundness result.* As explained above, the symbolic model only uses cryptographic primitives in an abstract way. The idea behind computational soundness results is to find requirements for the implementation of these primitives such that guarantees obtained with a symbolic analysis hold in the computational model. Intuitively, a soundness result captures the idea that an adversary in the computational model cannot compute something, that an adversary in the symbolic model cannot deduce. Note that such a soundness theorem has to be proven only once for a given abstraction of cryptographic primitives. It then applies to each protocol analyzed with respect to that abstraction.

Open Problems.

So far, we described the state of the art: We have two different models to capture and prove security properties. The computational model provides security guarantees against stronger adversaries while the symbolic model is more suitable for machine-assisted proofs. Computational soundness results allow us to conduct proofs in the latter and still get security guarantees in the former. So what remains to be done?

A first thing that comes to mind with respect to our needs mentioned above is that we are aiming for modular proofs. Since we additionally have to rely on tool support, we need modularity in the symbolic model. While first steps in this direction have been made by [54] and [23], we still lack a fully fledged framework for modularization. This is what we are going to discuss next in the paragraph about *Universal Composability.*

Furthermore, while computational soundness allows us to close the gap between the symbolic and the computational model, soundness results are very inflexible. They hold for a fixed set of cryptographic primitives only and are not composable.

Although we need to prove soundness only once for a set of primitives and can then use the result for the analysis of all protocols based on this set, we have to redo the complete soundness proof if we need additional primitives. Additionally, this inflexibility makes soundness results for large sets of primitives hard to achieve. We turn to this problem in detail in the paragraph about *Composable Soundness.*

Universal Composability.

The problem of unwieldy proofs for complex systems has already been identified in the computational model more than ten years ago. As a consequence, frameworks for so called *universal composability* (UC) were devised by Canetti et. al [36] and Backes et. al [11]. Intuitively, they combine the idea of simulation-based security with composability:

Say that a protocol π uses a functionality $\mathcal{F}$ and $\mathcal{R}$ is a secure realization of $\mathcal{F}$ (this is very much defined as described in simulation-based security above). Then, the UC framework features a composition theorem giving us that $\pi[\mathcal{R}]$ realizes $\pi[\mathcal{F}]$. Consequently, we can conduct two security proofs independently to get a security guarantee (captured by $\mathcal{F}'$) for $\pi[\mathcal{R}]$:

- $\mathcal{R}$ realizes $\mathcal{F}$.
- $\pi[\mathcal{F}]$ realizes $\mathcal{F}'$.

This technique gives us two important advantages: Usually, $\mathcal{F}$ is less complex than $\mathcal{R}$. Hence proving that $\pi[\mathcal{F}]$ realizes $\mathcal{F}'$ is much easier than proving that $\pi[\mathcal{R}]$ realizes $\mathcal{F}'$ directly. Furthermore, say that we have another realization $\mathcal{R}'$ for $\mathcal{F}$, then we can replace $\mathcal{R}$ by $\mathcal{R}'$ without re-doing the security proof for the full protocol. Furthermore, we can re-use that fact that $\mathcal{R}$ realizes $\mathcal{F}$ for each protocol that uses $\mathcal{F}$.

Our Contribution: Symbolic Universal Composability.

While we have the UC framework in the computational model to tame large proofs, we need it in the symbolic model to combine the advantages of automation with modularity.

The frameworks of Delaune et. al [54] and Böhl [23] are first steps in this direction. However, their results still suffer from major drawbacks. The framework of [54] cannot be used to capture important security properties like anonymity. We explain the differences between [54] and our results in detail in Section 2.6. While the work of Böhl solved this problem, it only features a restricted composition theorem that lacks the important concurrent composition.

In this work, we present the first fully fledged framework for composition in the symbolic model. This result makes the model suitable for the efficient analysis of large cryptographic protocols.

Our Contribution: Composable Computational Soundness Proofs.

Backed up by the new capabilities acquired for the symbolic model, we turn our focus to the security guarantees we can get. As explained above, a computational soundness result gives us guarantees against a computational adversary after a symbolic analysis. Although such a result needs to be proven only once for a given abstraction of cryptographic primitives, changes in the abstraction are a big nuisance. To analyze a protocol based on a set of cryptographic primitives that are not, at least not in this combination, part of a computational soundness result, a new proof of soundness has to be conducted from scratch. Furthermore, due to the complexity of computational soundness proofs, they usually cover only one or two cryptographic primitives. To get the soundness results we need to back up our symbolic analysis, we need a better flexibility while conducting the proof and techniques to get soundness for larger sets of cryptographic primitives. It would be optimal to have modular proofs of computational soundness at the level of cryptographic primitives. That is, we automatically get soundness for a symbolic model featuring different cryptographic primitives if there is a composable soundness result for each of the primitives.

A first step in this direction is the work of Cortier and Warinschi [48]. They establish a notion for composable soundness, dubbed *deduction soundness*. An implementation is deduction sound for a symbolic model if soundness holds even in the presence of certain adversarially picked functions. This gives us leverage when extending the symbolic model. If we can embed the extension as adversarially picked functions, soundness still holds. [48] use this technique to show a composable soundness result for public-key encryption. More concretely, any deduction sound implementation can be extended with public-key encryption without further proof.

We extend the result of Cortier and Warinschi by presenting composable soundness results for signatures, hashes, MACs and symmetric encryption. Since [48] expected that additional requirements would be necessary to achieve composable soundness for signatures, this part of our results is particularly interesting. Altogether, we present the largest existing soundness result by capturing the combination of all important cryptographic primitives. Additionally, this soundness result is composable, that is, every deduction sound implementation maintains soundness when extended with the primitives mentioned above. The importance of composition cannot be overemphasized: obtaining such general results without being able to study each primitive separately would be unmanageable.

Further Related Work.

Symbolic Universal Composability. Apart from the already mentioned works of Canetti et. al [36] and Backes et. al [11] on the computational side and Delaune et. al [54] and Böhl [23] on the symbolic side, there is a large body of further related work for symbolic universal composability on the computational side. Other models based on the same ideas are SPPC [51], IITM [64], Task-PIOA [41, 42], and GNUC [62]. Some of our results are adaptations of existing computational soundness results: the impossibility of commitments [37] in Section 2.8.3 and the joint state technique [39] in Section 2.7. Finally, the symbolic setting is not the first example of the fact that the UC framework can easily be adapted to other settings to get different or stronger security guarantees, e.g., GUC (UC with shared functionalities) [43], quantum-UC [73, 72], UC with local adversaries [40], UC/c (incoercibility) [74], UC with everlasting security [67]. Additionally, links between UC and symbolic models occurred where UC-like models were used to establish computational soundness results [9, 38]. Furthermore, [69, 12] present UC protocol constructions where impossibilities are circumvented by giving the simulator additional power (namely superpolynomial-time computation); this shows some parallels to our "virtual primitives"-approach, see the discussion on page 83.

Computational Soundness. The first computational soundness result by Abadi and Rogaway [2] started a whole field of research on this matter. Many interesting computational soundness results have been established since then. They cover basically all standard cryptographic primitives: symmetric encryption [8, 46], asymmetric encryption [9, 49, 52], signatures [9, 63, 49], MACs [10], hashes [63, 50, 58]. For a more comprehensive list we refer the reader to the survey on computational soundness by Cortier, Kremer, and Warinschi [47].

Outline of this Thesis.

In Chapter 2 we present our results for symbolic universal composability. Section 2.1 introduces the applied pi calculus that is the formal foundation of our symbolic model. The composition theorem (Theorem 1) can be found in Section 2.4. Readers interested in the technical problems connected with concurrent composition should probably have a look at the part starting with Lemma 18 on page 48. In Section 2.2 we provide some general lemmas for the applied pi calculus that might also be useful in other contexts. Finally, we present proof of concept in Section 2.7 where we modularly construct a secure channel from the widely-known NSL protocol and a PKI. We conduct a significant part of the proofs using Proverif.

In Chapter 3 we present our work on composable computational soundness. Note that the notion of a symbolic model in this chapter differs from that in Chapter 2. We discuss and motivate these differences in Section 3.2.1. In Section 3.6 we give the definition of deduction soundness. The soundness results for the different cryptographic primitives can be found in Section 3.7.2 for public-key encryption, Section 3.7.3 for signatures, Section 3.7.4 for symmetric encryption, Section 3.7.5 for message authentication codes, and Section 3.7.6 for hash functions.

Finally, in Chapter 4 we give some interesting directions for future work.

Other Results.

While this thesis is concerned with symbolic universal composability and computational soundness, there are more results I worked on during my time as a PhD student. These make a short appearance here.

On Definitions of Selective Opening Security. In [28], we investigate different notions of security against selective opening attacks and their relations. In a selective opening attack, the adversary observes some ciphertexts encrypted under the same public key (e.g., we could think of sensors that send their data encrypted to a central node for collection). The adversary can then adaptively corrupt senders and not only learn the messages sent but also the randomness used for encryption (we say it can adaptively *open* ciphertexts). Intuitively, the unopened ciphertexts should not help the adversary to learn something about their contents. There are several notions of security trying to capture the setting of selective opening attacks, each with its merits and drawbacks. In [28], we give concrete counterexamples showing that two important definitions of selective opening security do not imply one another. This paper received the best paper award at PKC 2012.

Practical Signatures from Standard Assumptions. In [34], we construct new digital signature schemes from standard assumptions, namely CDH, RSA, and SIS. The efficiency of our new schemes compares favorably against existing schemes. Furthermore, we developed a new proof technique we dub “confined guessing” to achieve the results. This technique provides a way to efficiently transform a tag-based signature scheme into a standard scheme. This Eurocrypt paper is a merge of [71] and [32].

Encryption schemes secure under related-key and key-dependent message attacks. In [26], we construct symmetric encryption schemes that are secure against related-key and key-dependent message attacks (RKA/KDM security). In a related-key attack, the adversary does not only get ciphertexts under the secret key k, but also under keys $\varphi(k)$ derived from k for φ picked by the adversary from a given class of functions. Security of messages in the sense of IND-CPA security should still hold, i.e., the adversary should still not be able to distinguish $\mathsf{Enc}(\varphi(k), m)$ from $\mathsf{Enc}(\varphi(k), 0^{|m|})$. For security against key-dependent message attacks, we similarly stipulate that $\mathsf{Enc}(k, \psi(k))$ is indistinguishable from $\mathsf{Enc}(\varphi(k), 0^{|\psi(k)|})$. Both RKA and KDM security are not implied by the standard IND-CPA security. The only existing RKA-KDM secure scheme is that of Applebaum [3], which is secure under the LPN assumption. We provide a construction to get schemes based on various other computational assumptions, namely DDH, LWE, QR, and DCR.

Verification of Secure Two-Party-Computations Implemented in Java. In [27], we provide a proof of correctness and security of a two-party-computation protocol based on garbled circuits and oblivious transfer in the presence of a semi-honest sender. To achieve this we are the first to combine a machine-assisted proof of correctness with advanced cryptographic primitives to prove security properties of Java code. The machine-assisted part of the proof is conducted with KeY, an interactive theorem prover. The proof includes a correctness result for the construction and evaluation of garbled circuits. This is particularly interesting since checking such an implementation by hand would be very tedious and error-prone. Although we stick to the secure two-party-computation of an n-bit AND in this paper, our approach is modular, and we explain how our techniques can be applied to other functions.

Evaluating the Usability of Interactive Theorem Provers. In recent years the effectiveness of interactive theorem provers has increased in a way that the bottleneck in the proof process shifted from effectiveness to efficiency. While in principle large theorems are provable, it takes a lot of effort for the user to interact with the system. A major obstacle for the user is to understand the proof state in order to guide the prover in successfully finding a proof. We conducted two focus groups to evaluate the usability of interactive theorem provers. We wanted to evaluate the impact of the gap between the user's model of the proof and the actual proof performed by the provers' strategies. In addition, our goals were to explore which mechanisms already exist and to develop, based on the existing mechanisms, new mechanisms that help the user in bridging this gap. [15] explains how the method of focus group could be applied to evaluate the usability of interactive theorem provers while [14] presents our results.

List of Own Publications

[14] Bernhard Beckert, Sarah Grebing, and Florian Böhl. "A Usability Evaluation of Interactive Theorem Provers Using Focus Groups". In: *Proceedings, Workshop on Human-Oriented Formal Methods (HOFM), Grenoble, September 2014*. LNCS. to appear. Springer, 2014.

[15] Bernhard Beckert, Sarah Grebing, and Florian Böhl. "How to Put Usability into Focus: Using Focus Groups to Evaluate the Usability of Interactive Theorem Provers". In: *Proceedings, Workshop on User Interfaces for Theorem Provers (UITP), Vienna, July 2014*. Ed. by Christoph Benzmüller and Bruno Woltzenlogel Paleo. EPTCS. to appear. 2014.

[24] Florian Böhl, Véronique Cortier, and Bogdan Warinschi. "Deduction soundness: prove one, get five for free". In: *ACM Conference on Computer and Communications Security*. ACM, 2013, pp. 1261–1272.

[26] Florian Böhl, Gareth T. Davies, and Dennis Hofheinz. "Encryption Schemes Secure under Related-Key and Key-Dependent Message Attacks". In: *Public Key Cryptography*. LNCS. Springer, 2014, pp. 483–500.

[27] Florian Böhl, Simon Greiner, and Patrik Scheidecker. "Proving Correctness and Security of Two-Party Computation Implemented in Java in Presence of a Semi-Honest Sender". In: *CANS*. LNCS. to appear. Springer, 2014.

[28] Florian Böhl, Dennis Hofheinz, and Daniel Kraschewski. "On Definitions of Selective Opening Security". In: *Public Key Cryptography*. LNCS. Springer, 2012, pp. 522–539.

[30] Florian Böhl and Dominique Unruh. "Symbolic Universal Composability". In: *CSF*. IEEE, 2013, pp. 257–271.

[33] Florian Böhl, Dennis Hofheinz, Tibor Jager, Jessica Koch, and Christoph Striecks. "Confined Guessing: New Signatures From Standard Assumptions". In: *J. Cryptology* to appear (2015).

[34] Florian Böhl, Dennis Hofheinz, Tibor Jager, Jessica Koch, Jae Hong Seo, and Christoph Striecks. "Practical Signatures from Standard Assumptions". In: *EUROCRYPT*. LNCS. Springer, 2013, pp. 461–485.

2. Symbolic Universal Composability

Contributions of this Chapter.

In this chapter we present the results of [31]. In particular, we transfer the ideas of the UC framework in the computational setting to the symbolic setting. We show that the composition theorem and the fact that security properties carry over still hold in the symbolic UC framework. (Concurrent composition turns out to be non-trivial because we need to encode a special variant of process replication in the applied pi calculus that provides session ids to replicated processes.) We present an example analysis of a key exchange using the Needham-Schroeder-Lowe protocol, and how to use it in a secure channel protocol via composition.

We show that impossibilities from the computational UC framework unfortunately still apply in the symbolic setting; in particular, implementing a commitment functionality without any trusted setup is impossible. On the positive side, we show that this impossibility can be circumvented to a large part by a trick that we call "virtual primitives"; here we perform the proof of security under the assumption that the cryptographic primitives have some exotic features, but in the end conclude security for the original cryptographic primitives without these exotic features. This "virtual primitives"-approach is unique to the symbolic setting. To the best of our knowledge no corresponding technique exists in the computational world.

We also show how to use Proverif as a helping tool for performing the observational equivalence proofs when showing security in our framework. For this we develop a set of lemmas that help in rewriting processes and allows us to use Proverif as a tool even for observational equivalence proofs that do not involve so-called biprocesses and are thus out of the scope of Proverif. (See Section 2.7.) We believe that this set of lemmas is useful also in other settings than that of our work.

Organization.

In Section 2.1 we review the applied pi calculus as used here (with some additional concepts introduced in Section 2.1.2). Section 2.3 introduces our security definition. Section 2.4 shows the composition theorem, Section 2.5 that we have property preservation. In Section 2.7 we give an example of our framework based on the Needham-Schroeder-Lowe protocol, we illustrate composition and joint state techniques (the latter is a technique for dealing with functionalities shared by several protocol instances). Finally, in Section 2.8 we present the virtual primitives technique.

2.1 Review of the applied pi calculus

In this section we review the variant of the applied pi calculus from [22] that we use in our paper. Below (Section 2.1.2) we list some non-standard definitions that we will use. Readers familiar with the applied pi calculus can directly skip to that section.

The process calculus presented in [22] is a combination of the original applied pi calculus [1] and one of its dialects [19].

We have a set of terms that is built upon three basic sets. The infinite set of *names* $\mathcal{N}$, the infinite set of *variables* $\mathcal{V}$ and the set of *function symbols* (called the *signature* Σ). Names describe all kinds of atomic data, i.e. are used as nonces or to represent messages. We distinguish two categories of function symbols: constructors, which are used to construct terms of higher order, and destructors. Let $\mathcal{T}(\Sigma)$ be the set of terms built from names in $\mathcal{N}$, variables in $\mathcal{V}$ and constructors in Σ.

A *substitution* is a function from variables to terms $\sigma : \mathcal{V} \to \mathcal{T}(\Sigma)$. For a term T, $T\sigma$ denotes the substitution of every variable x in T by $\sigma(x)$ (all variables are replaced at once). We write $\{M_1/x_1, \dots, M_n/x_n\}$ for a substitution σ s. t. $\sigma(x_i) = M_i$ and $\sigma(x) = x$ for all $x \in \mathcal{V} \setminus \{x_1, \dots, x_n\}$.

Sometimes it is desirable to consider two terms, that were constructed differently, equivalent. Therefore we have a finite set E of equations (M, N) (for $M = N$) where M and N are terms that contain only variables and constructors. E is called *equational theory.*

The equivalence relation $=_{\mathrm{E}}$ on terms is defined as the reflexive, transitive and symmetric closure of E closed under the application of substitutions[1] and contexts (i.e. for all terms M, N and T we have $M =_{\mathrm{E}} N \Rightarrow T\{M/x\} =_{\mathrm{E}} T\{N/x\}$).

To define the semantics of a destructor d, we introduce a finite set R of rewrite rules $d(M_1, \dots, M_n) \to_P M$ where M and M_i, $i \in \{1, \dots, n\}$ are terms that contain only variables and constructors and the variables in M must be a subset of the variables used in $M_1, \dots, M_n$, and P is a predicate on n-tuples of terms invariant under $=_{\mathrm{E}}$ [2]. (We write $\to$ instead of $\to_P$ when $P(\dots) = \mathit{true}$ always.) Analogous to [22], we introduce the rewrite rule $f(x_1, \dots, x_n) \to f(x_1, \dots, x_n)$ for each constructor $f \in \Sigma$ to avoid case distinctions between constructors in the definitions later on. (Destructors with conditional rewrite rules have been introduced in Proverif 1.87, see also [45]. None of our results need this additional generality. However, we explain in Section 2.8.1.1 why such destructors can be useful in some cases.)

$D \Downarrow M$ denotes the evaluation of D to M where D is a *destructor term*, i.e., a term or the application of a function to destructor terms. For all terms M we define $M \Downarrow M$ (i.e. when evaluating a term we obtain the term itself). If we have $D = g(D_1, \dots, D_n)$ for a function g where D_i are destructor terms we define $g(D_1, \dots, D_n) \Downarrow M\sigma$ for substitution σ iff there is a rewrite rule $g(M_1, \dots, M_n) \to_P M$ and terms $N_1, \dots, N_n$ s.t. $D_i \Downarrow N_i$, $N_i =_{\mathrm{E}} M_i\sigma$, and $P(N_1, \dots, N_n) = \mathit{true}$.

[1] I.e., for every substitution σ and $M =_{\mathrm{E}} N$ we have $M\sigma =_{\mathrm{E}} N\sigma$.

[2] I.e., "Invariant under $=_{\mathrm{E}}$" means that if $N_i =_{\mathrm{E}} N_i'$ for $i = 1, \dots, n$, then $P(N_1, \dots, N_n) = P(N_1', \dots, N_n')$.

$$\begin{aligned} P ::= {} & \mathbf{0} \\ & P|Q \\ & !P \\ & M(x).P \\ & \overline{M}\langle N\rangle.P \\ & \text{let } x = D \text{ in } P \text{ else } Q \\ & \nu a.P \end{aligned}$$

Figure 2.1: Syntax of processes in the applied pi calculus

Definition 1 (Symbolic model)**.** *By* symbolic model*, denoted* $\mathcal{M} = (\Sigma, \mathrm{E}, \mathrm{R})$*, we refer to the entity of a signature* Σ*, a finite set of equations* E *and a finite set of rewrite rules* R*.*

Note that the infinite set of names and infinite set of variables are not explicitly part of the symbolic model since they are not specific for any concrete model in our setting. We refer to them globally as $\mathcal{N}$ *and* $\mathcal{V}$ *respectively.*

Except for Section 2.8, it will be clear from the context which symbolic model we use. In Section 2.8 we focus on the relation between different symbolic models. Only then we will introduce a notation that explicitly states the symbolic model underlying a property, e.g., observational equivalence of two processes.

We can describe processes in our process calculus using the inductively defined grammar from Figure 2.1. For a better understanding of the syntax we anticipate the following section about its semantics and give a quick overview of the intuition connected to the syntax. The **0**-process simply does nothing and terminates (and is therefore often omitted). Two processes, P and Q, can be executed in parallel (denoted $P|Q$). They may interact with each other or with the environment independently of each other. A replication ($!P$) behaves as an infinite number of copies (instances) of P running in parallel. The scope of a name n may be restricted to a process P ($\nu n.P$). $M(x).P$ allows P to receive a message (a term) T on a channel *identified* by the term M. The variable x is used in P as a reference to the input. The counterpart of $M(x)$ is $\overline{M}\langle T\rangle.P$ which sends a message (a term) T on M and then behaves like P.

In let $x = D$ in P else Q the symbol D stands for a term or a destructor term. If we have $D \Downarrow M$ for a term M the process behaves like $P\{M/x\}$ otherwise it behaves like Q.

Except for the let-statement and parallel execution, processes do have the structure $\mathsf{statement}.P$ and we say for P (or any part of P) that it is *under* the $\mathsf{statement}$ (e.g. we say that "P is under a bang" in $!P$ or that P is under an input in $c(x).\nu n.P$). We say that P is under a let if P occurs in one of the two branches of a let.

An occurrence of a name n in a process is *bound* if it is under a νn. An occurrence of a variable x is bound if it is under a $M(x)$ or in the P-branch of a let $x = D$ in P else Q. $bn(P)$ resp. $bv(P)$ denotes the set of names resp. variables with bound occurrences in P. If an occurrence is not bound, it is called *free*, and $fn(P)$, $fv(P)$ denote the corresponding sets for names resp. variables. A process is *closed* if it has no free variables.

$$\begin{array}{lrcl}
\text{PAR-}\mathbf{0} & P & \equiv & P \mid \mathbf{0} \\
\text{PAR-A} & P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R \\
\text{PAR-C} & P \mid Q & \equiv & Q \mid P \\
\text{NEW-C} & \nu u.\nu v.P & \equiv & \nu v.\nu u.P \\
\text{NEW-PAR} & u \notin \mathit{fn}(P) \Rightarrow & & \\
 & P \mid \nu u.Q & \equiv & \nu u.(P \mid Q)
\end{array}$$

Figure 2.2: Rules for structural equivalence

$$\begin{array}{lrl}
\text{REPL} & & !P \to P|!P \\
\text{COMM} & & \overline{C}\langle T\rangle.P \mid C'(x).Q \\
 & \to & P \mid Q\{T/x\} \text{ if } C =_{\mathrm{E}} C' \\
\text{LET-THEN} & & \text{let } x = D \text{ in } P \text{ else } Q \\
 & \to & P\{M/x\} \;\text{ if } D \Downarrow M \\
\text{LET-ELSE} & & \text{let } x = D \text{ in } P \text{ else } Q \\
 & \to & Q \text{ if } \nexists M \text{ s.t. } D \Downarrow M
\end{array}$$

Figure 2.3: Rules for internal reduction

A *context* $\mathcal{C}$ is a process where exactly one occurrence of $\mathbf{0}$ is replaced with $\Box$. $\mathcal{C}[P]$ denotes the process resulting from the replacement of $\Box$ with P in $\mathcal{C}$. An *evaluation context* is a closed context $\mathcal{C}$ built from $\Box$, $\mathcal{C}|P$, $P|\mathcal{C}$, and $\nu a.\mathcal{C}$. We call an occurrence of a term or process within a process *unprotected* if it is only below parallel compositions ($|$) and restrictions (ν).

Definition 2 (Structural equivalence ($\equiv$))**.** Structural equivalence*, denoted* $\equiv$*, is the smallest equivalence relation on processes that is closed under* α*-conversion*[3] *on names and variables, application of evaluation contexts and the rules from Figure 2.2.*[4]

Definition 3 (Internal reduction ($\to$))**.** Internal reduction*, denoted* $\to$*, is the smallest relation on closed processes closed under structural equivalence and application of evaluation contexts such that the rules from Figure 2.3 hold for any closed processes* P *and* Q*.* $\to^*$ *denotes the reflexive, transitive closure of* $\to$*.*

A closed process P *emits* on M (denoted $P \downarrow_M$) if $P \equiv \mathcal{C}[\overline{M'}\langle N\rangle.Q]$ for some evaluation context $\mathcal{C}$ that does not bind $\mathit{fn}(M)$ and $M =_{\mathrm{E}} M'$.[5] Analogously it *reads* on M (denoted $P \uparrow_M$) if $P \equiv \mathcal{C}[M'(N).Q]$. We say that P *communicates* on M (denoted $P \updownarrow_M$) if $P \downarrow_M$ or $P \uparrow_M$.

[3] An α-conversion is a renaming process that does not change the meaning of a term. E.g. renaming b to c in $\nu a.\nu b.\overline{net}\langle a\rangle.\overline{net}\langle b\rangle$ is a valid α-conversion (and thus we have that $\nu a.\nu b.\overline{net}\langle a\rangle.\overline{net}\langle b\rangle \equiv \nu a.\nu c.\overline{net}\langle a\rangle.\overline{net}\langle c\rangle$), renaming b to a is not.

[4] We differ from [22] by defining $\equiv$ also for non-closed processes. But on closed processes, our definition coincides with that from [22].

[5] It is indeed intentional that the definition requires $\mathcal{C}$ not to bind $\mathit{fn}(M)$ (as opposed to $\mathit{fn}(M')$) even though we consider the process $\mathcal{C}[\overline{M'}\langle N\rangle.Q]$. This way the definition is equivalent to the following: $P \downarrow_M$ iff $P \equiv_{\mathrm{E}} \mathcal{C}[\overline{M}\langle N\rangle.Q]$ for some evaluation context $\mathcal{C}$ not binding $\mathit{fn}(M)$, and some process Q [17]. Here $\equiv_{\mathrm{E}}$ is structural equivalence modulo replacing terms by equivalent ones, see Definition 6.

Definition 4. *A* simulation $\mathcal{R}$ *is a relation on closed processes such that* $(P, Q) \in \mathcal{R}$ *implies*

(i) if $P \downarrow_M$ *then for some* Q' *we have that* $Q \to^* Q'$ *and* $Q' \downarrow_M$

(ii) if $P \to P'$ *then for some* Q' *we have that* $Q \to^* Q'$ *and* $(P', Q') \in \mathcal{R}$

(iii) $(\mathcal{C}[P], \mathcal{C}[Q]) \in \mathcal{R}$ *for all evaluation contexts* $\mathcal{C}$*.*

A relation $\mathcal{R}$ *is a* bisimulation *if both* $\mathcal{R}$ *and* $\mathcal{R}^{-1}$ *are a simulation.*

Observational equivalence *(≈) is the largest bisimulation.*

It is easy to check that the transitive hull of $\approx$ satisfies the conditions (i), (ii) and (iii) from above. Hence $\approx$ contains its own transitive hull and thus is indeed an equivalence relation.

Substitutions on processes work like substitutions on terms but must additionally respect the scopes of names and variables (bound or free). Since renaming of bound names and variables doesn't change the structural equivalence class of a process we assume w.l.o.g. from now on that for $P\sigma$ we have $\sigma(x) = x$ for all $x \in bv(P)$ and $\sigma(x)$ does not contain names $n \in bn(P)$ for all $x \in fv(P)$.

2.1.1 Syntactic sugar

We introduce if $D = D'$ then P else Q as syntactic sugar for let $x = equals(D, D')$ in P else Q where x must not occur in P or Q and D, D' are destructor terms. Note that we assume the existence of an *equals* destructor with the rewrite rule $equals(x, x) \to x$ throughout this paper (see Definition 5 (iii)). Furthermore, we write $C().P$ for $C(x).P$ where x is a fresh variable, and $\overline{C}\langle\rangle.P$ for $\overline{C}\langle empty\rangle$ assuming a nullary constructor *empty* (see Definition 5 (i)).

Later, when dealing with Proverif processes, e.g., in Definition 31, we use the Proverif syntax for pattern matching in inputs and lets: E.g., (let $(=n, x) = D$ in P else Q) executes $P\{T/x\}$ if $D \Downarrow (n, T)$ (i.e., D has to evaluate to a pair with n beeing the first value while x is used as a reference for the arbitrary second value T) and Q otherwise. Inputs of type $C((x, \_))$ expect a pair as input where the first value is referenced by x while the second value is dropped (i.e., when receiving an input (T, T') on C, $C((x, \_)).P$ continues to run as $P\{T/x\}$. For more details see the Proverif manual [21]. We stress that these constructions are just syntactic sugar and can be replaced by statements accoding to the grammar of the pi calculus we described above.

2.1.2 Additional concepts used in this work

In this section, we describe several nonstandard concepts related to the applied pi calculus that we use in this work.

Miscellaneous.

A context always contains a single occurrence of the hole. Sometimes we need a context which may or may not contain a hole: A *0-1-context* is defined like a context, except that there may be zero or one occurrences of the hole.

We refer to occurrences of terms that identify channels in a process as *channel identifiers*. E.g., in $\overline{M}\langle T\rangle$ M is a channel identifier and T is not – even if M and T were the same term (because M and T are different occurrences).

We allow destructors with conditional rewrite rules following [45], see page 14. None of our results actually requires these conditional destructors, though. The reader may safely assume the usual, unconditional definition of constructors.

Natural symbolic models.

A number of lemmas in this paper only hold when the symbolic model we use satisfies certain natural conditions. Instead of stating these explicitly each time, we collect all these conditions in the following definition:

Definition 5 (Natural symbolic model)**.** *We say a symbolic model is* natural *if it satisfies the following conditions:*

(i) *there is a constructor* $\mathit{empty}/0 \in \Sigma$*,*

(ii) *a constructor for pairings, denoted* $(\Box, \Box)$*, is part of the signature* Σ*,*

(iii) *there is a destructor* $\mathit{equals}/2 \in \Sigma$ *with rewrite rule* $\mathit{equals}(x, x) \to x$ *and no further rewrite rules that contain* equals*,*

(iv) *there are destructors* $\mathit{fst}/1, \mathit{snd}/1 \in \Sigma$ *with rewrite rules* $\mathit{fst}((x, y)) \to x$ *and* $\mathit{snd}((x, y)) \to y$*,*

(v) *for all terms* T*,* T_1 *with* $\mathit{fst}(T) \Downarrow T_1$ *there exists a term* T_2 *with* $\mathit{snd}(T) \Downarrow T_2$ *and furthermore* $(T_1, T_2) =_{\mathrm{E}} T$ *for all such* T_2 *and vice versa,*

(vi) *for arbitrary terms* T_1, T_2, T'_1, T'_2 *we require that* $(T_1, T_2) =_{\mathrm{E}} (T'_1, T'_2)$ *entails* $T_1 =_{\mathrm{E}} T'_1$ *and* $T_2 =_{\mathrm{E}} T'_2$*,*

(vii) *for any destructor term* D *and any name* $n \notin \mathit{fn}(D)$ *we require that* $D \Downarrow T$ *for a term* T *entails the existence of a term* T' *with* $D \Downarrow T'$*,* $n \notin \mathit{fn}(T')$ *and* $T =_{\mathrm{E}} T'$*,*

(viii) *there are terms* T, T' *with* $T \neq_{\mathrm{E}} T'$*.*

In the following, we will always assume that the symbolic model is natural in the sense of Definition 5.

Equivalence of processes modulo rewriting.

Structural equivalence $\equiv$ does not allow us to replace a term M by another term $M' =_{\mathrm{E}} M$. In some places, we will therefore need to apply $=_{\mathrm{E}}$ to processes, and we will also use an extension $\equiv_{\mathrm{E}}$ of $\equiv$ that allows us to replace terms:

Definition 6. *We extend* $=_{\mathrm{E}}$ *to destructor terms and processes as follows:*

Given two destructor terms D, D'*, we have* $D =_{\mathrm{E}} D'$ *iff* D *can be rewritten into* D' *by replacing subterms by* $=_{\mathrm{E}}$*-equivalent subterms. (But replacing destructors is not allowed. E.g., if* d *is a destructor and* f, g *are constructors, and* $f(x) =_{\mathrm{E}} g(x)$ *is in the equational theory, we have* $d(f(a)) =_{\mathrm{E}} d(g(a))$ *but not* $f(d(a)) =_{\mathrm{E}} g(d(a))$*. Formally,* $=_{\mathrm{E}}$ *is the smallest equivalence relation on destructor terms such that* $D\{M/x\} =_{\mathrm{E}} D\{M'/x\}$ *for destructor terms* D *and terms* $M =_{\mathrm{E}} M'$*.*

Given two processes P, P'*, we have* $P =_{\mathrm{E}} P'$ *iff* P *can be rewritten into* P' *by* α*-conversion and by replacing terms and destructor terms by* $=_{\mathrm{E}}$*-equivalent ones. Formally,* $=_{\mathrm{E}}$ *is the smallest equivalence relation closed under* α*-renaming such that* $P\{M/x\} =_{\mathrm{E}} P\{M'/x\}$ *for processes* P *and terms* $M =_{\mathrm{E}} M'$*.*

Given two processes P, P'*, we have* $P \equiv_{\mathrm{E}} P'$ *iff* P *can be rewritten into* P' *by* $=_{\mathrm{E}}$ *and* $\equiv$*. Formally,* $\equiv_{\mathrm{E}} \; := \; (=_{\mathrm{E}} \cup \equiv)^*$*.*

Full observational equivalence.

A substitution σ is a *closing substitution* if $P\sigma$ is closed. We call two (not necessarily closed) processes P and Q *fully observationally equivalent* (denoted $P \approxeq Q$) iff $P\sigma \approx Q\sigma$ for all closing substitutions σ (where we implicitly assume that the bound names in P, Q are renamed so that they are distinct from the free names of σ). Since $\approx$ is closed under $\equiv$ it follows in a straightforward way that $\approxeq$ is closed under $\equiv$.

The motivation behind the definition of $\approxeq$ is the following lemma which allows us to replace fully observationally equivalent subprocesses by each other.

Lemma 1. *Let P and Q be processes and $P \approxeq Q$. Then $\mathcal{C}[P] \approxeq \mathcal{C}[Q]$ for every context $\mathcal{C}$.*

To show this lemma, we first prove the following lemma:

Lemma 2. *Let P and Q be closed processes. We have $P \approx Q \Rightarrow\ !P \approx\ !Q$.*

Proof. We define a relation $\mathcal{R} := \approx \cup \{(\nu \underline{n}.(IP|!P), \nu \underline{n}.(IQ|!Q)) : IP, IQ$ closed processes with $IP \approx IQ$ and $\underline{n}$ a vector of names $\}$ closed under structural equivalence. Intuitively, IP and IQ represent the running instances of P resp. Q. For $(A, B) \in \mathcal{R}$ we show the three points of observational equivalence.

If $(A, B) \in \approx$ there is nothing to show. Otherwise $(A, B) = (\nu \underline{n}.(IP|!P), \nu \underline{n}.(IQ|!Q))$.

- If $\nu \underline{n}.(IP|!P) \downarrow_M$ we have $\nu \underline{n}.IP \downarrow_M$ and, since $IQ \approx IQ$, $\nu \underline{n}.IQ \downarrow_M$. Therefore $\nu \underline{n}.(IQ|!Q) \downarrow_M$.
- For internal reductions $\rightarrow$ in $\nu \underline{n}.(IP|!P)$ we distinguish two cases:
  - A new instance of P spawns, i.e., $\nu \underline{n}.(IP|!P) \rightarrow \nu \underline{n}.(IP|P|!P)$. We define $IP' := IP|P$ and IQ' analogously. Then there is a corresponding internal reduction (following the REPL rule) for the Q-side $\nu \underline{n}.(IQ|!Q) \rightarrow \nu \underline{n}.(IQ'|!Q)$ and therefore $(\nu \underline{n}.(IP'|!P), \nu \underline{n}.(IQ'|!Q) \in \mathcal{R}$ (note that $IP' \approx IQ'$ since $IP \approx IQ$ and $P \approx Q$).
  - The reduction $\rightarrow$ only affects $!P$ structurally. That is, we basically have $\nu \underline{n}.(IP|!P) \rightarrow \nu \underline{n}.(IP'|!P)$. Since $IP \approx IQ$ we find IQ' s.t. $IQ \rightarrow^* IQ'$ and $IP' \approx IQ'$. Hence $(\nu \underline{n}.(IP'|!P), \nu \underline{n}.(IQ'|!Q)) \in \mathcal{R}$.
- For any evaluation context $\mathcal{C}$ we have $\mathcal{C}[\nu \underline{n}.(IP|!P)] \equiv \nu \underline{n}'.(\mathcal{C}'[IP]|!P)$ where $\mathcal{C}'$ is $\mathcal{C}$ with all restrictions moved into $\underline{n}'$. Analogously we have $\mathcal{C}[\nu \underline{n}.(IQ|!Q)] \equiv \nu \underline{n}'.(\mathcal{C}'[IQ]|!Q)$ with the same $\mathcal{C}'$, $\underline{n}'$. Since $\mathcal{C}'$ is an evaluation context, $\mathcal{C}'[IP] \approx \mathcal{C}'[IQ]$. Altogether we have $(\nu \underline{n}'.(\mathcal{C}'[IP]|!P), \nu \underline{n}'.(\mathcal{C}'[IQ]|!Q)) \in \mathcal{R}$.

This concludes our proof since the definition of $\mathcal{R}$ is symmetric. □

We can now show Lemma 1:

Proof of Lemma 1. First consider the case that $\mathcal{C}$ is an evaluation context which is allowed to have free variables here. For all closing substitutions σ we have $P\sigma \approx Q\sigma$ and hence $\mathcal{C}\sigma[P\sigma] \approx \mathcal{C}\sigma[Q\sigma]$. Therefore $\mathcal{C}[P]\sigma \approx \mathcal{C}[Q]\sigma$ which entails $\mathcal{C}[P] \approxeq \mathcal{C}[Q]$.

To expand the proof from evaluation contexts to general contexts $\mathcal{C}$ we show the following properties for $\approxeq$ from which the Lemma immediately follows by induction:

1. If $P \approxeq Q$ then $\overline{M}\langle T\rangle.P \approxeq \overline{M}\langle T\rangle.Q$ for arbitrary terms M and T:
   Let σ be a closing substitution for $\overline{M}\langle T\rangle.P$ and $\overline{M}\langle T\rangle.Q$. We define the relation $\mathcal{R} := \approx \cup \{(\mathcal{C}[(\overline{M}\langle T\rangle.P)\sigma], \mathcal{C}[(\overline{M}\langle T\rangle.Q)\sigma]) : \mathcal{C}$ closed evaluation context$\}$ closed under structural equivalence. We show that $\mathcal{R}$ satisfies the three points of observational equivalence. Let $(A, B) \in \mathcal{R}$. For $(A, B) \in \approx$ there is nothing to do. Otherwise $(A, B) = (\mathcal{C}[(\overline{M}\langle T\rangle.P)\sigma], \mathcal{C}[(\overline{M}\langle T\rangle.Q)\sigma])$ for some closed evaluation context $\mathcal{C}$.

- $A \downarrow_N$: If $\mathcal{C}[\mathbf{0}] \downarrow_N$ obviously $B \downarrow_N$ as well. Otherwise $(\overline{M}\langle T\rangle.P)\sigma \downarrow_N$ where the free names of N are not bound by $\mathcal{C}$ which requires $N =_{\mathrm{E}} M$ and hence leads to $(\overline{M}\langle T\rangle.Q)\sigma \downarrow_N \Rightarrow B \downarrow_N$.
- For internal reductions in A we distinguish two cases:
  - $\rightarrow$ is the COMM reduction $\mathcal{C}[(\overline{M}\langle T\rangle.P)\sigma] \rightarrow \mathcal{C}'[P\sigma]$ (up to structural equivalence). In the same way we can reduce $\mathcal{C}[(\overline{M}\langle T\rangle.Q)\sigma] \rightarrow \mathcal{C}'[Q\sigma]$. Since $P\sigma \approx Q\sigma$ and $\mathcal{C}'$ is closed we have $(\mathcal{C}'[P\sigma], \mathcal{C}'[Q\sigma]) \in \approx \subseteq \mathcal{R}$.
  - The reduction $\rightarrow$ affects $(\overline{M}\langle T\rangle.P)\sigma$ only structurally. That is, we basically have $\mathcal{C}[\mathbf{0}] \rightarrow \mathcal{C}'[\mathbf{0}]$. In this case we apply the same reduction in effect to B and have $(\mathcal{C}'[(\overline{M}\langle T\rangle.P)\sigma], \mathcal{C}'[(\overline{M}\langle T\rangle.Q)\sigma]) \in \mathcal{R}$.
- Obviously, $\mathcal{R}$ is closed under the application of closed evaluation contexts.

This concludes our proof since the definition of $\mathcal{R}$ is symmetric.

2. If $P \approxeq Q$ then $M(x).P \approxeq M(x).Q$ for an arbitrary term M:
   We prove this statement analogously to the previous one: It only differs in the direction of message flow on M. In the corresponding branch of the proof an input of N on M results in $P\{N/x\}$ resp. $Q\{N/x\}$ (note that $\mathcal{C}$ is closed and hence N is closed). Since we have $P\sigma \approx Q\sigma$ in particular for every closing σ with $\sigma(x) = N$ we have that $P\{N/x\} \approxeq Q\{N/x\}$ holds.

3. If $P \approxeq Q$ then $!P \approxeq !Q$:
   A closing substitution σ with $P\sigma \approx Q\sigma$ but $!P\sigma \not\approx !Q\sigma$ contradicts Lemma 2.

4. If $P_1 \approxeq Q_1$ and $P_2 \approxeq Q_2$ then

$$(\text{let } x = D \text{ in } P_1 \text{ else } P_2) \approxeq (\text{let } x = D \text{ in } Q_1 \text{ else } Q_2)$$

   for an arbitrary destructor term D:
   Again, the complete proof is analogous to the one in case 2. Hence we only discuss the reduction of the let-statement here: For all closing substitutions σ for let $x = D$ in P_1 else P_2 and let $x = D$ in Q_1 else Q_2 we have that $D\sigma$ is closed. If we have $D\sigma \Downarrow M$ for a (closed!) term M the let-statement reduces to $P_1\{M/x\}\sigma \approx Q_1\{M/x\}\sigma$ (note that $\sigma(x) = x$ since x is a bound variable) which holds since $P_1 \approxeq Q_1$. Otherwise it reduces to $P_2\sigma \approx Q_2\sigma$ which holds since $P_2 \approxeq Q_2$.

□

Product processes.

In order to argue about concurrent composition, as a technical tool, we will need an extension of the applied pi calculus that supports infinite parallel compositions of processes which are tagged with distinct terms.

Intuitively, the *indexed replication* $\prod_{x \in S} P$ stands for $P\{s_1/x\}|P\{s_2/x\}|\ldots$ when $S = \{s_1, s_2, \ldots\}$. (Like $!P$ stands for $P|P|\ldots$.) We call processes from this extended calculus *product processes*. Note that our main definitions and results are still stated with respect to the original calculus from [22] (and Section 2.1); we only use product processes in some specific situations.

Definition 7 (Product processes)**.** Product processes *are defined by the grammar in Figure 2.1 with the additional construct $\prod_{x\in S} P$ where x is a variable, S a (possibly infinite) set of terms, and P a product process. (We call $\prod_{x\in S} P$ an* indexed replication*.)*

(Note that we consider $\prod_{x\in S}$ to be a binder. I.e., in $\prod_{x\in S} P$, we consider x a bound variable.)

Structural equivalence ($\equiv$) on product processes is defined using the same rules as on processes (see Figure 2.2).

The reduction relation $\to$ on product processes is defined using the same rules as on processes (see Figure 2.3), with the following additional rule (IREPL): If $M \in S$, then $\prod_{x\in S} P \to \left(\prod_{x\in S'} P\right) \mid P\{M/x\}$ with $S' := S \setminus \{M' : M =_{\mathrm{E}} M'\}$. (Essentially S is treated as a set of session ids which contains each sid at most once modulo $=_{\mathrm{E}}$.)

Observational equivalence ($\approx$) on product processes is defined like observational equivalence on processes (Definition 4). In particular, as in Definition 4, in rule (iii) we quantify over evaluation contexts that do not *contain indexed replications.*

Notice that processes are also product processes, and that on processes, the new definitions of $\equiv$, $\to$, and $\approx$ from Definition 7 coincide with the original definitions.

2.2 Useful properties of the pi calculus

In this section, we introduce a number of useful lemmas for the applied pi calculus. These lemmas are useful to derive observational equivalences of processes by step by step rewriting (and for using Proverif as a tool in deriving equivalences that Proverif cannot handle). We believe that they may be useful in other similar situations, too.

Lemma 3. *For natural symbolic models, the following hold:*

(i) If $n \notin fn(M)$, then $n \neq_{\mathrm{E}} M$.

(ii) $n \neq_{\mathrm{E}} m$ for names $n \neq m$.

(iii) $(n, M') \neq_{\mathrm{E}} M$ for all terms M, M' and names $n \notin fn(M)$.

Proof. We show (i):

Fix a term M with $n \notin fn(M)$. Assume for contradiction $n =_{\mathrm{E}} M$. Fix a renaming α such that $\alpha(n) =: n^* \neq n$ and $\alpha(m) = m$ for all $m \in fn(M)$. (This is possible since $n \notin fn(M)$.) Hence $n =_{\mathrm{E}} M = M\alpha =_{\mathrm{E}} n\alpha = n^*$ (since the rules defining $=_{\mathrm{E}}$ are closed under renaming). Thus $n =_{\mathrm{E}} n^* \neq n$. Intuitively, this means that all names are equivalent under $=_{\mathrm{E}}$.

By Definition 5 (viii) (natural symbolic model) there are terms T, T with $T \neq_{\mathrm{E}} T'$. Since the equations in E contain by definition only variables and constructors, all rules defining $=_{\mathrm{E}}$ are closed under substitutions of names by terms. Hence $n =_{\mathrm{E}} n^*$ implies $T =_{\mathrm{E}} T'$.

We have a contradiction, hence (i) follows.

(ii) follows from (i) with $M := m$.

We show (iii):

Assume $(n, M') =_{\mathrm{E}} M$ towards contradiction. Since M does not contain n, $M = M\sigma$ for $\sigma := (n \mapsto n', n' \mapsto n)$ and any $n' \notin fn(M)$. Then $(n', M'\sigma) = (n, M')\sigma =_{\mathrm{E}} M\sigma = M =_{\mathrm{E}} (n, M')$. (Here we use that $=_{\mathrm{E}}$ is closed under renaming which follows from the fact that equations and reduction rules in the symbolic model do not contain names.) By Definition 5 (vi) (natural symbolic model), this implies $n' =_{\mathrm{E}} n$ which contradicts (ii). Thus, the assumption that $(n, M') =_{\mathrm{E}} M$ was wrong. (iii) follows. □

Lemma 4. *Let P, P' be processes. Let D, D' be destructor terms. Let M, M' be terms.*

(i) If $a \notin fn(P)$, then $P \approxeq \nu a.P$.

(ii) If $a \notin fn(M)$, then $\nu a.M(x).P \approxeq M(x).\nu a.P$.

(iii) Assume P is closed and that P does not contain unprotected inputs or outputs. Assume $P \rightarrow P'$, and that for all P'' with $P \rightarrow P''$ we have $P' \approx P''$. Then $P \approx P'$.

(iv) If M, M' are terms with $M =_{\mathrm{E}} M'$, then $P\{M/x\} \approxeq P\{M'/x\}$.

(v) If for all substitutions σ that close D, M we have $D\sigma \Downarrow M\sigma$, and for all M' with $D\sigma \Downarrow M'\sigma$ we have $M\sigma =_{\mathrm{E}} M'\sigma$, then $(let\ x = D\ in\ P\ else\ P') \approxeq P\{M/x\}$.

(vi) If D is closed and there is no M with $D \Downarrow M$, then $(let\ x = D\ in\ P\ else\ P') \approxeq P'$.

(vii) If for all substitution σ that close D, D' there exist M, M' with $D\sigma \Downarrow M\sigma$, $D'\sigma \Downarrow M'\sigma$ and $M\sigma =_{\mathrm{E}} M'\sigma$ then $(if\ D = D'\ then\ P\ else\ P') \approxeq P$

(viii) We have $!P \approx P\,|!P$.

(ix) $\prod_{x \in SID} P \approx \prod_{x \in SID \setminus \{t_1,\dots,t_n\}} P|P\{\frac{t_1}{x}\}|\dots|P\{\frac{t_n}{x}\}$ for $t_1, \dots, t_n \in SID$.

Proof. We show (i): Let $\mathcal{R} := \{(Q, \nu a.Q) : Q\ \textit{a closed process}, a \notin fn(Q)\ \text{a name}\}$ up to structural equivalence. It is easy to see that $\mathcal{R}$ is a bisimulation. Thus $Q \approx \nu a.Q$ for any closed process. This implies that $P\sigma \approx \nu a.(P\sigma) \equiv (\nu a.P)\sigma$ for any closing σ. Hence $P \approxeq \nu a.P$.

We show (ii): Let $\mathcal{R} := \{(E[\nu a.M(x).Q], E[M(x).\nu a.Q])\} \cup \approx$ up to structural equivalence where E ranges over all evaluation contexts, Q over closed processes, a over names, and M over terms with $a \notin fn(M)$. One can check that $\mathcal{R}$ satisfies the conditions for a bisimulation. To show $\nu a.M(x).P \approxeq M(x).\nu a.P$, fix a closing substitution σ. Then $\big((\nu a.M(x).P)\sigma, (M(x).\nu a.P)\sigma\big) \in \mathcal{R}$, thus $(\nu a.M(x).P)\sigma \approx (M(x).\nu a.P)\sigma$. Since this holds for any closing σ, we have $\nu a.M(x).P \approxeq M(x).\nu a.P$ and (ii) follows.

We show (iii): Let $\mathcal{R} := \{(E[P], E[P']) : E\ \text{evaluation context}\} \cup \approx$. (Here P, P' refer to the processes from the statement of the lemma.) We check that $\mathcal{R}$ is a bisimulation. In all the following cases, if $A \approx B$, the statement is immediate. Thus we assume $A \equiv E[P]$, $B \equiv E[P']$ in each case.

- If $(A, B) \in \mathcal{R}$ and $A \downarrow_M$ then there exists a B' with $B \rightarrow^* B'$ and $B' \downarrow_M$: If $A \approx B$, then this is immediate. Thus assume $A \equiv E[P]$, $B \equiv E[P']$. Since P does not contain unprotected outputs, we have that the output on M is in E. Hence $B \equiv E[P'] \downarrow_M$.

- If $(A, B) \in \mathcal{R}$ and $B \downarrow_M$ then there exists an A' with $A \rightarrow^* A'$ and $A' \downarrow_M$: If $A \approx B$, then this is immediate. Thus assume $A \equiv E[P]$, $B \equiv E[P']$. Since $P \rightarrow P'$ we have $A \rightarrow A' := E[P'] \equiv B$. Since $B \downarrow_M$, also $A' \downarrow_M$.

- If $(A, B) \in \mathcal{R}$ and $A \rightarrow A'$ then there exists a B' with $B \rightarrow^* B'$ and $(A', B') \in \mathcal{R}$: If $A \approx B$, then this is immediate. Thus assume $A \equiv E[P]$, $B \equiv E[P']$. Since P does not contain unprotected inputs or outputs, $A' \equiv E'[P]$ for some evaluation context E or $A' \equiv E[P'']$ for some P'' with $P \rightarrow P''$. In the first case, $B \rightarrow B' := E'[P']$ and hence $(A', B') \in \mathcal{R}$. In the second case, $P'' \approx P'$ and thus $A' \approx E[P'] \equiv B =: B'$. Thus $B \rightarrow^* B'$ and $(A', B') \in \mathcal{R}$.

- If $(A, B) \in \mathcal{R}$ and $B \to B'$ then there exists a A' with $A \to^* A'$ and $(A', B') \in \mathcal{R}$: If $A \approx B$, then this is immediate. Thus assume $A \equiv E[P]$, $B \equiv E[P']$. Since $P \to P'$, we have $A \to A'' := E[P'] \equiv B$. Since $B \to B'$, we have $A \to A'' \to A' := B'$. Hence $A \to^* A'$ and $(A', B') \in \mathcal{R}$.

- $\mathcal{R}$ is closed under application of evaluation contexts by construction.

We show (iv): Let $(A, B) \in \mathcal{R}$ iff A results from B by replacing terms M by terms M' with $M =_{\mathrm{E}} M'$. It is easy to check that $\mathcal{R}$ is a bisimulation. Fix a process P, terms M, M' with $M =_{\mathrm{E}} M'$, and σ a substitution mapping variables to ground terms that closes $P\{M/x\}$ and $P\{M'/x\}$. Then $P\{M/x\}\sigma$ results from $P\{M'/x\}\sigma$ by replacing some occurrences of $M'\sigma$ by $M\sigma$. Since $M =_{\mathrm{E}} M'$, we have $M\sigma =_{\mathrm{E}} M'\sigma$. Thus $(P\{M/x\}\sigma, P\{M'/x\}\sigma) \in \mathcal{R}$, hence $P\{M/x\}\sigma \approx P\{M'/x\}\sigma$. Since this holds for any closing σ, $P\{M/x\} \approxeq P\{M'/x\}$.

We show (v): First, assume that $A :=$ (let $x = D$ in P else P') is closed. We have that if $A \to A'$, then $A' \equiv P\{M'/x\}$ for some M' with $D \Downarrow M'$. By (iv) and using that $M =_{\mathrm{E}} M'$ for all M' with $D \Downarrow M'$, this implies $A' \approx P\{M/x\}$. Furthermore A does not contain unprotected inputs or outputs. Thus by (iii), we have $A \approx P\{M/x\}$. From this follows that (let $x = D$ in P else P') $\approxeq P\{M/x\}$ even if (let $x = D$ in P else P') is not closed, analogously to (i).

We show (vi): First, assume that $A :=$ (let $x = D$ in P else P') is closed. We have that if $A \to A'$, then $A' \equiv P'$. Furthermore A does not contain unprotected inputs or outputs. Thus by (iii), we have $A \approx P'$. From this follows that (let $x = D$ in P else P') $\approxeq P'$ even if (let $x = D$ in P else P') is not closed, analogously to (i).

We show (vii): First, assume that $A :=$ (if $D = D'$ then P else P') is closed. We resolve the syntactic sugar for "if" and have $A =$ (let $x = \mathit{equals}(D, D')$ in P else P'). If $A \to A'$, then $A' \equiv P$ ($x \notin \mathit{fv}(P)$). Thus by (iii), we have $A \approx P'$. From this follows that (let $x = D$ in P else P') $\approxeq P'$ even if (let $x = D$ in P else P') is not closed, analogously to (i).

We show (viii): If $!P \to P''$, then $P'' \equiv P|!P$ by definition of $\to$. By (iii) this implies $!P \approx P|!P$.

We show (ix): Given a set $A = \{t_1, \dots, t_k\} \subseteq \mathit{SID}$, we write $\sum_{x \in A} P$ for $P\{t_1/x\}| \dots |P\{t_k/x\}$. Let

$$\mathcal{R} := \left\{ \left(E[\prod_{x \in \mathit{SID} \setminus A \setminus D} P | \sum_{x \in A} P], E[\prod_{x \in \mathit{SID} \setminus B \setminus D} P | \sum_{x \in B} P] \right) \right\}$$

up to structural equivalence where E ranges over evaluation contexts and A, B, D range over subsets of SID with D disjoint of $A \cup B$. One can check that $\mathcal{R}$ satisfies all conditions for being a bisimulation. Since

$$\left(\prod_{x \in \mathit{SID}} P, \prod_{x \in \mathit{SID} \setminus \{t_1, \dots, t_n\}} |P\{t_1/x\}| \dots |P\{t_n/x\} \right) \in \mathcal{R},$$

(ix) follows. □

Lemma 5. *Let C be a 0-1-context whose hole is not under a bang and such that n does not occur in C, Q, or t. Assume that C does not bind any of $fv(Q) \setminus \{x\}$ or $fn(Q)$ over its hole. Then $\nu n.C[\overline{n}\langle t\rangle]|n(x).Q \approxeq C[Q\{t/x\}]$*

Proof. We show the lemma for $\approx$ instead of $\approxeq$, and assuming that $\nu n.C[\overline{n}\langle t\rangle]|n(x).Q$ and $C[Q\{t/x\}]$ are closed and that $fn(Q) \subseteq \{x\}$. The general case then follows by definition of $\approxeq$. We define the relation $\mathcal{R}$: $(A, B) \in \mathcal{R}$ iff $A \approx B$ or there is a name n, a list of names $\tilde{a}$, a term t, a variable x, an integer k, a 0-1-context C not containing n and not having its hole under a bang and not binding $fn(Q)$ over its hole, such that the following holds:

$$A \equiv \nu n\tilde{a}.C[\overline{n}\langle t\rangle]|n(x).Q, \qquad B \equiv \nu n\tilde{a}.C[Q\{t/x\}] \tag{2.1}$$

We check the three conditions for bisimulations (in both directions).

- If $(A, B) \in \mathcal{R}$ and $A \downarrow_M$, then $B \downarrow_M$:

  The case $A \approx B$ is trivial. We thus assume that A, B are as in (Equation 2.1).

  If $\nu n\tilde{a}.C[\overline{n}\langle t\rangle]|n(x).Q \downarrow_M$, then the output on M is in C. ($\overline{n}\langle t\rangle$ cannot be that output, because n is bound.) Hence $\nu n\tilde{a}.C[Q\{t/x\}] \downarrow_M$.

- If $(A, B) \in \mathcal{R}$ and $B \downarrow_M$, then there exists an A' with $A \rightarrow^* A'$ and $A' \downarrow_M$:

  The case $A \approx B$ is trivial. We thus assume that A, B are as in (Equation 2.1).

  If $\nu n\tilde{a}.C[Q\{t/x\}] \downarrow_M$, we distinguish two cases. If the output on M is in C, then $\nu n\tilde{a}.C[\overline{n}\langle t\rangle]|n(x).Q \downarrow_M$. Consider the case that the output on M is in $Q\{t/x\}$. Without loss of generality, we can assume that no name in t is bound in C (otherwise we could move the corresponding restrictions from C into $\nu\tilde{a}$ since C does not bind $fn(Q)$ over its hole). Since the output on M is in $Q\{t/x\}$, C is an evaluation context and thus $\nu n\tilde{a}.C[\overline{n}\langle t\rangle]|n(x).Q \rightarrow \nu n\tilde{a}.C[0]|Q\{t/x\} \downarrow_M$.

- If $(A, B) \in \mathcal{R}$ and $A \rightarrow A'$, then there is a B' with $B \rightarrow^* B'$ and $(A', B') \in \mathcal{R}$:

  The case $A \approx B$ is trivial. We thus assume that A, B are as in (Equation 2.1).

  We distinguish the following cases:

  If the reduction $A \rightarrow A'$ involves only C, then $A' \equiv \nu n\tilde{a}.\tilde{C}[\overline{n}\langle t\rangle\sigma]|n(x).Q$ for some 0-1-context $\tilde{C}$. Here the substitution σ represents possible variable assignments performed over the hole of C (e.g., if $C = \overline{a}\langle T\rangle \mid a(y).\square$, then $\sigma = \{T/y\}$).

  Then $B \rightarrow B' := \nu n\tilde{a}.\tilde{C}[Q\{t/x\}\sigma] = \nu n\tilde{a}.\tilde{C}[Q\{t\sigma/x\}]$ where the last equality uses that $fn(Q) \subseteq x$. Also, $\tilde{C}$ does not have more that one hole (in which case $\tilde{C}$ would not be a zero-or-one-hole context) because the hole in C does not occur under a bang.

  Thus we have $(A', B') \in \mathcal{R}$.

  If the reduction involves $\overline{n}\langle t\rangle$ or $n(x).Q$, then the hole of C is only under restrictions and parallel compositions. We assume without loss of generality that the hole in C is not under any restriction (otherwise we could move the corresponding restrictions into $\nu\tilde{a}$ since C does not bind $fn(Q)$ over its hole). Then $A' \equiv \nu n\tilde{a}.C[0]|Q\{t/x\} \equiv \nu n\tilde{a}.C[Q\{t/x\}] =: B' \equiv B$. Thus $B \rightarrow^* B'$ and $(A', B') \in \mathcal{R}$ (since $A' \approx B'$).

- If $(A, B) \in \mathcal{R}$ and $B \to B'$, then there is an A' with $A \to^* A'$ and $(A', B') \in \mathcal{R}$:

  The case $A \approx B$ is trivial. We thus assume that A, B are as in (Equation 2.1).

  If the reduction $B \to B'$ involves only C, then $B' \equiv \nu n\tilde{a}.\tilde{C}[Q\{t/x\}\sigma] \overset{(*)}{=} \nu n\tilde{a}.\tilde{C}[Q\sigma\{t/x\}]$ for some zero-or-one-hole context $\tilde{C}$. Here the substitution σ represents possible variable assignments performed over the hole of C (e.g., if $C = \overline{a}\langle T\rangle \mid a(y).\square$, then $\sigma = \{T/y\}$). And the equality $(*)$ uses that $fn(Q) \subseteq x$.

  Then $A \to A' := \nu n\tilde{a}.\tilde{C}[\overline{n}\langle t\rangle\sigma]|n(x).Q$. Also, $\tilde{C}$ does not have more that one hole (in which case $\tilde{C}$ would not be a context) because the hole in C does not occur under a bang.

  Thus we have $(A', B') \in \mathcal{R}$.

  If the reduction $B \to B'$ involves $Q\{t/x\}$, then the hole of C is only under restrictions and parallel compositions. We assume without loss of generality that the hole in C is not under any restriction (otherwise we could move the corresponding restrictions into $\nu\tilde{a}$ since C does not bind $fn(Q)$ over its hole). Then $A \to \nu n\tilde{a}.C[0]|Q\{t/x\} \equiv \nu n\tilde{a}.C[Q\{t/x\}] \equiv B \to B' =: A'$. Thus trivially $(A', B') \in \mathcal{R}$ (since $A' = B'$ and thus $A' \approx B'$), and $A \to^* A'$.

- If E is an evaluation context, and $(A, B) \in \mathcal{R}$, then $(E[A], E[B]) \in \mathcal{R}$:

  The case $A \approx B$ is trivial. We thus assume that A, B are as in (Equation 2.1). Then $E[A] \equiv E[\nu n\tilde{a}.C[\overline{n}\langle t\rangle]|n(x).Q] \equiv \nu n\tilde{a}.C[\overline{n}\langle t\rangle]|P|n(x).Q$ for some process P up to renaming of the names $n, \tilde{a}$. And $E[B] \equiv E[\nu n\tilde{a}.C[Q\{t/x\}]] \equiv \nu n\tilde{a}.C[Q\{t/x\}]|P$. Thus (using the context $C|P$ instead of C), we have $(E[A], E[B]) \in \mathcal{R}$.

Thus $\mathcal{R}$ is a bisimulation. Thus $\nu n.C[\overline{n}\langle t\rangle]|n(x).Q \approx \nu n.C[Q\{t/x\}]$ (where n, C, t, x refer to the values from the statement of the lemma). And since n does not occur in C, Q, t, we have $\nu n.C[Q\{t/x\}] \approx C[Q\{t/x\}]$ by Lemma 4 (i). Thus

$$\nu n.C[\overline{n}\langle t\rangle]|n(x).Q \approx C[Q\{t/x\}].$$

□

Lemma 6. *Let C, D be contexts, Q a process, n, m names, t, t' terms, and x a variable. Assume that C, D have no bang over their holes. Assume that $n, m \notin fn(C, D, Q, t, t')$. Assume that C, D do not bind $n, m, fn(Q)$. Assume that $fv(Q) \subseteq \{x\}$. Then $\nu n.(C[!\overline{n}\langle t\rangle] \mid D[n(x).Q]) \approx \nu m.(C[m().Q\{t/x\}] \mid D[\overline{m}\langle t'\rangle])$.*

Proof. We define the relation $\mathcal{R}$ as follows: We have $(A, B) \in \mathcal{R}$ iff $A \approx B$ or there exist 0-1-contexts C, D without a bang over their holes and not binding $n, fn(Q)$, terms t, t', a name $n \notin fn(C, D, Q, t, t')$, a list of names $\tilde{a}$ not containing n, and an integer $i \geq 0$ such that

$$A \equiv \nu n\tilde{a}.(C[\overline{n}\langle t\rangle^i \mid !\overline{n}\langle t\rangle] \mid D[n(x).Q])$$
$$B \equiv \nu n\tilde{a}.(C[n().Q\{t/x\}] \mid D[\overline{n}\langle t'\rangle]) \qquad (2.2)$$

Here $\overline{n}\langle t\rangle^i$ denotes $\overline{n}\langle t\rangle|\dots|\overline{n}\langle t\rangle$ (i copies). Note: Q is the process from the statement of the lemma. (It is intentional that we use n in the definition of B, not m as in the statement of the lemma. We will rename n into m at the end of the proof.)

We show that $\mathcal{R}$ is a bisimulation. In all cases below, the case $A \approx B$ is trivial by the properties of $\approx$, so we assume in each case that A, B are as in (Equation 2.2).

- If $(A,B) \in \mathcal{R}$ and $A \downarrow_M$, then $B \rightarrow^* \downarrow_M$:

  Since n is bound, the output on M is not one of the $\overline{n}\langle t\rangle$ (here we use that $M \neq_{\mathrm{E}} n$ if $n \notin \mathit{fn}(M)$ by Lemma 3 (i)). Hence $C \downarrow_M$ or $D \downarrow_M$. Thus $B \downarrow_M$.

- If $(A,B) \in \mathcal{R}$ and $B \downarrow_M$, then $A \rightarrow^* \downarrow_M$:

  Since n is bound, the output on M is not $\overline{n}\langle t'\rangle$. Hence $C \downarrow_M$ or $D \downarrow_M$. Thus $A \downarrow_M$.

- If $(A,B) \in \mathcal{R}$ and $A \rightarrow A'$, then there is a B' such that $B \rightarrow^* B'$ and $(A',B') \in \mathcal{R}$:

  We distinguish the following cases:

  – $A \rightarrow A'$ is a reduction $!\overline{n}\langle t\rangle \rightarrow \overline{n}\langle t\rangle \mid !\overline{n}\langle t\rangle$: Then $A' \equiv \nu n\tilde{a}.(C[\overline{n}\langle t\rangle^{i+1} \mid !\overline{n}\langle t\rangle] \mid D[n(x).Q])$ and hence $(A',B') \in \mathcal{R}$ for $B' := B$.
  – $A \rightarrow A'$ is a reduction within C, within D, or a communication between C and D (in all cases not involving the argument of C,D): Then $A' \equiv \nu n\tilde{a}.(C'[\overline{n}\langle t\rangle^{i} \mid !\overline{n}\langle t\rangle] \mid D'[n(x).Q])$ for suitable contexts C', D' (satisfying all the conditions required for C, D in the definition of $\mathcal{R}$), and $B \rightarrow B' := \nu n\tilde{a}.(C'[n().Q\{t/x\}] \mid D'[\overline{n}\langle t'\rangle])$. (Note: This uses implicitly that Q has no free variables except x, otherwise Q might change in this reduction.)
  – $A \rightarrow A'$ is a communication between $\overline{n}\langle t\rangle$ and $n(x).Q$:
  Then C and D are evaluation contexts.
  Without loss of generality, we can assume that C, D do not bind any names over their holes: For this, we first rename the bound names in C, D such that they become distinct from all free names (possibly also renaming the names in t in the process, but not in Q since $\mathit{fn}(Q)$ are not bound), and then move the restrictions up into $\nu\tilde{a}$.
  Then $A' \equiv \nu n\tilde{a}.(C[\overline{n}\langle t\rangle^{i-1} \mid !\overline{n}\langle t\rangle] \mid D[Q\{t/x\}])$. Furthermore

$$\begin{aligned} B' := B &\equiv \nu\ \tilde{a}.(C[0] \mid D[\nu n.(n().Q\{t/x\} \mid \overline{n}\langle t'\rangle)]) \\ &\overset{(*)}{\approx} \nu\tilde{a}.(C[0] \mid D[Q\{t/x\}]) \\ &\overset{(**)}{\approx} \nu\tilde{a}.(C[\nu n.(\overline{n}\langle t\rangle^{i-1} \mid !\overline{n}\langle t\rangle)] \mid D[Q\{t/x\}]) \equiv A' \end{aligned}$$

  Here $(*)$ follows from Lemma 5. And $(**)$ uses that $\nu n.(\overline{n}\langle t\rangle^{i-1} \mid !\overline{n}\langle t\rangle) \approx 0$, which can be seen by verifying that

$$\mathcal{R}' := \{(E[\nu n.(\overline{n}\langle t\rangle^{i-1} \mid !\overline{n}\langle t\rangle)], E[0]) : E \text{ evaluation context}\}$$

  is a bisimulation.
  Thus $A' \approx B'$ and hence $(A',B') \in \mathcal{R}$. And $B = B'$ implies $B \rightarrow^* B'$.
  – $A \rightarrow A'$ is a communication between C or D and $\overline{n}\langle t\rangle$ or $n(x).Q$:
  This case does not occur because $n \notin \mathit{fn}(C,D)$.

- If $(A,B) \in \mathcal{R}$ and $B \rightarrow B'$, then there is a A' such that $A \rightarrow^* A'$ and $(A',B') \in \mathcal{R}$:

  We distinguish the following cases:

  – $B \rightarrow B'$ is a reduction within C, within D, or a communication between C and D (in all cases not involving the argument of C,D): Then $B' =$

$\nu n\tilde{a}.(C'[n().Q\{t/x\}] \mid D'[\overline{n}\langle t'\rangle])$ for suitable contexts C', D' (satisfying all the conditions required for C, D in the definition of $\mathcal{R}$), and $A \to A' \equiv \nu n\tilde{a}.(C'[\overline{n}\langle t\rangle^i \mid !\overline{n}\langle t\rangle] \mid D'[n(x).Q])$.

- $B \to B'$ is a communication between $n().Q\{t/x\}$ and $\overline{n}\langle t'\rangle$:
  Then C, D are evaluation contexts. Without loss of generality, we can assume that C, D do not bind any names over their holes (analogous to the corresponding subcase of $A \to A'$ above).
  Then $B' \equiv \nu n\tilde{a}.(C[Q\{t/x\}] \mid D[0])$. Furthermore,

  $$\begin{aligned} A \to^* A' &:= \nu\tilde{a}.(C[\nu n.(\overline{n}\langle t\rangle^i \mid !\overline{n}\langle t\rangle)] \mid D[Q\{t/x\}]) \\ &\overset{(*)}{\approx} \nu\tilde{a}.(C[0] \mid D[Q\{t/x\}]) \equiv \nu\tilde{a}.(C[Q\{t/x\}] \mid D[0]) \overset{(**)}{\approx} B' \end{aligned}$$

  Here $(*)$ uses that $\nu n.(\overline{n}\langle t\rangle^i \mid !\overline{n}\langle t\rangle) \approx 0$ (see the corresponding subcase of $A \to A'$ above). And $(**)$ uses Lemma 4 (i). So $A' \approx B'$, hence $(A', B') \in \mathcal{R}$.
  Hence $A \to^* A'$ and $(A', B') \in \mathcal{R}$.
- $B \to B'$ is a communication between C or D and $\overline{n}\langle t\rangle$ or $n(x).Q$:
  This case does not occur because $n \notin \mathit{fn}(C, D)$.

- If $(A, B) \in \mathcal{R}$ and E is an evaluation context, then $(E[A], E[B]) \in \mathcal{R}$:

  Then $E \equiv \nu\tilde{b}.(\square|P)$ for some names $\tilde{b}$ and some process P. Without loss of generality, n does not occur in $\tilde{b}$ or $\mathit{fn}(P)$ (otherwise we rename n). Thus with $\tilde{a}' := \tilde{a}\tilde{b}$ and $C' := C|P$, we have

  $$\begin{aligned} E[A] &\equiv \nu n\tilde{a}'.(C'[\overline{n}\langle t\rangle^i \mid !\overline{n}\langle t\rangle] \mid D[n(x).Q]) \\ E[B] &\equiv \nu n\tilde{a}'.(C'[n().Q\{t/x\}] \mid D[\overline{n}\langle t'\rangle]) \end{aligned}$$

  Hence $(E[A], E[B]) \in \mathcal{R}$.

Under the conditions of the lemma, we have

$$(\nu n.C[!\overline{n}\langle t\rangle] \mid D[n(x).Q], \nu n.C[n().Q\{t/x\}] \mid D[\overline{n}\langle t'\rangle]) \in \mathcal{R}$$

where C, D, Q, n, t, t', x are as in the statement of the lemma. Since $\mathcal{R}$ is a bisimulation, this implies

$$\begin{aligned} n.C[!\overline{n}\langle t\rangle] \mid D[n(x).Q] &\approx \nu n.C[n().Q\{t/x\}] \mid D[\overline{n}\langle t'\rangle] \\ &\equiv \nu m.C[m().Q\{t/x\}] \mid D[\overline{m}\langle t'\rangle]) \end{aligned}$$

□

Lemma 7. *Let A, B, C be closed processes. If $A \equiv_{\mathrm{E}} B \to C$, then there is a closed process B' such that $A \to B' \equiv_{\mathrm{E}} C$.*

Proof. It is easy to see that $\to$ is the smallest relation satisfying the following rules:

STREQ	If $P \equiv P' \to Q' \equiv Q$, then $P \to Q$
E-REPL	$E[!P] \to E[P \mid !P]$
E-COMM	$E[\overline{C}\langle T\rangle.P \mid C'(x).Q] \ \to\ E[P \mid Q\{T/x\}]$ if $C =_{\mathrm{E}} C'$
E-LET-THEN	$E[\text{let } x = D \text{ in } P \text{ else } Q] \ \to\ E[P\{M/x\}]$ if $D \Downarrow M$
E-LET-ELSE	$E[\text{let } x = D \text{ in } P \text{ else } Q] \ \to\ E[Q]$ if $\nexists M$ s.t. $D \Downarrow M$

Here in all rules E ranges over evaluation contexts with the following property: Let $E[R]$ denote the left hand side of the rule. Then all bound names in $E[R]$ are different from each other and from the free names in $E[R]$. (In a derivation of $\rightarrow$, we can always enforce this latter property by first using STREQ to alpha-rename the left hand side of the reduction.) We say $E[R]$ has no name conflicts.

For stating the next claim, we also need to introduce an asymmetric variant $\mathrel{\mathrlap{\nearrow}\equiv}$ of the structural equivalence $\equiv$. The difference is that in $\equiv$, we are allowed to apply the rule NEW-PAR in both directions, while in $\mathrel{\mathrlap{\nearrow}\equiv}$ we are only allowed to move restrictions up ($P \mid \nu u.Q \mathrel{\mathrlap{\nearrow}\equiv} \nu u.(P \mid Q)$), but not down (not: $\nu u.(P \mid Q) \mathrel{\mathrlap{\nearrow}\equiv} P \mid \nu u.Q$). More formally, $\mathrel{\mathrlap{\nearrow}\equiv}$ is the smallest transitive, reflexive (but not necessarily symmetric) relation closed under α-conversion, and closed under application of evaluation contexts, and satisfying the rules PAR-0, PAR-A, PAR-C, NEW-C, NEW-PAR from Figure 2.2 as well as the reversed rule PAR-0-rev (but not NEW-PAR-rev). (By reversed rule we mean the rules with left hand side and right hand side exchanged. E.g., PAR-0-rev says $P|0 \mathrel{\mathrlap{\nearrow}\equiv} P$. Note that PAR-C-rev and NEW-C-rev are not needed since PAR-C and NEW-C are symmetric. And PAR-A-rev follows from PAR-C and PAR-A via $(P|Q)|R \mathrel{\mathrlap{\nearrow}\equiv} R|(P|Q) \mathrel{\mathrlap{\nearrow}\equiv} (R|P)|Q \mathrel{\mathrlap{\nearrow}\equiv} Q|(R|P) \mathrel{\mathrlap{\nearrow}\equiv} (Q|R)|P \mathrel{\mathrlap{\nearrow}\equiv} P|(Q|R)$.)

Also, we define $\mathrel{\mathrlap{\nearrow}\equiv}_E$ analogously to $\equiv_{\mathrm{E}}$: $\mathrel{\mathrlap{\nearrow}\equiv}_E$ corresponds to a sequence of rewritings using $\mathrel{\mathrlap{\nearrow}\equiv}$ and $=_{\mathrm{E}}$, i.e., $\mathrel{\mathrlap{\nearrow}\equiv}_E := (\mathrel{\mathrlap{\nearrow}\equiv} \cup =_{\mathrm{E}})^*$.

Claim 1. *For closed processes A, B, C, if $A =_{\mathrm{E}} B \mathrel{\mathrlap{\nearrow}\equiv} C$, then there exists a closed process B' such that $A \mathrel{\mathrlap{\nearrow}\equiv} B' =_{\mathrm{E}} C$.*

We show this claim by induction over the derivation of $B \mathrel{\mathrlap{\nearrow}\equiv} C$. We distinguish the following cases:

- *α-conversion:* Then $B = C$ up to α-conversion. Hence $A =_{\mathrm{E}} B$ implies $A =_{\mathrm{E}} C$ since $=_{\mathrm{E}}$ allows α-conversions. Thus $A \mathrel{\mathrlap{\nearrow}\equiv} B^* =_{\mathrm{E}} C$ with $B^* := A$.
- *Closure under evaluation contexts:* Then $B = E[\tilde{B}]$ and $C = E[\tilde{C}]$ for processes $\tilde{B} \mathrel{\mathrlap{\nearrow}\equiv} \tilde{C}$ and an evaluation context E. And the induction hypothesis holds for $\tilde{B} \mathrel{\mathrlap{\nearrow}\equiv} \tilde{C}$. Since $A =_{\mathrm{E}} B = E[\tilde{B}]$, we have that $A = E^*[\tilde{B}^*\sigma]$ for some evaluation context $E^* =_{\mathrm{E}} E$, some process $\tilde{B}^* =_{\mathrm{E}} \tilde{B}$, and a renaming σ that corresponds to the alpha-renaming over the hole of E. Since $\tilde{B}^* =_{\mathrm{E}} \tilde{B}$, the induction hypothesis implies that $\tilde{B}^* \mathrel{\mathrlap{\nearrow}\equiv} \tilde{B}' =_{\mathrm{E}} \tilde{C}$ for some process $\tilde{B}'$. Hence

  $$A = E^*[\tilde{B}^*\sigma] \mathrel{\mathrlap{\nearrow}\equiv} E^*[\tilde{B}'\sigma] =_{\mathrm{E}} E[\tilde{B}'] =_{\mathrm{E}} E[\tilde{C}] = C.$$

  Thus $A \mathrel{\mathrlap{\nearrow}\equiv} B' =_{\mathrm{E}} C$ with $B' := E^*[\tilde{B}'\sigma]$.
- *Reflexivity:* Then $B = C$. Hence $A \mathrel{\mathrlap{\nearrow}\equiv} B^* =_{\mathrm{E}} C$ with $B^* := A$.
- *Transitivity:* Then $B \mathrel{\mathrlap{\nearrow}\equiv} S \mathrel{\mathrlap{\nearrow}\equiv} C$ for some process S. And the induction hypothesis applies to $B \mathrel{\mathrlap{\nearrow}\equiv} S$ and $S \mathrel{\mathrlap{\nearrow}\equiv} C$. Since $A =_{\mathrm{E}} B \mathrel{\mathrlap{\nearrow}\equiv} S$, by induction hypothesis, there is a process B' with $A \mathrel{\mathrlap{\nearrow}\equiv} B' =_{\mathrm{E}} S$. Since $B' =_{\mathrm{E}} S \mathrel{\mathrlap{\nearrow}\equiv} C$, by induction hypothesis there is a process S^* with $B' \mathrel{\mathrlap{\nearrow}\equiv} S^* =_{\mathrm{E}} C$. Thus $A \mathrel{\mathrlap{\nearrow}\equiv} S^* =_{\mathrm{E}} C$, and the claim follows with $B^* := S^*$.
- *PAR-0*: In this case, $C = B|0$ and $A =_{\mathrm{E}} B$. Hence $A \mathrel{\mathrlap{\nearrow}\equiv} B^* =_{\mathrm{E}} C$ with $B^* := A|0$.
- *PAR-0-rev*: In this case, $B = C|0$ and $A =_{\mathrm{E}} B$. Hence $A = B^*|0$ for some process $B^* =_{\mathrm{E}} C$. Then $A \mathrel{\mathrlap{\nearrow}\equiv} B^* =_{\mathrm{E}} C$.
- *PAR-A*: In this case, $B = B_1|(B_2|B_3)$ and $C = (B_1|B_2)|B_3$. Since $A =_{\mathrm{E}} B$, $A = A_1|(A_2|A_3)$ for some processes A_i with $A_i =_{\mathrm{E}} B_i$, $i = 1, 2, 3$. Then with $B^* := (A_1|A_2)|A_3$, we have $A \mathrel{\mathrlap{\nearrow}\equiv} B^* =_{\mathrm{E}} C$.

- *PAR-C, PAR-C*: Analogous to PAR-A.
- *NEW-C*: In this case, $B = \nu nm.\hat{B}$ and $C = \nu mn.\hat{B}$ for some names n, m and a process $\hat{B}$. Since $A =_{\mathrm{E}} B$, we have that $A = \nu ab.\hat{A}$ for some names a, b and a process $\hat{A}$. (Not necessarily $ab = nm$, because $=_{\mathrm{E}}$ allows α-conversion.) Thus $\nu ab.\hat{A} =_{\mathrm{E}} \nu nm.\hat{B}$. This implies $\nu ba.\hat{A} =_{\mathrm{E}} \nu mn.\hat{B}$ (by induction over the derivation of $\nu ab.\hat{A} =_{\mathrm{E}} \nu nm.\hat{B}$). Hence with $B^* := \nu ba.\hat{A}$, we have that $A \not\equiv B^* =_{\mathrm{E}} C$.
- *NEW-PAR*: Then $B = B_1|\nu n.B_2$ and $C = \nu n.(B_1|B_2)$ with $n \notin \mathit{fn}(B_1)$. Since $A =_{\mathrm{E}} B$, we have $A = A_1|\nu a.A_2$ for some name a and processes A_1, A_2 with $A_1 =_{\mathrm{E}} B_1$ and $\nu a.A_2 =_{\mathrm{E}} \nu n.B_2$. (Not necessarily $a = n$, because $=_{\mathrm{E}}$ allows α-conversion.) Let m be a fresh name, i.e., $m \notin \mathit{fn}(A_1, A_2, B_1, B_2)$. Let $B^* := \nu m.(A_1|A_2\{m/a\})$. Since $\nu n.B_2 =_{\mathrm{E}} \nu a.A_2$ and $m \notin \mathit{fn}(A_2, B_2)$, we have $\nu m.B_2\{m/n\} =_{\mathrm{E}} \nu m.A_2\{m/a\}$. Hence $\nu m.(A_1|B_2\{m/n\}) =_{\mathrm{E}} \nu m.(A_1|A_2\{m/a\})$. And using $A_1 =_{\mathrm{E}} B_1$, we get $\nu m.(B_1|B_2\{m/n\}) =_{\mathrm{E}} \nu m.(A_1|A_2\{m/a\}) = B^*$. Furthermore $C = \nu n.(B_1|B_2) =_{\mathrm{E}} \nu m.(B_1|B_2\{m/n\})$ since $n, m \notin \mathit{fn}(B_1)$, $m \notin \mathit{fn}(B_2)$. Thus $B^* =_{\mathrm{E}} C$. And $A = A_1|\nu a.A_2 \not\equiv A_1|\nu m.A_2\{m/a\} \not\equiv \nu m.(A_1|A_2\{m/a\}) = B^*$. Thus B^* is a process with $A \not\equiv B^* =_{\mathrm{E}} C$.

This shows Claim 1.

Claim 2. *If $A \not\equiv_E B$, then there exists an S such that $A \not\equiv S =_{\mathrm{E}} B$.*

This follows directly from Claim 1.

Claim 3. *If B, C are closed processes and $B \to C$ (derived using the rules listed at the beginning of this proof), then for any closed A with $A \equiv_{\mathrm{E}} B$ there exists a closed B' with $A \to B' \equiv_{\mathrm{E}} C$.*

This claim will then immediately prove the lemma. We show the claim by induction over the derivation of $B \to C$. We distinguish the following rule applications:

- STREQ: Then $B \equiv \tilde{B} \to \tilde{C} \equiv C$ for some $\tilde{B}, \tilde{C}$, and the induction hypothesis holds for $\tilde{B} \to \tilde{C}$. Since $A \equiv_{\mathrm{E}} B \equiv \tilde{B}$, the induction hypothesis implies that $A \to B' \equiv_{\mathrm{E}} \tilde{C}$ for some closed B'. Since $\tilde{C} \equiv C$, we have $A \to B' \equiv_{\mathrm{E}} C$.

- E-REPL: Then $B = E[!\tilde{B}]$ and $C = E[\tilde{B} \mid !\tilde{B}]$ where E is an evaluation context and $E[!\tilde{B}]$ has no name conflicts. We have $A \equiv_{\mathrm{E}} E[!\tilde{B}]$. From this it follows that $A \not\equiv_E E'[!\tilde{B}]$ where E' results from E by moving all unprotected restrictions to the top (no names in $\tilde{B}$ need to be renamed because $E[!\tilde{B}]$ has no name conflicts). By Claim 2, this implies that $A \not\equiv S =_{\mathrm{E}} E'[!\tilde{B}]$ for some S. Hence $S = E''[!\tilde{B}'\sigma]$ where $E'' =_{\mathrm{E}} E'$ and $\tilde{B}' =_{\mathrm{E}} \tilde{B}$ and where σ is a renaming that corresponds to the alpha-conversions between E' and E'' over the hole. Thus $A \not\equiv S \to E''[(\tilde{B}|!\tilde{B})\sigma] =_{\mathrm{E}} E'[\tilde{B} \mid !\tilde{B}] \equiv E[\tilde{B} \mid !\tilde{B}] = C$ and hence $A \to B' \equiv_{\mathrm{E}} C$ with $B' := E''[(\tilde{B}|!\tilde{B})\sigma]$.

- E-COMM: Then $B = E[\overline{M}\langle T\rangle.P \mid N(x).Q]$ and $C = E[P \mid Q\{T/x\}]$ where E is an evaluation context, $M =_{\mathrm{E}} N$, and B has no name conflicts. As in the E-REPL case, we have $A \not\equiv_E E'[\overline{M}\langle T\rangle.P \mid N(x).Q]$ where E' results from E by moving all unprotected restrictions to the top. By Claim 2, this implies that $A \not\equiv S =_{\mathrm{E}} E'[\overline{M}\langle T\rangle.P \mid N(x).Q]$ for some S. Hence $S = E''[(\overline{M'}\langle T'\rangle.P' \mid$

$N'(x).Q')\sigma]$ where $E'' =_{\mathrm{E}} E'$, $M' =_{\mathrm{E}} M$, $T' =_{\mathrm{E}} T$, $P' =_{\mathrm{E}} P$, $N' =_{\mathrm{E}} N$, $Q' =_{\mathrm{E}} Q$, and σ is as in the case of E-REPL. Then

$$\begin{aligned} A \not\equiv S \to E''[P' \mid Q'\{T'/x\}\sigma] &=_{\mathrm{E}} E'[P \mid Q\{T'/x\}] \\ &\overset{(*)}{=}_{\mathrm{E}} E'[P \mid Q\{T/x\}] \\ &\equiv E[P \mid Q\{T/x\}] = C. \end{aligned}$$

(Note that $(*)$ also uses the fact that $=_{\mathrm{E}}$ may also rewrite terms that are subterms of destructor terms; this is needed if x occurs in a destructor term in Q.)

Hence $A \to B' \equiv_{\mathrm{E}} C$ for $B' := E''[P' \mid Q'\{T'/x\}\sigma]$.

- E-LET-THEN: Then $B = E[\text{let } x = D \text{ in } P \text{ else } Q]$ and $C = E[P\{M/x\}]$ where E is an evaluation context, $D \Downarrow M$, and B has no name conflicts. As in the E-REPL case, we have $A \not\equiv_E E'[\text{let } x = D \text{ in } P \text{ else } Q]$ where E' results from E by moving all unrestricted restrictions to the top. By Claim 2, this implies that $A \not\equiv S =_{\mathrm{E}} E'[\text{let } x = D \text{ in } P \text{ else } Q]$ for some S. Hence $S = E''[(\text{let } x = D' \text{ in } P' \text{ else } Q')\sigma]$ where $E'' =_{\mathrm{E}} E'$, $D' =_{\mathrm{E}} D$, $P' =_{\mathrm{E}} P$, $Q' =_{\mathrm{E}} Q$, and σ is as in the case of E-REPL. Then $D' =_{\mathrm{E}} D$ and $DM \Downarrow$ imply $D'M \Downarrow'$ for some $M' =_{\mathrm{E}} M$. Hence $(\text{let } x = D' \text{ in } P' \text{ else } Q') \to P'\{M'/x\}$. Then

$$\begin{aligned} A \not\equiv S \to E''[P'\{M'/x\}\sigma] &=_{\mathrm{E}} E'[P\{M'/x\}] \\ &\overset{(*)}{=}_{\mathrm{E}} E'[P\{M/x\}] \\ &\equiv E[P\{M/x\}] = C. \end{aligned}$$

(Here $(*)$ again uses that $=_{\mathrm{E}}$ rewrites destructor terms, see the case E-COMM.)

Hence $A \to B' \equiv_{\mathrm{E}} C$ for $B' := E''[P'\{M'/x\}\sigma]$.

- E-LET-ELSE: Then $B = E[\text{let } x = D \text{ in } P \text{ else } Q]$ and $C = E[Q]$ where E is an evaluation context, $\forall M.\ D \not\Downarrow M$, and B has no name conflicts. As in the E-REPL case, we have $A \not\equiv_E E'[\text{let } x = D \text{ in } P \text{ else } Q]$ where E' results from E by moving all unrestricted restrictions to the top. By Claim 2, this implies that $A \not\equiv S =_{\mathrm{E}} E'[\text{let } x = D \text{ in } P \text{ else } Q]$ for some S. Hence $S = E''[(\text{let } x = D' \text{ in } P' \text{ else } Q')\sigma]$ where $E'' =_{\mathrm{E}} E'$, $D' =_{\mathrm{E}} D$, $P' =_{\mathrm{E}} P$, $Q' =_{\mathrm{E}} Q$, and σ is as in the case of E-REPL. Since $D' =_{\mathrm{E}} D$ and $\forall M.\ D \not\Downarrow M$, we have $\forall M.\ D' \not\Downarrow M$. Hence $(\text{let } x = D' \text{ in } P' \text{ else } Q') \to Q'$. Then

$$A \not\equiv S \to E''[Q'\sigma] =_{\mathrm{E}} E'[Q] \equiv E[Q] = C.$$

Hence $A \to B' \equiv_{\mathrm{E}} C$ for $B' := E''[Q'\sigma]$.

This shows Claim 3. And from that claim the lemma follows. □

2.2.1 Relating events and observational equivalence

For stating Lemma 9 below, we will need processes containing events. The variant of the applied pi calculus presented in Section 2.1 (which is used by Proverif for observational equivalence proofs) does not support events. When using Proverif for showing trace properties defined in terms of events, a different variant of the applied pi calculus is used [20]. We will call processes in that calculus *event processes*. Syntactically, event processes differ from processes as in Figure 2.1 only by an additional construct $\mathit{event}\ f(t_1,\dots,t_n).P$ which means that the event f is raised, with arguments $t_1,\dots,t_n$ (these are normal terms), and then the event process P is executed.

The semantics of event processes are formulated in [20] in a different way from the semantics used here. Fortunately, we will be able to encapsulate everything that we need to know about that semantics in Lemma 8 below, so we do not need to repeat those semantics here.

Instead, we extend the definition of the internal reduction relation $\to$ to event processes. $\to$ is defined as in Definition 3, except that we add the following rule:

$$\text{EVENT:}\quad \mathit{event}\ f(t_1,\dots,t_n).P \to P$$

The semantics defined by $\to$ will be related to those from [20] by Lemma 8 below.

Finally, [20] defines the concept of a *trace property*. We will only need trace properties of a specific form, namely

$$\mathit{end}(x) \Rightarrow \mathit{start}(x) \lor x = t_1 \lor \dots \lor x = t_n$$

Intuitively, an event process P *satisfies a trace property* $\mathit{end}(x) \Rightarrow \mathit{start}(x) \lor x = t_1 \lor \dots \lor x = t_n$ if in any execution $P|R \to P_1 \to \dots \to P_n$, we have that if one of the transitions raises the event $\mathit{end}(t)$, then $t \in \{t_1,\dots,t_n\}$ and in the same trace, the event $\mathit{start}(t)$ is also raised (for any adversarial R not containing events).

Formally, satisfying a trace property is defined with respect to the semantics from [20].[6] Instead of giving those semantics here, we present the following lemma which summarizes seven fact about that definition. We will not use any other facts. The facts can be verified by inspecting the semantics and definitions from [20].

Lemma 8. *Let $t_1,\dots,t_n$ be terms. Let $\wp$ stand for the trace property $\mathit{start}(x) \Rightarrow \mathit{end}(x) \lor x = t_1 \lor \dots \lor x = t_n$. Let P be an event process.*

(i) If $P \equiv P'$ and P satisfies $\wp$, then P' satisfies $\wp$.

(ii) Assume $P \to P'$ and P satisfies $\wp$ and the reduction $P \to P'$ does not use the EVENT rule. Then P' satisfies $\wp$.

(iii) Let t be a closed term. Assume $P = C[\mathit{event}\ \mathit{start}(t).Q]$ where C is an event context not binding $\mathit{fn}(t)$ over its hole. Assume that P satisfies $\wp$. Then $P' := C[Q]$ satisfies $\wp \lor x = t$.

(iv) Assume $P = C[\mathit{event}\ \mathit{end}(t).Q]$ where C is an event context. Assume that P satisfies $\wp$. Then $P' := C[Q]$ satisfies $\wp$.

(v) Assume P satisfies $\wp$ and E is an evaluation context (not containing events) and E does not bind $\mathit{fn}(t_1,\dots,t_n)$ over its hole. Then $E[P]$ satisfies $\wp$.

(vi) Assume E is an evaluation event context that does not bind any names over its hole. Assume $P = E[\mathit{event}\ \mathit{end}(t).Q]$. Assume that P satisfies $\wp$. Then $t =_{\mathrm{E}} t_i$ for some i.

[6] Strictly speaking, the semantics described in [20] does not allow expressions of the form $x = t_i$ in trace properties. Such expressions are, however, supported by Proverif. Also, [20, footnote 3 in the full version] explains how to encode such equality tests in the trace properties supported by [20]. In their notation, our trace property becomes the somewhat less readable trace property: $\mathit{end}(x) \Rightarrow (\mathit{end}(x) \rightsquigarrow \mathit{start}(x)) \lor (\mathit{end}(t_1) \rightsquigarrow \mathit{true}) \lor \dots \lor (\mathit{end}(t_n) \rightsquigarrow \mathit{true})$.

Also, the semantics described [20] do not support equations (i.e., $t =_{\mathrm{E}} t'$ iff $t = t'$ in their semantics). However, Proverif supports these, so we assume the intended semantics of Proverif is that of [20] with the natural extension of equality tests to equality modulo $=_{\mathrm{E}}$.

(vii) If $\nu a.P$ satisfies $\wp$, then P satisfies $\wp$.

We explain the intuitive reason for each fact:

(i) Structurally equivalent processes behave identically and thus raise the same events.

(ii) If $P \to P'$ without raising an event, then for any event trace that P' may produce, P may produce the same by first reducing to P'.

(iii) P' has the same event traces as P, except that some $\mathit{start}(t)$-events are removed. If P' does not satisfy $\wp \vee x = t$, then there must be an event $\mathit{end}(t')$ with $t \neq t'$ that is not preceded by a $\mathit{start}(t')$-event. But then also in a trace of P, there would be an $\mathit{end}(t')$-event not preceded by $\mathit{start}(t')$ (since the traces only differ in their $\mathit{start}(t)$-events and $\mathit{start}(t) \neq \mathit{start}(t')$).

(iv) P' has the same event traces as P, except that various $\mathit{end}(\cdot)$-events are removed. (Since t is not necessarily closed, $\mathit{end}(t)$ may be instantiated to different $\mathit{end}(\cdot)$-events.) If a trace of P' does not satisfy $\wp$, this means there was an $\mathit{end}(t')$-event not preceded by a $\mathit{start}(t')$ event. Then also in P the corresponding $\mathit{end}(t')$-event is not preceded by a $\mathit{start}(t')$-event, as P has the same $\mathit{start}(\cdot)$-events, and more $\mathit{end}(\cdot)$-events.

(v) The semantics of satisfying trace properties are defined with respect to P running in parallel with an adversary R not containing events. Thus the case of an evaluation context running with P is already covered. (It is important that E does not bind $\mathit{fn}(t_1,\dots,t_n)$ because otherwise the terms $t_1,\dots,t_n$ occurring in the process would be considered different from those in $\wp$.)

(vi) There is a trace of P that consists only of an $\mathit{end}(t)$-event. That trace does not satisfy $\mathit{end}(t) \Rightarrow \mathit{start}(t)$. Thus it satisfies $\wp$ only if $\wp$ contains $x = t$ as one of its clauses.

(vii) $\nu a.P$ has the same traces as P, except that occurrences of a in the P-traces are replaced by a fresh restricted name a'. Thus, if P does not satisfy $\wp$, then there is a trace containing an $\mathit{end}(t)$-event without preceding $\mathit{start}(t)$-event such that $t \notin \{t_1,\dots,t_n\}$. In the corresponding $\nu a.P$-trace, we have an $\mathit{end}(t\{a'/a\})$-event without preceding $\mathit{start}(t\{a'/a\})$-event. Since $t \notin \{t_1,\dots,t_n\}$ and a is fresh, also $t\{a'/a\} \notin \{t_1,\dots,t_n\}$. Hence the $\nu a.P$-trace does not satisfy $\wp$, either.

Lemma 9. *Let s be a name. Let P be a process containing s only in constructs of the form $(!\overline{(s,t)}\langle t'\rangle)|P'$ and $(s,t)().P'$ (for arbitrary and possibly different t, t', P').*

Let $\mathit{plain}^s(P)$ denote the process resulting from P by replacing all occurrences $!\overline{(s,t)}\langle t'\rangle|P'$ and $(s,t)().P'$ by P'.

Let $\mathit{ev}^s(P)$ denote the process resulting from P by replacing all occurrences $!\overline{(s,t)}\langle t'\rangle|P'$ by event $\mathit{start}(t).P'$ and $(s,t)().P'$ by event $\mathit{end}(t).P'$.

Assume that $\mathit{ev}^s(P)$ satisfies the trace property $\mathit{end}(x) \Rightarrow \mathit{start}(x)$.

Then $\mathit{plain}^s(P) \approx \nu s.P$.

Proof. We call a process P s-well-formed if it contains s only in constructs of the form $!\overline{(s,t)}\langle t'\rangle|P'$ and $(s,t)().P'$ (for arbitrary and possibly different t, t', P'). Given a multiset $T = \{t_1 \mapsto t_1', \dots, t_n \mapsto t_n'\}$ with t_i, t_i' terms, we call an event-process P T-good if P satisfies the trace property $\mathit{end}(x) \Rightarrow \mathit{start}(x) \lor x = t_1 \lor \dots \lor x = t_n$.

For example, the process P from the statement of the lemma is s-well-formed, and $\mathit{ev}^s(P)$ is $\varnothing$-good.

We define the following relation $\mathcal{R}$ (up to structural equivalence): $\mathcal{R} :=$

$$\Big\{\Big(\nu\underline{a}.\mathit{plain}^s(P), \nu\underline{a}s.(P \mid !\overline{(s,t_1)}\langle t_1'\rangle \mid \cdots \mid !\overline{(s,t_n)}\langle t_n'\rangle \mid \overline{(s,u_1)}\langle u_1'\rangle \mid \cdots \mid \overline{(s,u_m)}\langle u_m'\rangle\Big)$$
$$P \text{ } s\text{-well-formed}, \quad s, \underline{a} \text{ distinct names}, \quad \mathit{ev}^s(P) \text{ is } \{t_1, \dots, t_n\}\text{-good}\Big\}$$

Here $P, n, m, t_i, t_i', u_i, u_i', s, \underline{a}$ refer to arbitrary values, not only to the values P, s from the statement of the lemma.

We write short $\mathit{syncout}^s(\{t_1 \mapsto t_1', \dots, t_n \mapsto t_n'\}; \{u_1 \mapsto u_1', \dots, u_n \mapsto u_n'\})$ for $!\overline{(s,t_1)}\langle t_1'\rangle \mid \cdots \mid !\overline{(s,t_n)}\langle t_n'\rangle \mid \overline{(s,u_1)}\langle u_1'\rangle \mid \cdots \mid \overline{(s,u_m)}\langle u_m'\rangle$.

We now show that $\mathcal{R}$ is a bisimulation:

- If $(A, B) \in \mathcal{R}$, and $A \downarrow_M$, then $B \downarrow_M$:

  Then $A = \nu\underline{a}.\mathit{plain}^s(P)$. Hence $\mathit{plain}^s(P) \downarrow_M$ and $\underline{a} \notin \mathit{fn}(M)$. Also, $s \notin \mathit{fn}(\mathit{plain}^s(P))$, so $s \notin \mathit{fn}(M)$. By definition of $\mathit{plain}^s(\cdot)$, $\mathit{plain}^s(P) \downarrow_M$ implies $P \downarrow_M$. Since $\underline{a}, s \notin \mathit{fn}(M)$, it follows $B = \nu\underline{a}s.(P|\dots) \downarrow_M$.

- If $(A, B) \in \mathcal{R}$, and $B \downarrow_M$, then $A \downarrow_M$:

  Then $B = \nu\underline{a}s.(P|\mathit{syncout}^s(T;U))$. Thus $\underline{a}, s \notin \mathit{fn}(M)$ and $P|\mathit{syncout}^s(T;U) \downarrow_M$. Since all channels in $\mathit{syncout}^s(T;U)$ are of the form $(s,\cdot)$, we have $\mathit{syncout}^s(T;U) \not\downarrow_M$.[7] Hence $P \downarrow_M$. By definition of $\mathit{plain}^s(P)$ and since M does not contain s, this implies $\mathit{plain}^s(P) \downarrow_M$. Hence $A = \nu\underline{a}.\mathit{plain}^s(P) \downarrow_M$.

- If $(A, B) \in \mathcal{R}$, and $A \to A'$, then there exists a B' with $B \to^* B'$ and $(A', B') \in \mathcal{R}$:

  Then $A \equiv \nu\underline{a}.\mathit{plain}^s(P)$ and $B \equiv \nu\underline{a}s.(P|\mathit{syncout}^s(T;U))$. We call an event process *name-reduced*, if it does not contain unprotected restrictions.

  Without loss of generality, assume that P (and hence also $\mathit{ev}^s(P)$) is name-reduced (otherwise we could move the superfluous restrictions into the $\nu\underline{a}$).

  Let $\underline{a}_0 := \underline{a}$ and $P_0 := P$ and $T_0 := T$. We first construct a sequence $P_1, \dots, P_k$ of processes and a sequence of lists of names $\underline{a}_1, \dots, \underline{a}_k$, and a sequence of sets $T_1, \dots, T_k$ such that P_k does not contain unprotected inputs $(s,\cdot)().Q$ or unprotected outputs $!\overline{(s,\cdot)}\langle\cdot\rangle$, and for all $i = 0, \dots, k$ we have:

  (a) $\nu s.(P|\mathit{syncout}^s(T;U)) \to^* \nu\underline{a}_i s.(P_i|\mathit{syncout}^s(T_i;U))$, and
  (b) $\mathit{ev}^s(P_i)$ is T_i-good, and
  (c) $\mathit{plain}^s(P) \equiv \nu\underline{a}_i.\mathit{plain}^s(P_i)$.
  (d) P_i is s-well-formed.

  For $i = 0$, these conditions are trivially satisfied. When constructing P_i for $i > 0$, we already have a process P_{i-1} satisfying these conditions. We distinguish three cases:

[7] Here we implicitly use the fact that $(s,\cdot) \neq_{\mathrm{E}} M$ for any M not containing s (Lemma 3 (iii)).

- If P_{i-1} does not contain unprotected inputs $(s,\cdot)()$, we are done ($k := i-1$).
- If P_{i-1} does contain an unprotected input $(s,t)()$ that is not part of a subterm of the form $!\overline{(s,\cdot)}\langle\cdot\rangle|Q$, then we can write P_{i-1} as $P_{i-1} = \nu\underline{b}.E[(s,t)().P']$ for some names $\underline{b}$ and some evaluation context E that has no restrictions over its hole. Since $(s,t)()$ is not part of a subterm of the form $!\overline{(s,\cdot)}\langle\cdot\rangle|Q$, $\mathit{ev}^s(E)$ is an evaluation context ($!\overline{(s,\cdot)}\langle\cdot\rangle|Q$ would have translated to $\mathit{event\ start}(\cdot).Q$). Without loss of generality, $\underline{b} \cap \mathit{fn}(T_{i-1},U) = \varnothing$.

  Since $\mathit{ev}^s(P_{i-1}) \equiv \nu\underline{b}.\mathit{ev}^s(E)[\mathit{event\ end}(t).\mathit{ev}^s(P')]$ is T_{i-1}-good by (b), Lemma 8 (vii) implies that $\mathit{ev}^s(E)[\mathit{event\ end}(t).\mathit{ev}^s(P')]$ is T_{i-1}-good. Since E does not bind any names over its hole, Lemma 8 (vi) implies that $t =_{\mathrm{E}} t^*$ for some $t^* \in T_{i-1}$. Thus $P_{i-1}|\mathit{syncout}^s(T_{i-1};U) \equiv (\nu\underline{b}.E[(s,t)().P'])|\mathit{syncout}(T_{i-1};U) \to^* (\nu\underline{b}.E[P'])|\mathit{syncout}(T_{i-1};U)$. Since without loss of generality, $\underline{b} \cap \mathit{fn}(T_{i-1},U) = \varnothing$, $(\nu\underline{b}.E[P'])|\mathit{syncout}(T_{i-1};U) \equiv \nu\underline{b}.P_i|\mathit{syncout}(T_{i-1};U)$ with $P_i := E[P']$. Hence $\nu s.P|\mathit{syncout}^s(T;U)\overset{(\mathrm{a})}{\to}{}^*\nu\underline{a}_{i-1}s.(P_{i-1}|\mathit{syncout}^s(T_{i-1};U)) \to^* \nu\underline{a}_{i-1}s\underline{b}.P_i|\mathit{syncout}^s(T_{i-1};U) \equiv \nu\underline{a}_i s.P_i|\mathit{syncout}^s(T_i;U)$ with $T_i := T_{i-1}$ and $\underline{a}_i := \underline{a}_{i-1}\underline{b}$. Thus (a) is satisfied by $P_i, \underline{a}_i, T_i$.

  Since $\mathit{ev}^s(P_{i-1}) \equiv \nu\underline{b}.\mathit{ev}^s(E)[\mathit{event\ end}(t).\mathit{ev}^s(P')]$ is T_{i-1}-good by (b) and thus T_i-good, we have by Lemma 8 (vii) that $\mathit{ev}^s(E)[\mathit{event\ end}(t).\mathit{ev}^s(P')]$ is T_i-good. Since E does not bind names over its hole, neither does $\mathit{ev}^s(E)$. Thus by Lemma 8 (iv), $\mathit{ev}^s(E)[\mathit{ev}^s(P')] = \mathit{ev}^s(P_i)$ is T_i-good. Thus (b) is satisfied by $P_i, \underline{a}_i, T_i$.

  Since $P_{i-1} = \nu\underline{b}.E[(s,t)().P']$ is s-well-formed by (d), so is $P_i = E[P']$. Thus (d) is satisfied by $P_i, \underline{a}_i, T_i$.

  Finally, $\mathit{plain}^s(P_{i-1}) = \nu\underline{b}.\mathit{plain}^s(E)[\mathit{plain}^s(P')] = \nu\underline{b}.\mathit{plain}^s(P_i)$. Since by (c) we have that $\mathit{plain}^s(P) \equiv \nu\underline{a}_{i-1}.\mathit{plain}^s(P_{i-1})$, we have $\mathit{plain}^s(P) \equiv \nu\underline{a}_i.\mathit{plain}^s(P_i)$. Thus (c) is satisfied by $P_i, \underline{a}_i, T_i$.
- If P_{i-1} contains an unprotected output $!\overline{(s,t)}\langle t'\rangle$ that is not part of a subterm of the form $!\overline{(s,\cdot)}\langle\cdot\rangle|Q$, then we can write P_{i-1} as $P_{i-1} = \nu\underline{b}.E[\overline{(s,t)}\langle t'\rangle|P']$ for some names $\underline{b}$ and some evaluation context E that has no restrictions over its hole. Since $\overline{(s,t)}\langle t'\rangle$ is not part of a subterm of the form $!\overline{(s,\cdot)}\langle\cdot\rangle|Q$, $\mathit{ev}^s(E)$ is an evaluation context ($!\overline{(s,\cdot)}\langle\cdot\rangle|Q$ would have translated to $\mathit{event\ start}(\cdot).Q$). Without loss of generality, $\underline{b} \cap \mathit{fn}(T_{i-1},U) = \varnothing$.

  We have $P_{i-1}|\mathit{syncout}^s(T_{i-1};U) \equiv (\nu\underline{b}.E[!\overline{(s,t)}\langle t'\rangle|P'])|\mathit{syncout}(T_{i-1};U) \overset{(*)}{\equiv} \nu\underline{b}.(E[!\overline{(s,t)}\langle t'\rangle|P']|\mathit{syncout}(T_{i-1};U)) \equiv \nu\underline{b}.(E[P']|\mathit{syncout}(T_i;U))$ with $T_i := T_{i-1} \cup \{t \mapsto t'\}$. Here $(*)$ uses that $\underline{b} \cap \mathit{fn}(T_{i-1},U) = \varnothing$. Hence $\nu s.P| \mathit{syncout}^s(T;U)\overset{(\mathrm{a})}{\to}{}^*\nu\underline{a}_{i-1}s.(P_{i-1}|\mathit{syncout}^s(T_{i-1};U)) \to^* \nu\underline{a}_{i-1}s\underline{b}.(E[P']| \mathit{syncout}^s(T_i;U)) \equiv \nu\underline{a}_i s.(P_i|\mathit{syncout}^s(T_i;U))$ with $P_i := E[P']$ and $\underline{a}_i := \underline{a}_{i-1}\underline{b}$ (remember that $T_i = T_{i-1} \cup \{t \mapsto t'\}$. Thus (a) is satisfied by $P_i, \underline{a}_i, T_i$.

  Since $\mathit{ev}^s(P_{i-1}) \equiv \nu\underline{b}.\mathit{ev}^s(E)[\mathit{event\ start}(t).\mathit{ev}^s(P')]$ is T_{i-1}-good by (b), we have by Lemma 8 (vii) that $\mathit{ev}^s(E)[\mathit{event\ start}(t).\mathit{ev}^s(P')]$ is T_{i-1}-good. Since E does not bind names over its hole, neither does $\mathit{ev}^s(E)$. Thus by

Lemma 8 (iii), $ev^s(E)[ev^s(P')] = ev^s(P_i)$ is T_i-good. Thus (b) is satisfied by $P_i, \underline{a}_i, T_i$.

Since $P_{i-1} = \nu\underline{b}.E[\overline{(s,t)}\langle t'\rangle.P']$ is s-well-formed by (d), so is $P_i = E[P']$. Thus (d) is satisfied by $P_i, \underline{a}_i, T_i$.

That (c) is satisfied by $P_i, \underline{a}_i, T_i$ is shown as in the previous case.

Note that in the last two cases, the size of P_i is smaller than that of P_{i-1}, so we eventually reach the first case. Hence the construction terminates and we get a process P_k that satisfies (a)–(d) and that does not contain unprotected inputs $(s,\cdot)()$ or unprotected outputs $!\overline{(s,\cdot)}\langle\cdot\rangle$. We have $A \equiv \nu\underline{a}.plain^s(P) \overset{(c)}{\equiv} \nu\underline{aa}_k.plain^s(P_k)$. Thus $A \to A'$ implies that $\nu\underline{aa}_k.plain^s(P_k) \to A'$ and and thus $plain^s(P_k) \to A''$ where A'' is A' with the restrictions $\nu\underline{aa}_k$ removed. (I.e. $A' \equiv \nu\underline{aa}_k.A''$.) Since P_k is s-well-formed by (d) and does not contain unprotected inputs $(s,\cdot)()$ or unprotected outputs $!\overline{(s,\cdot)}\langle\cdot\rangle$, by inspection of the definition of $plain^s$, ev^s, and $\to$, it follows that $P_k \to P'$ and $ev^s(P_k) \to ev^s(P')$ for some s-well-formed P' with $plain^s(P') \equiv A''$. The reduction $ev^s(P_k) \to ev^s(P')$ does not use the EVENT rule. Since $ev^s(P_k)$ is T_k-good by (b), from Lemma 8 (ii) we have that $ev^s(P')$ is T_k-good. Let $B' := \nu\underline{aa}_k s.(P'|syncout^s(T_k;U))$. Then $(A',B') \equiv (\nu\underline{aa}_k.plain^s(P'),B') \in \mathcal{R}$. Finally, $B = \nu\underline{a}s.(P|syncout^s(T;U)) \overset{(a)}{\to^*} \nu\underline{aa}_k s.(P_k|syncout^s(T_k;U)) \to \nu\underline{aa}_k s.(P'|syncout^s(T_k;U)) = B'$.

- If $(A,B) \in \mathcal{R}$, and $B \to B'$, then there exists an A' with $A \to^* A'$ and $(A',B') \in \mathcal{R}$:

  We have $A \equiv \nu\underline{a}.plain^s(P)$ and $B \equiv \nu\underline{a}s.(P|syncout^s(T;U))$ for some s-well-formed P and T-good $ev^s(P)$.

  We distinguish three cases for $B \to B'$:

  - $B \to B'$ is a reduction within $syncout^s(T;U)$:

    In this case, the reduction of the form $E[!\overline{(s,t)}\langle t'\rangle] \to E[\overline{(s,t)}\langle t'\rangle|!\overline{(s,t)}\langle t'\rangle]$ for some t,t'. Thus $B' \equiv \nu\underline{a}s.(P|syncout^s(T;U \cup \{t \mapsto t'\}))$. Then $A = A' := \nu\underline{a}.plain^s(P)$ and $ev^s(P)$ is T-good. Hence $A \to^* A'$ and $(A',B') \in \mathcal{R}$.

  - $B \to B'$ is a COMM reduction between P and $syncout^s(T;U)$:

    Then for some terms t,t', some process Q, and some evaluation context E, we have $P \equiv E[(s,t)().Q]$ for some t,t', and $B' \equiv \nu\underline{a}s.(P'|syncout^s(T;U'))$ with $P' := E[Q]$ and U' with $U = U' \cup \{t \mapsto t'\}$. Since $plain^s((s,t)().Q) = plain^s(Q)$, we have $A \equiv A' := \nu\underline{a}.plain^s(P')$. Furthermore, $ev^s(P) = ev^s(E)[event\ end(t).ev^s(Q)]$ and $ev^s(P') = ev^s(E)[ev^s(Q)]$. Thus by Lemma 8 (iv), the fact that $ev^s(P)$ is T-good implies that $ev^s(P')$ is T-good.

    Hence $A \to^* A'$ and $(A',B') \in \mathcal{R}$.

  - $B \to B'$ is a reduction within P.

    Thus $P \to P'$ for some P', and $B' \equiv \nu\underline{a}s.(P'|syncout^s(T;U))$. Since P is s-well-formed, we have $P \equiv E[Q] \to E[Q'] \equiv P'$ for some evaluation context E and process Q, such that Q is of the form $!\overline{(s,t)}\langle t'\rangle|Q_1$, or Q is a redex not of the form $!\overline{(s,\cdot)}\langle\cdot\rangle$, or $Q = \overline{M}\langle N\rangle.Q_1|M'(x).Q_2$ with $M \neq_E$

$(s,\cdot)$. (We cannot have a reduction on a channel $(s,\cdot)$, since s-well-formed terms have outputs on such channels only below bangs.) Without loss of generality, we can assume that all unprotected occurrences of $!\overline{(s,t)}\langle t'\rangle$ in E are not below a restriction (otherwise we could move these restrictions from E to $\nu\underline{a}$).

Let E^* be E with all unprotected occurrences of $!\overline{(s,t)}\langle t'\rangle$ removed (for arbitrary t,t'). Let T^* be the multiset of the pairs $(t\mapsto t')$ from these occurrences. Then $E[Q]\equiv E^*[Q]|\mathit{syncout}^s(T^*;\varnothing)$. Since $\mathit{ev}^s(P)=\mathit{ev}^s(E[Q])$ is T-good, and since $\mathit{ev}^s(E^*[Q])$ results from $\mathit{ev}^s(P)$ by removing $\mathit{event}\ \mathit{start}(t)$ for all $(t\mapsto\cdot)\in T^*$, by Lemma 8 (iii) we have that $\mathit{ev}^s(E^*[Q])$ is $T\cup T^*$-good.

We now distinguish on the form of Q:

∗ If $Q=!\overline{(s,t)}\langle t'\rangle|Q_1$:

Then $B'\equiv\nu\underline{a}s.(E^*[Q_1]|\mathit{syncout}^s(T';U'))$ for $T':=T\cup T^*\cup\{t\mapsto t'\}$ and $U':=U\cup\{t\mapsto t'\}$, and $A':=\nu\underline{a}s.\mathit{plain}(E^*[Q_1])=\nu\underline{a}s.\mathit{plain}(E^*[!\overline{(s,t)}\langle t'\rangle|Q_1])\equiv A$. And since $\mathit{ev}^s(E^*[Q])=\mathit{ev}^s(E^*)[\mathit{event}\ \mathit{start}(t).\mathit{ev}^s(Q_1)]$ is $T\cup T^*$-good, we have that $\mathit{ev}^s(E^*[Q])\equiv\mathit{ev}^s(E^*)[\mathit{ev}^s(Q_1)]$ is T'-good by Lemma 8 (iii). Thus $A\to^* A'$ and $(A',B')\in\mathcal{R}$.

∗ If Q is a redex, or $Q=\overline{M}\langle N\rangle.Q_1|M'(x).Q_2$ with $M=_{\mathrm{E}}M'$ and $M\neq_E(s,\cdot)$:

Then $B'\equiv\nu\underline{a}s.(P'|\mathit{syncout}^s(T';U))$ with $P'=E^*[Q']$ and $Q\mapsto Q'$ and $T':=T\cup T^*$. And $A\to A':=\nu\underline{a}.\mathit{plain}(P')$. And $\mathit{ev}^s(Q)\to\mathit{ev}^s(Q')$. Since E^* is an evaluation context and does not contain unprotected $!\overline{(s,t)}\langle t'\rangle$, we have that $\mathit{ev}^s(E^*)$ is an event evaluation context. Hence $\mathit{ev}^s(E^*[Q])=\mathit{ev}^s(E^*)[\mathit{ev}^s(Q)]\to\mathit{ev}^s(E^*)[\mathit{ev}^s(Q')]=\mathit{ev}^s(P')$, not using the EVENT rule. By Lemma 8 (ii) and using that $\mathit{ev}^s(E^*[Q])$ is T'-good, this implies that $\mathit{ev}^s(P')$ is T'-good, too. Thus $A\to^* A'$ and $(A',B')\in\mathcal{R}$.

- If $(A,B)\in\mathcal{R}$, and E is an evaluation context, then $(E[A],E[B])\in\mathcal{R}$:

We have $A\equiv\nu\underline{a}.\mathit{plain}^s(P)$ for some s-well-formed P. And $B\equiv\nu\underline{a}s.(P\mid\mathit{syncout}^s(T;U))$ for some sets T,U. And $\mathit{ev}^s(P)$ is T-good. Without loss of generality, $\underline{a},s$ do not occur in E (neither bound nor free). Let $\nu\underline{b}.E'$ be E with all restrictions over the hole moved up into $\underline{b}$. Then $E[A]\equiv\nu\underline{b}.E'[A]$ and $E[B]\equiv\nu\underline{b}.E'[B]$.

Since P is s-well-formed, and E and hence E' does not contain s, $E'[P]$ is s-well-formed.

Since E does not contain $\underline{a},s$, we have that $\underline{abs}$ are distinct names.

Since $\mathit{ev}^s(P)$ is T-good, by Lemma 8 (v) we have $\mathit{ev}^s(E'[P]))=E'[\mathit{ev}^s(P)]$ is T-good. (We use the fact that E' does not bind the $\mathit{fn}(T)$ as they have been moved into $\nu\underline{b}$.)

Thus $(\nu\underline{ab}.\mathit{plain}^s(E'[P]),\nu\underline{abs}.(E'[P]|\mathit{syncout}^s(T;U)))\in\mathcal{R}$ with $E'[P]$ instead of P and $\underline{ab}$ instead of $\underline{a}$.

By definition of $\mathit{plain}^s(\cdot)$, $E[A] \equiv \nu\underline{b}.E'[A] \equiv \nu\underline{b}.E'[\nu\underline{a}.\mathit{plain}^s(P)] = \nu\underline{ab}.\mathit{plain}^s(E'[P])$. And $E[B] \equiv \nu\underline{b}.E'[B] \equiv \nu\underline{b}.E[\nu\underline{a}s.(P|\mathit{syncout}^s(T;U))] \equiv \nu\underline{ab}s.(E'[P]|\mathit{syncout}^s(T;U))$.

Since $\mathcal{R}$ is closed under structural equivalence, this implies that $(E[A], E[B]) \in \mathcal{R}$.

Since $\mathcal{R}$ is a bisimulation, and $(\mathit{plain}^s(P), \nu s.P) \in \mathcal{R}$ (using P, s as in the statement of the lemma), we have $\mathit{plain}^s(P) \approx \nu s.P$. □

2.2.2 Unpredictability of nonces

Lemma 10 (Unpredictability of nonces)**.** *Let C be a context not binding the variable x and let P, Q be processes. Then $\nu r.C[\mathit{if}\ x = r\ \mathit{then}\ P\ \mathit{else}\ Q] \approxeq \nu r.C[Q]$.*

Proof. In the following, a *multi-hole context* is a context C with zero, one, or more holes. $C[P]$ means C with every occurrence of the hole replaced by the *same* process P.

We define the following relation $\mathcal{R}$:

$$\mathcal{R} := \Big\{(\nu r.C[\text{if } T = r \text{ then } P \text{ else } Q],\ \nu r.C[Q])\Big\}$$

up to structural equivalence. Here C ranges over multi-hole contexts, T over terms, $r \notin \mathit{fv}(T)$ over names, and P, Q over processes.

We show that $\mathcal{R}$ is a bisimulation:

- If $(A, B) \in \mathcal{R}$ and $A \downarrow_M$, then $B \rightarrow^* \downarrow_M$:

  Immediate since "if $T = r$ then P else Q" does not have unprotected outputs.

- If $(A, B) \in \mathcal{R}$ and $B \downarrow_M$, then $A \rightarrow^* \downarrow_M$:

  If the output on M is in C, $A \downarrow_M$. Otherwise the output is in an unprotected instance of Q in $\nu r.C[Q] \equiv B$. Since $r \notin \mathit{fn}(T)$, we have $T \neq_{\mathrm{E}} r$ by Lemma 3 (i) and hence (if $T = r$ then P else Q) $\rightarrow Q$. Then $A \rightarrow A'$ where A' results from replacing one instance of "if $T = r$ then P else Q" by Q. Then $A' \downarrow_M$.

- If $(A, B) \in \mathcal{R}$ and $A \rightarrow A'$ then there is a B' with $B \rightarrow^* B'$ and $(A', B') \in \mathcal{R}$:

  Then $A \equiv \nu r.C[\text{if } T = r \text{ then } P \text{ else } Q]$ and $B \equiv \nu r.C[Q]$. If the reduction $A \rightarrow A'$ takes place in C, then there is a corresponding reduction $B \rightarrow B'$ and $(A', B') \in \mathcal{R}$.

  Thus we can assume that one of the "if $T = r$ then P else Q" is being reduced in A. Since $T \neq_{\mathrm{E}} r$ by Lemma 3 (i), that subprocess reduces to Q. Thus $A' \equiv \nu r.C'[\text{if } T = r \text{ then } P \text{ else } Q]$ where C' is C with one of the holes replaced by Q. Then $B' := B \equiv \nu r.C[Q] = \nu r.C'[Q]$. Hence $B \rightarrow^* B'$ and $(A', B') \in \mathcal{R}$.

- If $(A, B) \in \mathcal{R}$ and $B \rightarrow B'$ then there is an A' with $A \rightarrow^* A'$ and $(A', B') \in \mathcal{R}$:

  Then $A \equiv \nu r.C[\text{if } T = r \text{ then } P \text{ else } Q]$ and $B \equiv \nu r.C[Q]$. As before, we have (if $T = r$ then P else Q) $\rightarrow Q$. The reduction $B \rightarrow B'$ may involve C and up to two instances of Q. We can thus write B as $B \equiv C''[Q]$ where C'' results from replacing in C the holes corresponding to these instances of Q. These instances of Q are not protected, so the holes we have replaced by Q are not protected, either. Thus $A \rightarrow^* C''[\text{if } T = r \text{ then } P \text{ else } Q] =: A''$. Then the reduction $B \equiv C''[Q] \rightarrow B'$ involves only C''. Hence $B' \equiv C'[Q]$ for some C', and $A'' \rightarrow C'[\text{if } T = r \text{ then } P \text{ else } Q] =: A'$. Thus $A \rightarrow^* A'$ and $(A', B') \in \mathcal{R}$.

- If $(A,B) \in \mathcal{R}$ and E is an evaluation context, then $(E[A], E[B]) \in \mathcal{R}$:

  Then $A \equiv \nu r.C[\text{if } T = r \text{ then } P \text{ else } Q]$ and $B \equiv \nu r.C[Q]$. Without loss of generality, $r \notin \mathit{fn}(E), \mathit{bn}(E)$. Hence $E[A] \equiv \nu r.E[C[\text{if } T = r \text{ then } P \text{ else } Q]]$ and $E[B] \equiv \nu r.E[C[Q]]$. Hence $(E[A], E[B]) \in \mathcal{R}$ (with $E[C]$ instead of C).

We can now show the lemma. Let C, P, Q, r be as in the lemma. Let σ be a substitution closing $\nu r.C[\text{if } x = r \text{ then } P \text{ else } Q]$ and $\nu r.C[Q]$. Without loss of generality, $r \notin \mathit{fn}(\sigma)$ (otherwise we rename r and change C, P, Q accordingly). In particular, $\sigma(x)$ will be some closed term T with $r \notin \mathit{fn}(T)$. Then $C[\text{if } x = r \text{ then } P \text{ else } Q]\sigma = C'[\text{if } T = r \text{ then } P' \text{ else } Q']$ and $C[Q]\sigma = C'[Q']$ where C', P', Q' are the result of applying σ to C, P, Q. (In the case of P, Q, restricted to those variables not bound by C.) And $(C'[\text{if } T = r \text{ then } P' \text{ else } Q'], C'[Q']) \in \mathcal{R}$. Thus $C'[\text{if } T = r \text{ then } P' \text{ else } Q'] \approx C'[Q']$. Since this holds for any closing σ, we have $C[\text{if } x = r \text{ then } P \text{ else } Q] \approxeq C[Q]$. □

2.3 Symbolic UC

Intuition.

We start by presenting the intuition that underlies the original UC framework [36] and thus also our work. The basic idea is to define security of a protocol π by comparing it to a so-called ideal functionality $\mathcal{F}$. The ideal functionality is a machine that by definition does what the protocol should achieve. For example, if the task of the protocol is to transmit a message m securely from Alice to Bob, then the functionality is a trusted machine that expects a message m from Alice over a secure channel, sends to the adversary that such a message was received (but does not send the message itself), and then after the adversary allows delivery, forwards the message to Bob. (In the applied pi calculus, this functionality would be $\mathit{net}_{\mathit{scstart}}().\mathit{io}_A(x).(\overline{\mathit{net}_{\mathit{notify}}}\langle\rangle \mid \mathit{net}_{\mathit{deliver}}().\overline{\mathit{io}_B}\langle x\rangle)$ where the $\mathit{net}_{\dots}$-channels belong to the adversary; see Definition 26 below.) In a sense, the functionality is an abstract specification of the protocol behavior, and the protocol is supposed to be a concrete instantiation of that specification using crypto, in a way that preserves the security properties of the specification.

So how to model that a protocol π is as secure as a functionality $\mathcal{F}$? The basic idea is to ensure that any attack on π is also possible on $\mathcal{F}$. Since by assumption $\mathcal{F}$ does not allow any attacks, this implies that π does not allow any attacks either, so π is secure. To model that any attack on π is possible on $\mathcal{F}$, we require that for any adversary attacking π, there is a corresponding adversary (the “simulator”) attacking $\mathcal{F}$ that performs an equivalent attack. And what do we mean by equivalent? Any “environment” that can observe the overall protocol outcome (inputs and outputs), and that can talk to the adversary (i.e., it learns what secret information the adversary might have obtained), cannot distinguish between the two attacks. In other words, for any adversary A, there is a simulator S such that for all environments Z, we have that $\pi + A + Z$ (the protocol running with A and Z) and $\mathcal{F} + S + Z$ are indistinguishable from Z’s point of view. Notice that we do not wish to allow Z to observe the internal protocol communication – doing so would require that π and $\mathcal{F}$ work the same way internally, but we only want that the two have the same “observable effects”, we do not care about their inner workings. Due to this, in a formal definition, we need to distinguish between the protocol-internal communication channels (net-channels), and the protocol’s interface (io-channels). Only the latter is accessible to the environment.

Formal definition.

To formalize the above intuition in the applied pi calculus, we first formalize the distinction between channels that make up the protocol's input/output interface, and those that make up the protocol's internal channels. We partition the set of all names into two sets IO and NET (both infinite). We will then require adversaries and simulators to only communicate on NET channels. (We do not forbid the environment to access NET channels. But we will give the adversary/simulator the ability to rename and hide NET channels, and thus effectively protect the protocol's NET channels from the environment.)

In order to keep the distinction between NET-channels and IO-channels, we also want to avoid that NET-channels are transmitted to the environment (we use this in a few places in our proofs):

Definition 8. *We call a process P* NET-stable *if every name $n \in \mathrm{NET} \cap \mathit{fn}(P)$ in P occurs only in channel identifiers (i.e., in particular, P does not send n to the environment).*

Note that there is no restrictions on the bound names. Thus a NET-stable adversary is free to share arbitrary fresh names with the environment and to use them as channels.

We now define the concept of an adversary. Essentially, an adversary is just a process A that is intended to interact with the protocol (or functionality). Since the adversary connects to the protocol over some NET-names, the specification of the adversary additionally includes a list of NET-names $\underline{n}$ of the protocol that will be accessed by A (and are thus private between A and the protocol). Finally, an adversary/simulator sometimes needs to rename NET-channels of the protocol/functionality to avoid name clashes. Since NET-channels are protocol internal and not part of the externally visible interface, it should not matter whether the same name is used in protocol and functionality or not. This is achieved by letting the adversary rename NET-names, we model this by specifying a renaming φ as part of the adversary.

Definition 9. *An* adversary *is a triple $(A, \varphi, \underline{n})$ where A is a closed* NET*-stable process with $\mathrm{IO} \cap \mathit{fn}(A) = \emptyset$, $\varphi : \mathrm{NET} \to \mathrm{NET}$ a bijection and $\underline{n}$ a list of names $\underline{n} \subseteq \mathrm{NET}$.*

We can now state our security definition. Both protocol and functionality are modeled by processes P and Q, respectively. An adversary $(A, \varphi_A, \underline{n}_A)$ connecting to P is modeled as $\nu\underline{n}_A.(P\varphi_A|A)$, as we would expect from the meaning of φ and $\underline{n}$ explained above. To model that P emulates Q, we would require that $\nu\underline{n}_A.(P\varphi_A|A)$ and $\nu\underline{n}_S.(Q\varphi_S|S)$ are indistinguishable for any environment for a suitable simulator $(S, \varphi_S, \underline{n}_S)$. We do not need to specify the environment explicitly because we have the notion of observational equivalence: $\nu\underline{n}_A.(P\varphi_A|A) \approx \nu\underline{n}_S.(Q\varphi_S|S)$ means that no context can distinguish the left and right hand side. The following definition captures this, except that we make one simplification: Instead of quantifying over all adversaries $(A, \varphi_A, \underline{n}_A)$, we fix $A := 0$, φ_A the identity, and $\underline{n}_A$ the empty list, so that $\nu\underline{n}_A.(P\varphi_A|A) = P$. (Such an adversary, that essentially just leaves all NET-channels accessible to the environment, is usually called a *dummy adversary*.) This definition is often technically much simpler to handle, and Lemma 11 below guarantees that it is equivalent to the more natural definition that quantifies over all adversaries.

Definition 10. *Let P and Q be processes. We say P* emulates *Q (written $P \leq Q$) iff there exists an adversary $(S, \varphi, \underline{n})$ such that $P \approx \nu\underline{n}.(Q\varphi|S)$. $(S, \varphi, \underline{n})$ will often be called* simulator.

We use $\approxeq$ instead of $\approx$ to get a more general definition, allowing non-closed P, Q. For the applications presented in this paper, the special case using $\approx$ (which is equivalent to our definition restricted to closed processes) is sufficient. (Note however that we would still use $\approxeq$ to state various technical lemmas more conveniently.)

Note that there is no formal distinction between protocols and functionalities. Indeed, it can sometimes be convenient to compare two protocols P, Q. Furthermore, note that $\leq$ is weaker than $\approxeq$: $P \approxeq Q$ entails $P \leq Q$ (and $Q \leq P$) with the simulator $(\mathbf{0}, id, \emptyset)$.

As observed in [65] there are several approaches to define simulation based security. The following Lemma shows that our definition (resembling *strong simulatability*) is equivalent to the two alternatives: *black-box simulatability* and *universally-composable simulatability* (the latter being the definition that corresponds directly to the intuition given at the beginning of this section).

Lemma 11. *For processes P, Q we have that the following are equivalent:*

(i) strong simulatability:

$$P \leq Q$$

(ii) black-box simulatability:

$$\exists(S, \varphi_S, \underline{n}_S)\ \forall(A, \varphi_A, \underline{n}_A)\ \nu\underline{n}_A.(P\varphi_A|A) \approxeq \nu\underline{n}_A.((\nu\underline{n}_S.(Q\varphi_S|S))\varphi_A|A)$$

(iii) universally-composable simulatability:

$$\forall(A, \varphi_A, \underline{n}_A)\ \exists(S, \varphi_S, \underline{n}_S)\ \nu\underline{n}_A.(P\varphi_A|A) \approxeq \nu\underline{n}_S.(Q\varphi_S|S)$$

where all triples are adversaries according to Definition 9.

Proof.

- (i) $\Rightarrow$ (ii):

$$\begin{aligned} P \leq Q &\Rightarrow \exists(S, \varphi_S, \underline{n}_S)\quad P \approxeq \nu\underline{n}_S.(Q\varphi_S|S) \\ &\overset{(*)}{\Rightarrow} \forall \text{ bijections } \varphi_A \quad P\varphi_A \approxeq (\nu\underline{n}_S.(Q\varphi_S|S))\varphi_A \\ &\overset{(**)}{\Rightarrow} \forall(A, \varphi_A, \underline{n}_A)\quad \nu\underline{n}_A.(P\varphi_A|A) \approxeq \nu\underline{n}_A.((\nu\underline{n}_S.(Q\varphi_S|S))\varphi_A|A) \end{aligned}$$

  $(*)$ since $\approxeq$ is closed under renaming and $(**)$ since $\approxeq$ is closed under the application of evaluation contexts.

- (ii) $\Rightarrow$ (iii): Let $(S, \varphi_S, \underline{n}_S)$ be the simulator from (ii), $(A, \varphi_A, \underline{n}_A)$ be an adversary and φ a bijection on names such that $\underline{n}_S(\varphi \circ \varphi_A) \cap fn(A) = \varnothing$ and φ is the identity on the free names of $Q(\varphi_A \circ \varphi_S)$ and $S\varphi_A$ (this φ can be used as α-conversion in step three below). We observe

$$\begin{aligned} &\nu\underline{n}_A.((\nu\underline{n}_S.(Q\varphi_S|S))\varphi_A|A) \\ \equiv\ &\nu\underline{n}_A.(\nu\underline{n}_S\varphi_A.(Q(\varphi_A \circ \varphi_S)|S\varphi_A)|A) \\ \equiv\ &\nu\underline{n}_A.(\nu\underline{n}_S(\varphi \circ \varphi_A).(Q(\varphi \circ \varphi_A \circ \varphi_S)|S(\varphi \circ \varphi_A))|A) \\ \equiv\ &\nu\underline{n}_A.\nu\underline{n}_S(\varphi \circ \varphi_A).(Q(\varphi \circ \varphi_A \circ \varphi_S)|S(\varphi \circ \varphi_A)|A) \end{aligned}$$

  and thus $(S_A, \underline{n}_{S_A}, \varphi_{S_A}) := (S(\varphi \circ \varphi_A)|A, \underline{n}_A \cup \underline{n}_S(\varphi \circ \varphi_A), (\varphi \circ \varphi_A \circ \varphi_S))$ is an adversary such that

$$\nu\underline{n}_A.(P\varphi_A|A) \approxeq \nu\underline{n}_{S_A}.(Q\varphi_{S_A}|S_A)$$

- (iii) $\Rightarrow$ (i) We construct the simulator from the last step for the adversary $(\mathbf{0}, \varnothing, \mathit{id})$ and have (i).

□

Lemma 12 (Reflexivity, transitivity)**.** *Let P, Q, R be processes. Then $P \leq P$. And if $P \leq Q$ and $Q \leq R$, then $P \leq R$.*

Proof. $P \leq P$ follows directly from Definition 10 by setting $S := \mathbf{0}$, φ as the identity, and $\underline{n} := \varnothing$.

Assume now that $P \leq Q$ and $Q \leq R$. Then there are processes S_1, S_2 with $\mathrm{IO} \cap \mathit{fn}(S_1) = \mathrm{IO} \cap \mathit{fn}(S_2) = \varnothing$, bijections $\varphi_1, \varphi_2 : \mathrm{NET} \to \mathrm{NET}$, and lists of names $\underline{n}_1, \underline{n}_2 \subseteq \mathrm{NET}$ such that $P \approxeq \nu\underline{n}_1.(Q\varphi_1|S_1)$ and $Q \approxeq \nu\underline{n}_2.(R\varphi_2|S_2)$. Without loss of generality we can choose $\underline{n}_2$ such that $\underline{n}_2\varphi_1 \cap \mathit{fn}(S_1) = \varnothing$. It follows

$$\begin{aligned} P &\approxeq \nu\underline{n}_1.(Q\varphi_1|S_1) \\ &\overset{(*)}{\approxeq} \nu\underline{n}_1.((\nu\underline{n}_2.(R\varphi_2|S_2))\varphi_1|S_1)] \\ &\overset{(**)}{\equiv} \nu\underline{n}_1.((\nu\underline{n}_2\varphi_1.(R(\varphi_1 \circ \varphi_2)|S_2\varphi_1))|S_1) \\ &\overset{(***)}{\equiv} \nu\underline{n}_1.\nu\underline{n}_2\varphi_1.(R(\varphi_1 \circ \varphi_2)|S_2\varphi_1|S_1) \end{aligned}$$

Here $(*)$ follows since $\approxeq$ is closed under the application of evaluation contexts and under renaming of free names.

And $(**)$ follows since for any process R, we have $(\nu\underline{n}_2.R)\varphi_1 \equiv \nu\underline{n}_2\varphi_1.(R\varphi_1)$.

And $(***)$ follows since $\underline{n}_2\varphi_1 \cap \mathit{fn}(S_1) = \varnothing$.

Thus, choosing $\underline{n} := \underline{n}_1 \cup \underline{n}_2\varphi_1$, $\varphi := \varphi_1 \circ \varphi_2$, and $S := S_2\varphi_1|S_1$, we get $P \approxeq \nu\underline{n}.(R\varphi|S)$. Hence $P \leq R$. □

Corruption.

So far, we have not yet modeled the ability of the adversary to corrupt parties. There are two main variants of corruption: static and adaptive corruption. In the case of static corruption, it is determined in the beginning of the protocol who is corrupted. For adaptive corruption, corruption may occur during the protocol and depend on protocol messages. Modeling static corruption is quite easy in our model: When a party X is corrupted, we simply remove the subprocess P_X corresponding to that party from the protocol P, make all NET-names occurring in P_X public, and – in the case of a functionality – additionally rename all IO-names of P_X into NET-names. For example, if $P = \nu \mathit{net}_1\mathit{net}_2.(P_A|P_B|\mathcal{F})$ where net_1 occurs in P_A and P_B and net_2 only in P_B, and $\mathcal{F}$ has IO-names $\mathit{io}_{FA}, \mathit{io}_{FB}$ then corrupting A leads to $P' = \nu\mathit{net}_2.(P_B|\mathcal{F}\{\mathit{net}_{FA}/\mathit{io}_{FA}\})$. And a functionality $\mathcal{G}$ with IO-names $\mathit{io}_A, \mathit{io}_B$ becomes $\mathcal{G}\{\mathit{net}_A/\mathit{io}_A\}$.

So, if we want to verify that a P emulates $\mathcal{G}$ for any corruption, we need to check:

- Uncorrupted: $P \leq \mathcal{G}$.
- Alice corrupted: $\nu\mathit{net}_2.(P_B|\mathcal{F}\{\mathit{net}_{FA}/\mathit{io}_{FA}\}) \leq \mathcal{G}\{\mathit{net}_A/\mathit{io}_A\}$.
- Bob corrupted: $P_A|\mathcal{F}\{\mathit{net}_{FB}/\mathit{io}_{FB}\} \leq \mathcal{G}\{\mathit{net}_B/\mathit{io}_B\}$.

An example is given in Section 2.8.1 in the case of UC secure commitments.

Modeling adaptive corruptions is more complex. For this one would need to introduce special parties that react to a special signal from the environment and then switch into a corrupted mode. We do not follow that approach here.

2.4 Composition

One of the salient properties of the UC framework is composition. Assume a protocol π UC-emulates a functionality $\mathcal{F}$ and ρ is a protocol using $\mathcal{F}$. Then $\rho^{\pi/\mathcal{F}}$ (which is ρ with $\mathcal{F}$ replaced by π) UC-emulates ρ. And hence, by transitivity, if ρ emulates some functionality $\mathcal{G}$, $\rho^{\pi/\mathcal{F}}$ UC-emulates $\mathcal{G}$.

In our context, ideally we would like a composition theorem such as $P \leq Q \Longrightarrow C[P] \leq C[Q]$ for arbitrary contexts C. Unfortunately, the situation is not as simple. A simple observation is that if C may contain NET-names, then composition will not work: For example, assume $P \leq Q$, and P is a protocol using some NET-channel *net* to implement an ideal functionality Q (which does not use *net*). And $C = \square|R$ receives on a NET-channel *net* and outputs the received messages on an IO-channel *io*. Then $C[P]$ will output protocol-internal messages on *io* (observable to the environment), while $C[Q]$ will not (since the functionality Q will not use the channel *net*). Hence $C[P] \not\leq C[Q]$. (We give a formal analysis of the various cases in which the composition theorem does not hold in Section 2.9).

Thus a first condition on C is that it may not use the same NET-names. In fact, we show below (Theorem 1) that if C is an evaluation context binding only IO-names and not using any of the NET-names of P, Q, then $P \leq Q \Longrightarrow C[P] \leq C[Q]$ holds.

This already allows for a large range of composition operations. (In particular, we can connect different protocols through their interfaces securely by composing them in parallel, and restricting the IO-channels through which they are connected.) But one important operation is missing, namely concurrent composition. Concurrent composition means that if $P \leq Q$, then $P' \leq Q'$ where P' consists of many instances of P and Q' analogously. Such a result is important in many cases, e.g., if P is a single session key-exchange, but an embedding protocol needs a large number of keys. The most obvious way to model this in our setting would be a theorem stating $P \leq Q \Longrightarrow\, !P \leq\, !Q$.

Unfortunately, such a theorem cannot hold, either. The intuitive reason is as follows: When trying to construct a simulator for $!Q$, then this simulator will not be able to distinguish messages from different instances of Q. The simulator will then be unable to even decide whether he talks to a single instance or several. For example:

$$\begin{aligned} P &:= \nu n\, m.\Big(\ \overline{io_1}\langle n\rangle \mid io_2(x).\text{if } x = n \text{ then } \overline{net_2}\langle m\rangle \\ &\qquad \mid io_3(x).\text{if } x = n \text{ then } \overline{net_3}\langle m\rangle\Big) \\ Q &:= \nu n.\Big(\ \overline{io_1}\langle n\rangle \mid io_2(x).\text{if } x = n \text{ then } \overline{net_2}\langle \mathit{empty}\rangle \\ &\qquad \mid io_3(x).\text{if } x = n \text{ then } \overline{net_3}\langle \mathit{empty}\rangle\Big) \end{aligned}$$

Here we have $P \leq Q$ because a simulator receiving *empty* on net_2 or net_3 just has to replace it by some fresh name m. However, we do not have $!P \leq\, !Q$. Depending on the messages the environment sends on io_2, $!P$ will output either the same name m on net_2, net_3, or different names m, m'. However, a simulator interacting with $!Q$ in both cases gets $\mathit{empty}, \mathit{empty}$ on net_2, net_3. The simulator then does not know whether he should change this into m, m or m, m' for fresh m, m'. Thus the simulator fails. (The formal argument is in Section 2.9.)

So we cannot have a theorem stating $P \leq Q \Longrightarrow\, !P \leq\, !Q$. Does this mean concurrent composition is not possible? No, just that ! is not the right operator to model it. In the computational UC framework, composition also does not involve a number of indistinguishable instances. Instead, each instance of P and Q is given a unique session id, and all communication is tagged with that session id so that it can be routed to the right instance. In our setting, one possibility to achieve this is to define an operator !! [44] such that $!!P$ behaves like an unlimited number of instances of P, where each instance is tagged with a unique session id *sid*. I.e., each channel C in P is replaced by (sid, C).[8]

The question is how to define $!!P$. The applied pi calculus does not have any construct that conveniently allows to perform infinite branching with different ids. Thus, we have to work around this restriction by introducing a more elaborate construction. As a first step, we define the tagged version $P(\!(M)\!)$ of the process P:

Definition 11. *Let P be a process, and let M be a term. We write $P(\!(M)\!)$ for P with every occurrence of $C(x)$ replaced by $(M, C)(x)$ and every occurrence of $\overline{C}\langle T\rangle$ replaced by $\overline{(M, C)}\langle T\rangle$. (If M contains bound variables or bound names from P, we assume that these bound variables/names are first renamed in P.)*

Now we have to somehow define $!!P$ as $P(\!(s_1)\!)|P(\!(s_2)\!)|\dots$ where $s_1, s_2, \dots$ range over some infinite set *SID* of session ids. Using product processes (see Section 2.1.2) this is easy: $!!P := \prod_{x\in \mathit{SID}} P(\!(x)\!)$ does the job. However, product processes are a nonstandard extension of the applied pi calculus, but we wish to stay compatible with existing variants (in particular, to be able to use Proverif for verification). Thus, instead of using $\prod_{x\in \mathit{SID}} P(\!(x)\!)$, we define a suitable context $\mathcal{C}$ such that $\mathcal{C}[P(\!(x)\!)]$ behaves like $\prod_{x\in \mathit{SID}} P(\!(x)\!)$. Then we can define $!!P := \mathcal{C}[P(\!(x)\!)]$. Of course, depending on the particular set *SID* we choose, a different context $\mathcal{C}$ will be needed. Instead of fixing a particular one, we thus give a general definition what contexts are suitable for a given set *SID*, and from then on, just assume an arbitrary such context.

Definition 12 (Indexing context). *Given a set S of terms, a variable x (will be used for tagging), and names $\underline{n}$, we call a closed context $\mathcal{C}_{x,\underline{n}}$ with $bn(\mathcal{C}_{x,\underline{n}}) = \underline{n}$ and $fn(\mathcal{C}_{x,\underline{n}}) = \emptyset$ (not containing indexed replications) an S-*indexing context *iff for all processes P with $x \notin bv(P)$* [9] *and $\underline{n} \cap fn(P) = \emptyset$ we have*

$$\mathcal{C}_{x,\underline{n}}[P(\!(x)\!)] \approx \prod_{x\in S} P(\!(x)\!)$$

In the following, we fix a set *SID* of terms containing no names or variables. The set *SID* will represent the set of all session IDs. We assume that $\mathit{id} =_{\mathrm{E}} \mathit{id}'$ entails $\mathit{id} = \mathit{id}'$ for $\mathit{id}, \mathit{id}' \in \mathit{SID}$ (different IDs are never equivalent by the equational theory).

Note that not for every set *SID* a *SID*-indexing context exists. For example, if *SID* is not semi-decidable (but the equational theory is), then there is no *SID*-indexing context. One might be concerned that our definition of *SID*-indexing contexts cannot be fulfilled. The following definition shows that this is not the case, at least if we use suitably encoded bitstrings as SIDs.

[8] One might instead consider tagging the messages sent over the channel with *sid*. This, however, does not work as well: One would need a specific multiplexer process that given a message (sid, T) discovers the corresponding instance of P and delivers to it. This might be possible, but is probably considerably more complicated than the approach we take below.

[9] P may have $x \in fv(P)$ but we forbid $x \in bv(P)$ to avoid technicalities in the definition of $P(\!(x)\!)$ due to the shadowed x.

Definition 13. *Assume that a nullary constructor nil and unary constructors zero and one are part of our symbolic model. Let SID_{bits} be the set of all terms built from $nil, zero$ and one. Assume furthermore that for $id, id' \in SID_{bits}$ in our symbolic model $id =_{\mathrm{E}} id'$ entails $id = id'$. Let*

$$\mathcal{C}_{x,a}^{SID_{bits}} := \nu a.(\overline{a}\langle nil\rangle | !a(x).(\overline{a}\langle zero(x)\rangle | \overline{a}\langle one(x)\rangle | \Box))$$

Intuitively, $\mathcal{C}_{x,a}^{SID_{bits}}$ is a factory with parameters x and a for tagged instances of P that realizes the abstract construction of $\prod_{x \in SID_{bits}} P(\!(x)\!)$.

We now show that $\mathcal{C}_{x,a}^{SID_{bits}}$ actually is an SID_{bits}-indexing context. Towards this goal we first define an intermediate representation of $\mathcal{C}_{x,a}^{SID_{bits}}$.

Definition 14. *Let P be a process. We write P^n for n parallel instances of P $(P| \dots |P)$. We define the following functions on the set of processes:*

$$\begin{aligned} G_{x,a}(P) :=& a(x).(\overline{a}\langle zero(x)\rangle \mid \overline{a}\langle one(x)\rangle \mid P) \\ \mathcal{G}_{x,a}^{n}(P) :=& (G_{x,a}(P))^n \mid !G_{x,a}(P) \\ \mathcal{C}_{x,a}^{(sID,gID,n)}(P) :=& \Sigma_{x \in sID} P \mid \nu a.(\Sigma_{x \in gID} \overline{a}\langle x\rangle \mid \mathcal{G}_{x,a}^{n}(P)) \end{aligned}$$

where $\Sigma_{x \in \mathcal{T}} P$ for a finite set of terms $\mathcal{T} = \{T_1, \dots, T_l\}$ is syntactic sugar for $P\{T_1/x\}| \dots |P\{T_l/x\}$ (this is only well-defined up to structural equivalence), $sID \subseteq SID$, $gID \subseteq SID$ and $n \in \mathbb{N}$.

Intuitively, sID (spawned IDs) contains the ids for all instances of P, that have already been tagged but are still formally a part of $\mathcal{C}_{x,a}^{SID_{bits}}$ (i.e., "are still in the factory"). gID is the foundation for the ids yet to be generated. These ids are the elements of the *span* of gID which we will introduce in the following definition. The last parameter n exists mainly for technical reasons and counts the number of currently active generator instances $G^{x,a}(P)$.

Definition 15 (Span)**.** *Let $S \subseteq SID_{bits}$ be a set of IDs. We call $\langle S\rangle := S \cup \{c_n(\dots c_2(c_1(s))\dots) : s \in S, c_i \in \{zero, one\}\}$ the span of S (note that $\langle S\rangle \subseteq SID_{bits}$).*

The following definition bridges the gap between $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))$ and $\prod_{x \in S} P(\!(x)\!)$. Have in mind that S denotes the set of ids that are yet to be used by the product process for tagging and we have $S = SID_{bits}$ at the beginning.

Definition 16 (S-valid)**.** *Let $sID \subseteq SID_{bits}$, $gID \subseteq SID_{bits}$ and $S \subseteq SID_{bits}$ be sets of ids and sID and gID be finite. We call $\mathcal{C}_{x,a}^{(sID,gID,n)}$ S-*valid *if $sID = \varnothing$ and $gID = \{nil\}$ or if*

(i) $sID \subseteq S$

(ii) $gID = \{f(x) : x \in G, f(x) \notin G, f \in \{zero, one\}\}$ *where* $G := (SID_{bits} \setminus S) \cup sID$ *(intuitively, G is the set of ids already generated)*

(iii) $\langle gID\rangle = S \setminus sID$

Lemma 13. *Let $S \subseteq SID_{bits}$ and $\mathcal{C}_{x,a}^{(sID,gID,n)}$ be S-valid where $n \geq 1$. Then for any $id \in gID$ we have that $\mathcal{C}_{x,a}^{(sID',gID',n-1)}$ is S-valid where $sID' := sID \cup \{id\}$ and $gID' := gID \setminus \{id\} \cup \{zero(id), one(id)\}$.*

Proof. Due to Definition 16 point iii we have that $gID \cap sID = \varnothing$ and $gID \subseteq S$. We check the three points of Definition 16 for sID' and gID':

(i) $id \in gID \subseteq S$ and $sID \subseteq S$ entail $sID' = (sID \cup \{id\}) \subseteq S$

(ii) For a set $G \subseteq SID_{bits}$ we define $M(G) := \{f(x) : x \in G, f(x) \notin G, f \in \{zero, one\}\}$. By assumption we have $gID = \{nil\}$ or $gID = M(G)$ for $G := (SID_{bits} \setminus S) \cup sID$. The first case leads to $sID' = \{nil\}$ and $gID' = \{zero(nil), one(nil)\}$ for which this point can easily be verified. For the second case we define $G' := G \cup \{id\}$. $id \notin M(G')$ since $id \in G'$. $f(id) \in M(G')$ for $f \in \{zero, one\}$ iff $f(id) \notin G'$. We assume towards contradiction that $f(id) \in G'$. Then $f(id) \in G$ and by definition of G $f(id) \in (SID_{bits} \setminus S) \cup sID$. However

- $f(id) \in (SID_{bits} \setminus S)$ entails $f(id) \notin S$ and thus $f(id) \notin \langle gID \rangle$. This contradicts $f(id) \in \langle gID \rangle$ (which holds since $id \in gID$).
- $f(id) \in sID$ entails $f(id) \notin \langle gID \rangle$ and leads to a contradiction analogously.

All together we have $f(id) \notin G'$ and hence
$M(G') = M(G) \setminus \{id\} \cup \{zero(id), one(id)\} = gID'$.

(iii) $\langle gID' \rangle = \langle gID \setminus \{id\} \cup \{zero(id), one(id)\} \rangle = \langle gID \rangle \setminus \{id\} = S \setminus sID \setminus \{id\} = S \setminus gID'$.

□

To show that $\mathcal{C}_{x,a}^{SID_{bits}}$ is a SID_{bits}-indexing context (see Lemma 16) we first show $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!)) \approx \nu a. \prod_{x \in S} P(\!(x)\!)$ for every S-valid $\mathcal{C}_{x,a}^{(sID,gID,n)}$.

Lemma 14. *Let P be a process and M be a term. If $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!)) \updownarrow_M$ there is exactly one $id \in sID$ such that $P(\!(id)\!) \updownarrow_M$.*

Proof. It is easy to see that $\mathcal{C}_{x,a}^{(sID,gID,n)}(\mathbf{0})$ never communicates on a channel (note that a is bound). Hence for $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!)) \updownarrow_M$ we need one of the tagged instances of P in $\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))$ to communicate on M, i.e., $P(\!(id)\!) \updownarrow_M$ for some $id \in sID$ requiring $M =_{\mathrm{E}} (id, \square)$. Analogously, for any $id' \in sID$ with $P(\!(id')\!) \updownarrow_M$ we have $M =_{\mathrm{E}} (id', \square)$. Due to Definition 5 (vi) (natural symbolic model) this entails $id =_{\mathrm{E}} id'$ which leads to $id = id'$ by definition of SID_{bits} ($sID \subseteq SID_{bits}$). Thus, the ID id with $P(\!(id)\!) \updownarrow_M$ is unique. □

Lemma 15. *Let P be a process with at most one free variable, which we call x if existent, and $x \notin bv(P)$. Let $a \notin fn(P)$ be a name. Then*

$$\mathcal{C}_{x,a}^{(\emptyset,\{nil\},0)}(P(\!(x)\!)) \approx \prod_{x \in SID_{bits}} P(\!(x)\!)$$

Proof. We define the relation

$$\mathcal{R} := \ \approx\ \cup \{(\mathcal{E}[\mathcal{C}_{x,a}^{(sID,gID,n)}(P(\!(x)\!))], \mathcal{E}[\prod_{x \in S} P(\!(x)\!)]) : \text{ for any } n \geq 0,\ S \subseteq SID_{bits},$$
$$\text{evaluation context } \mathcal{E}, \text{ process } P \text{ and } \mathcal{C}_{x,a}^{(sID,gID,n)}\ S\text{-valid}\}$$

closed under structural equivalence. Then we show that $\mathcal{R} \subseteq\ \approx$. Towards this goal we show that $\mathcal{R}$ and $\mathcal{R}^{-1}$ are simulations. We start with $\mathcal{R}$:

- $\mathcal{E}[\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))] \downarrow_M$: If $\mathcal{E}[\mathbf{0}] \downarrow_M$ we obviously have $\mathcal{E}[\prod_{x\in S} P(\!(x)\!)] \downarrow_M$. Otherwise $\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!)) \downarrow_M$. In this case, according to Lemma 14, there is a distinct $id \in sID$ such that $P(\!(id)\!) \downarrow_M$ and, since $\mathcal{E}[\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))] \downarrow_M$, $\mathcal{E}[P(\!(id)\!)] \downarrow_M$. On the other hand, due to the S-validity of $\mathcal{C}^{(sID,gID,n)}_{x,a}$, $sID \subseteq S$. With $id \in S$ we have $\prod_{x\in S} P(\!(x)\!) \rightarrow P(\!(id)\!) | \prod_{x\in S\setminus\{id\}} P(\!(x)\!)$ and hence $\mathcal{E}[\prod_{x\in S} P(\!(x)\!)] \rightarrow\downarrow_M$.

- $\mathcal{E}[\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))] \rightarrow (\mathcal{E}[\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))])'$: We distinguish three cases

  1. $\rightarrow$ does only affect $\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))$ up to structural equivalence. In this case we have $\mathcal{E}[\mathbf{0}] \rightarrow \mathcal{E}'[\mathbf{0}]$, $\mathcal{E}[\prod_{x\in S} P(\!(x)\!)] \rightarrow \mathcal{E}'[\prod_{x\in S} P(\!(x)\!)]$ and $(\mathcal{E}'[\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))], \mathcal{E}'[\prod_{x\in S} P(\!(x)\!)]) \in \mathcal{R}$.

  2. $\rightarrow$ is a COMM reduction that interferes with $\mathcal{E}$ and $\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))$. Due to Lemma 14 we find a distinct $id \in sID$ such that

     $$\mathcal{E}[\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))] \rightarrow \mathcal{E}'[P(\!(id)\!)' | \mathcal{C}^{(sID\setminus\{id\},gID,n)}_{x,a}(P(\!(x)\!))]$$

     Analogously to the case for $\mathcal{E}[\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))] \downarrow_M$ we spawn a properly tagged instance of P from $\prod_{x\in S} P(\!(x)\!)$. With $\widetilde{\mathcal{E}}[\square] := \mathcal{E}'[P(\!(id)\!)' | \square]$ we have

     $$(\widetilde{\mathcal{E}}[\mathcal{C}^{(sID\setminus\{id\},gID,n)}_{x,a}(P(\!(x)\!))], \widetilde{\mathcal{E}}[\prod_{x\in S\setminus\{id\}} P(\!(x)\!)]) \in \mathcal{R}$$

     since $\mathcal{C}^{(sID\setminus\{id\},gID,n)}_{x,a}$ is $(S \setminus \{id\})$-valid.

  3. $\rightarrow$ does only affect $\mathcal{E}$ up to structural equivalence. In this case we have $\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!)) \rightarrow \mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))'$. We distinguish three cases:

     - $\rightarrow$ is a REPL reduction and spawns a new instance of $G_{x,a}$ (see Definition 14). In this case $\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!)) \rightarrow \mathcal{C}^{(sID,gID,n+1)}_{x,a}(P(\!(x)\!))$ and $(\mathcal{E}[\mathcal{C}^{(sID,gID,n+1)}_{x,a}(P(\!(x)\!))], \mathcal{E}[\prod_{x\in S} P(\!(x)\!)]) \in \mathcal{R}$.
     - $\rightarrow$ is a COMM reduction on channel a ($\overline{a}\langle id\rangle$) (note that this requires $n \geq 1$). In this case $id \in gID \subseteq S$ and $\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!)) \rightarrow \mathcal{C}^{(sID',gID',n-1)}_{x,a}(P(\!(x)\!))$ where $sID' := sID \cup \{id\}$ and $gID' := gID \setminus \{id\} \cup \{zero(id), one(id)\}$. By Lemma 13 we see that $\mathcal{C}^{(sID',gID',n-1)}_{x,a}(P(\!(x)\!))$ is still S-valid. Hence $(\mathcal{E}[\mathcal{C}^{(sID',gID',n-1)}_{x,a}(P(\!(x)\!)))], \mathcal{E}[\prod_{x\in S} P(\!(x)\!)]) \in \mathcal{R}$.
     - $\rightarrow$ is a reduction of one of the P-instances $P(\!(id)\!)$ ($id \in sID$) (note that due to Lemma 14 and $a \notin fn(P)$ only one instance can be affected). In this case we proceed analogously to case 2.

- Obviously $\mathcal{R}$ is closed under the application of evaluation contexts.

We continue by showing the three points of observational equivalence for $\mathcal{R}^{-1}$:

- $\mathcal{E}[\prod_{x\in S} P(\!(x)\!)] \downarrow_M$ iff $\mathcal{E}[\mathbf{0}] \downarrow_M$. Therefore $\mathcal{E}[\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))] \downarrow_M$.

- $\mathcal{E}[\prod_{x\in S} P(\!(x)\!)] \rightarrow \mathcal{E}[\prod_{x\in S} P(\!(x)\!)]'$: If we have $\mathcal{E}[\prod_{x\in S} P(\!(x)\!)] \rightarrow \mathcal{E}'[\prod_{x\in S} P(\!(x)\!)]$ we have $(\mathcal{E}'[\prod_{x\in S} P(\!(x)\!)], \mathcal{E}'[\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))]) \in \mathcal{R}^{-1}$. Otherwise $\rightarrow$ is an IREPL reduction: $\prod_{x\in S} P(\!(x)\!) \rightarrow P(\!(id)\!) | \prod_{x\in S\setminus\{id\}} P(\!(x)\!)$ with $id \in S$. On the right side of the relation we have $\mathcal{E}[\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))]$. Since $\mathcal{C}^{(sID,gID,n)}_{x,a}(P(\!(x)\!))$ is S-valid, we have that $id \in sID$ or $id \in \langle gID\rangle$.

If $\mathit{id} \notin \mathit{sID}$, i.e., $\mathit{id} \in \langle \mathit{gID} \rangle$, id is of the form $\mathit{id} = c_l(\dots c_1(\mathit{id}_0)\dots)$ for some $\mathit{id}_0 \in \mathit{gID}$, some $l \in \mathbb{N}$ and $c_i \in \{\mathit{zero}, \mathit{one}\}$ for $i \in \{1, \dots, l\}$. We write id_i for $c_i(\dots c_1(\mathit{id}_0)\dots)$ for $i \in \{1, \dots, l\}$, $\overline{c_i} := \mathit{zero}$ if $c_i = \mathit{one}$, $\overline{c_i} := \mathit{one}$ otherwise and $\overline{\mathit{id}_i}$ for $\overline{c_i}(c_{i-1}(\dots c_1(\mathit{id}_0)\dots))$. The reduction $\xrightarrow{\overline{a}\langle \mathit{id}_i \rangle}$ denotes a REPL reduction that spawns an instance of $G_{x,a}$ (see Definition 14) and a following COMM reduction on channel a with message $\mathit{id}_i \in \mathit{gID}$. The application of the sequence $\xrightarrow{\overline{a}\langle \mathit{id}_0 \rangle} \dots \xrightarrow{\overline{a}\langle \mathit{id}_k \rangle}$ to $\mathcal{E}[\mathcal{C}_{x,a}^{(\mathit{sID},\mathit{gID},n)}(P(\!(x)\!))]$ for some $0 \leq k \leq l$ yields a process that is structurally equivalent to $\mathcal{E}[\mathcal{C}_{x,a}^{(\mathit{sID}_k,\mathit{gID}_k,n)}(P(\!(x)\!))]$ with $\mathit{sID}_k := \mathit{sID} \cup \{\mathit{id}_0, \dots, \mathit{id}_k\}$ and $\mathit{gID}_k := \mathit{gID} \setminus \{\mathit{id}_0\} \cup \{\overline{\mathit{id}_1}, \dots, \overline{\mathit{id}_{k-1}}\} \cup \{\mathit{zero}(\mathit{id}_k), \mathit{one}(\mathit{id}_k)\}$. For each step $k \rightsquigarrow k+1$ the S-validity of $\mathcal{C}_{x,a}^{(\mathit{sID}_k,\mathit{gID}_k,n)}$ is guaranteed by Lemma 13. We define $\mathit{sID}' := \mathit{sID}_l$ and $\mathit{gID}' := \mathit{gID}_l$ and have that $\mathit{id} \in \mathit{sID}'$.

Otherwise, if $\mathit{id} \in \mathit{sID}$, we define $\mathit{sID}' := \mathit{sID}$ and $\mathit{gID}' := \mathit{gID}$.

With $\mathit{id} \in \mathit{sID}'$ and $\mathcal{E}'[\Box] := \mathcal{E}[P(\!(\mathit{id})\!)|\Box]$ we have that

$$(\mathcal{E}'[\prod_{x \in S \setminus \{\mathit{id}\}} P(\!(x)\!)], \mathcal{E}'[\mathcal{C}_{x,a}^{(\mathit{sID}' \setminus \{\mathit{id}\},\mathit{gID}',n)}(P(\!(x)\!))]) \in \mathcal{R}^{-1}$$

since $\mathcal{C}_{x,a}^{(\mathit{sID}' \setminus \{\mathit{id}\},\mathit{gID}',n)}$ is $(S \setminus \{\mathit{id}\})$-valid.

- Obviously $\mathcal{R}^{-1}$ is closed under the application of evaluation contexts.

Since $\mathcal{C}_{x,a}^{(\emptyset,\{\mathit{nil}\},0)}$ is $\mathit{SID}_{\mathit{bits}}$-valid the Lemma holds. □

Lemma 16. *$\mathcal{C}_{x,a}^{\mathit{SID}_{\mathit{bits}}}$ is an $\mathit{SID}_{\mathit{bits}}$-indexing context.*

Proof. Let, according to Definition 12, P be a process and x be a variable with $x \notin \mathit{bv}(P)$. We pick a name a with $a \notin \mathit{fn}(P)$. We claim

$$\mathcal{C}_{x,a}^{\mathit{SID}_{\mathit{bits}}} \approxeq \prod_{x \in \mathit{SID}_{\mathit{bits}}} P(\!(x)\!)$$

We have to show $\mathcal{C}_{x,a}^{\mathit{SID}_{\mathit{bits}}}[P(\!(x)\!)]\sigma \approx (\prod_{x \in \mathit{SID}_{\mathit{bits}}} P(\!(x)\!))\sigma$ for all closing substitutions σ. W.l.o.g. $a \notin \sigma$ and $\sigma(x) = x$ and thus it suffices to show

$$\mathcal{C}_{x,a}^{\mathit{SID}_{\mathit{bits}}}[P(\!(x)\!)\sigma] \approx \prod_{x \in \mathit{SID}_{\mathit{bits}}} (P(\!(x)\!)\sigma) \tag{2.3}$$

Note that $P\sigma$ is a process with at most one free variable, denoted x. Furthermore $x \notin \mathit{bv}(P\sigma)$, $a \notin \mathit{fn}(P\sigma)$ and $\mathcal{C}_{x,a}^{\mathit{SID}_{\mathit{bits}}}[P(\!(x)\!)\sigma] = \mathcal{C}_{x,a}^{(\emptyset,\{\mathit{nil}\},0)}(P(\!(x)\!)\sigma)$ by Definition 14. By Lemma 15 we have (2.3) which concludes our proof. □

We stress that $\mathcal{C}_{x,a}^{\mathit{SID}_{\mathit{bits}}}$ is just one example of an indexing context. From now on SID is an arbitrary but fixed set of indexes and $\mathcal{C}_{x,\underline{n}}^{\mathit{SID}}$ an arbitrary but fixed SID-indexing context according to Definition 12. All our results then hold independently of the particular choice of SID.

We can now finally define $!!P$:

Definition 17 (Indexed replication)**.** *Let P be a process. We define $!!_x P := \mathcal{C}_{x,\underline{n}}^{\mathit{SID}}[P(\!(x)\!)]$ for some arbitrary $\underline{n}$ with $\underline{n} \cap \mathit{fn}(P) = \emptyset$. We write $!!P$ for $!!_x P$ with $x \notin (\mathit{fv}(P) \cup \mathit{bv}(P))$.*

Notice that our definition is a bit more general, we can even write $!!_x P$, in this case P will have access to the sid via the variable x. We need this added flexibility in Section 2.7.3 for the protocol KE^*.

Note that since $\mathcal{C}^{SID}_{x,\underline{n}}[P(\!(x)\!)] \approx \prod_{x\in S} P(\!(x)\!)$ by definition, we can actually think of $!!_x P$ as being defined as $\prod_{x\in S} P(\!(x)\!)$. Our definition, however, has the advantage that $!!_x P$ is actually a process in the original calculus, the concept of product processes was only used as a tool for defining !!.

On real-life implementations of !!.

When implementing a process $!!P$ in real life (i.e., in software for actual deployed protocols), a process such as $!!c(x).P'$ is probably best implemented by a process that listens on c for messages of the form (sid, m). Whenever such a message is received, a new instance of P' with session id sid is spawned, and all further messages with that sid are routed to that instance of P'. On the other hand, a process such as $!!\overline{c}\langle M\rangle.P'$ cannot be implemented, because such a process would non-deterministically send (sid, M) for all possible sid. A process $!!(A|B)$, where A and B correspond to processes run on different computers, does not immediately make sense, because if, e.g., A receives a message that spawns a new instance, B would have to spawn a new instance, too, without communication between A and B. Fortunately, we show in Lemma 33 below that $!!(A|B) \approx !!A \mid !!B$; then A and B can spawn instances independently.

Properties of !!.

The following four lemmas state several important properties of !!. We will need these to prove the composition theorem below. Lemmas 17, 18, and 33 also hold for ! instead of !!. But Lemma 32 is specific to !!, and is crucial for enabling the composition theorem.

Lemma 17. *Let P be a process and $\varphi : \mathcal{N} \to \mathcal{N}$ be a permutation on names. Then $(!!_x P)\varphi \equiv !!_x(P\varphi)$ for all variables $x \notin bv(P)$.*

Proof. Pick names $\underline{n}$ with $\underline{n} \cap fn(P) = \emptyset$ and $\varphi(\underline{n}) \cap fn(P) = \emptyset$. Note that $(!!_x P)\varphi \equiv \mathcal{C}^{SID}_{x,\underline{n}}[P(\!(x)\!)]\varphi$. Therefore $(!!_x P)\varphi \equiv \mathcal{C}^{SID}_{x,\underline{n}}[P(\!(x)\!)]\varphi = C^{SID}_{x,\varphi(\underline{n})}[P(\!(x)\!)\varphi] \equiv !!_x(P\varphi)$ since $\varphi(\underline{n}) \cap fn(P) = \emptyset$. □

Lemma 18. *Let P, Q be processes. Then $P \approx Q \Rightarrow !!_x P \approx !!_x Q$ for all variables $x \notin bv(P) \cup bv(Q)$.*

This lemma was surprisingly hard to prove. Before we proceed to the proof (for which we have to develop a number of auxiliary concepts and definitions first) We very roughly sketch the proof idea here: The main thing to show is that $P \approx Q \Longrightarrow P(\!(M)\!) \approx Q(\!(M)\!)$ for arbitrary fixed M. To show this, we define an operation *untag* that maps $P(\!(M)\!)$ to P, i.e., removes the tag M from all channels. Then we wish to prove that the following relation is a bisimulation: $\sim_{\mathcal{S}_{sid}} := \{(P, Q) : untag(P) \approx untag(Q)\}$. Once we have that, we see that $P(\!(M)\!) \sim_{\mathcal{S}_{sid}} Q(\!(M)\!)$ and hence $P(\!(M)\!) \approx Q(\!(M)\!)$. Unfortunately, $\sim_{\mathcal{S}_{sid}}$ is not really a bisimulation. A bisimulation must be closed under evaluation contexts, even under contexts in which not all channels are tagged with M. To solve this problem, we tweak *untag* in such a way that non-tagged channels C are mapped to specially marked channels (using a special name n_{sid})which can then be mapped back to C when tagging again. And we need to tweak the notion of a bisimulation slightly, so that $\sim_{\mathcal{S}_{sid}}$ only needs to be closed under evaluation contexts on which our operation *untag* works properly. These tweaks lead to an unexpectedly complex proof of Lemma 18.

Before we prove Lemma 18 (on page 60), we will hence need to develop a number of tools and lemmas.

Definition 18. *A set $\mathcal{S}$ of closed processes is n-*complete *for a name n iff for any closed process P with $n \notin fn(P) \cup bn(P)$, there is a closed process $S \in \mathcal{S}$ such that $P \approx S$.*

Definition 19 ($\mathcal{S}$-n-observational equivalence)**.** *Let $\mathcal{S}$ be a set of closed processes and n be a name. An $\mathcal{S}$-n-*simulation *$\mathcal{R}$ is a relation on closed processes P, Q with $n \notin (fn(P) \cup fn(Q) \cup bn(P) \cup bn(Q))$ such that $(P, Q) \in \mathcal{R}$ implies*

(i) if $P \downarrow_M$ then $Q \rightarrow^* \downarrow_M$

(ii) if $P \rightarrow P'$ with $n \notin fn(P') \cup bn(P')$ then $Q \rightarrow^* Q'$ and $(P', Q') \in \mathcal{R}$ for some Q'

(iii) $(\nu \underline{s}.(S|P), \nu \underline{s}.(S|Q)) \in \mathcal{R}$ for all closed $S \in \mathcal{S}$ and names $\underline{s} \subseteq \mathcal{N}$ with $n \notin (fn(S) \cup bn(S) \cup \underline{s})$.

*A relation $\mathcal{R}$ is an $\mathcal{S}$-n-*bisimulation *if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are $\mathcal{S}$-n-simulations. $\mathcal{S}$-n-*observational equivalence *($\approx^n_{\mathcal{S}}$) is the largest $\mathcal{S}$-n-bisimulation.*

Intuitively $\approx^n_{\mathcal{S}}$ is like observational equivalence on processes that do not contain n where the environment is restricted to be a process from $\mathcal{S}$. It is easy to check that the transitive hull of $\approx^n_{\mathcal{S}}$ satisfies the conditions (i), (ii) and (iii) from above. Hence $\approx^n_{\mathcal{S}}$ contains its own transitive hull and thus is indeed an equivalence relation.

Lemma 19. *If a set of processes $\mathcal{S}$ is n-complete and $n \notin (fn(S) \cup bn(S))$ for all $S \in \mathcal{S}$, then $P \approx^n_{\mathcal{S}} Q \Leftrightarrow P \approx Q$ for all closed processes P, Q with $n \notin (fn(P) \cup fn(Q) \cup bn(P) \cup bn(Q))$.*

Proof. Let $P, Q \in \{(P, Q) : P, Q$ closed processes with $n \notin (fn(P) \cup fn(Q) \cup bn(P) \cup bn(Q)\}$.

$\boldsymbol{P \approx Q \Rightarrow P \approx^n_{\mathcal{S}} Q}$**.**

We show that observational equivalence restricted to processes that do not contain n is an $\mathcal{S}$-n-bisimulation. Points (i) and (iii) of Definition 19 follow directly from points (i) and (iii) of observational equivalence (see Definition 4). It remains to show that for $P \rightarrow P'$ with $n \notin fn(P') \cup bn(P')$ we can find a sequence of corresponding internal reductions for Q. Since $P \approx Q$ we find a sequence $Q =: Q_1 \rightarrow \ldots \rightarrow Q_\ell =: Q'$ with $P' \approx Q'$. However, we do not necessarily have $n \notin fn(Q') \cup bn(Q')$ since this is not a requirement for observational equivalence. Fortunately, we we will see that we can find a process $\hat{Q}'$ with $Q \rightarrow^* \hat{Q}'$, $P' \approx \hat{Q}'$ and $n \notin fn(\hat{Q}') \cup bn(\hat{Q}')$. For this, we transform the sequence $Q_1 \rightarrow \ldots \rightarrow Q_\ell$ to a sequence $\hat{Q}_1 \rightarrow \ldots \rightarrow \hat{Q}_\ell$ with $Q_i \equiv_{\mathrm{E}} \hat{Q}_i$ and $n \notin fn(\hat{Q}_i) \cup bn(\hat{Q}_i)$ for $i \in \{1, \ldots, \ell\}$: First, we set $\hat{Q}_1 := Q_1$ and in particular have $Q_1 \equiv_{\mathrm{E}} \hat{Q}_1$ and $n \notin fn(\hat{Q}_1) \cup bn(\hat{Q}_1)$. For $i \in \{2, \ldots, \ell\}$ we define $\hat{Q}_i$ as follows: By Lemma 7, since $\hat{Q}_{i-1} \equiv_{\mathrm{E}} Q_{i-1} \rightarrow Q_i$, we find $\tilde{Q}$ with $\hat{Q}_{i-1} \rightarrow \tilde{Q} \equiv_{\mathrm{E}} Q_i$. W.l.o.g. we can assume $n \notin bn(\tilde{Q})$ since $\rightarrow$ and $\equiv_{\mathrm{E}}$ allow for renaming of bound names. We distinguish two cases:

- $n \notin fn(\tilde{Q})$: Then $\hat{Q}_i := \tilde{Q}$ meets our requirements.

- $n \in fn(\tilde{Q})$: Since $\hat{Q}_{i-1} \to \tilde{Q}$ and $n \notin fn(\hat{Q}_{i-1})$, the free occurrences of n can only be the result of a destructor evaluation (LET-THEN, Figure 2.3). Let D denote the corresponding destructor term with $D \Downarrow T$. By Definition 5 (vii) (natural symbolic model) and since $n \notin fn(D)$ we find a term T' with $n \notin fn(T')$ such that $D \Downarrow T'$ and $T =_{\mathrm{E}} T'$. Then $\hat{Q}_i := \tilde{Q}\{T/T'\}$ meets our requirements.

Finally, $\hat{Q}_\ell$ does not contain n and $Q = \hat{Q}_1 \to^* \hat{Q}_\ell \equiv_{\mathrm{E}} Q_\ell = Q' \approx P'$. Hence $(P', \hat{Q}_\ell) \in \approx \cap \{(P, Q) : P, Q$ closed processes with $n \notin (fn(P) \cup fn(Q) \cup bn(P) \cup bn(Q)\}$ and thus observational equivalence restricted to processes that do not contain n fulfills Definition 19 (ii).

$\boldsymbol{P \approx^n_{\mathcal{S}} Q \Rightarrow P \approx Q.}$

We first introduce a bisimulation $\approx_\varphi$ and then show $P \approx^n_{\mathcal{S}} Q \Rightarrow P \approx_\varphi Q \Rightarrow P \approx Q$: Let $\varphi : \mathcal{N} \to \mathcal{N} \setminus \{n\}$ be a bijection on names. We define

$$\approx_\varphi := \{(P, Q) : P\varphi \approx^n_{\mathcal{S}} Q\varphi\}$$

We claim that $\approx_\varphi$ is a bisimulation: It is easy to verify that $\approx_\varphi$ satisfies points (i) and (ii) of Definition 4 (both follow straightforwardly by Definition 19). For point (iii) we have to show $\mathcal{C}[P] \approx_\varphi \mathcal{C}[Q]$, i.e., $\mathcal{C}[P]\varphi \approx^n_{\mathcal{S}} \mathcal{C}[Q]\varphi$, for all evaluation contexts $\mathcal{C}$ and $P \approx_\varphi Q$, i.e., $P\varphi \approx^n_{\mathcal{S}} Q\varphi$. For any evaluation context $\mathcal{C}$ we have $\mathcal{C}[\Box] \equiv \nu\underline{n}.(C|\Box)$ for some process C and names $\underline{n} \subseteq \mathcal{N}$. Due to the completeness of $\mathcal{S}$ we find an evaluation context $\tilde{\mathcal{C}}[\Box] := \nu\underline{n}\varphi.(\tilde{C}|\Box)$ such that $C\varphi \approx \tilde{C}$ with $\tilde{C} \in \mathcal{S}$. Since n is not in the range of φ and $n \notin (fn(\tilde{C}) \cup bn(\tilde{C}))$ for $\tilde{C} \in \mathcal{S}$ we have $\tilde{\mathcal{C}}[P\varphi] \approx^n_{\mathcal{S}} \tilde{\mathcal{C}}[Q\varphi]$. Furthermore $\tilde{\mathcal{C}}[P\varphi] \approx \mathcal{C}[P]\varphi$ and hence (both sides do not contain n) $\tilde{\mathcal{C}}[P\varphi] \approx^n_{\mathcal{S}} \mathcal{C}[P]\varphi$ (analogously for Q). Altogether we have $\mathcal{C}[P]\varphi \approx^n_{\mathcal{S}} \tilde{\mathcal{C}}[P\varphi] \approx^n_{\mathcal{S}} \tilde{\mathcal{C}}[Q\varphi] \approx^n_{\mathcal{S}} \mathcal{C}[Q]\varphi$. Since $\approx_\varphi$ is symmetric by definition this closes the proof of our claim that $\approx_\varphi$ is a bisimulation.

We have that $P \approx^n_{\mathcal{S}} Q$ entails $P \approx_\varphi Q$ by definition of $\approx_\varphi$. Furthermore $P \approx_\varphi Q$ entails $P \approx Q$ since $\approx$ is the largest bisimulation. Hence $P \approx^n_{\mathcal{S}} Q$ entails $P \approx Q$. This closes the second part of our proof. □

In the following we fix a name n_{sid} and closed term M_{sid} with $n_{sid} \notin fn(M_{sid})$.

Definition 20 (Sid-sensitive processes)**.** *$\mathcal{S}_{sid}$, the set of* sid-sensitive processes, *is the set of processes following the grammar from Figure 2.4.*

Definition 21 ($\mathcal{S}_{sid}$-transformation)**.** *We define the function $\Phi : P \mapsto \Phi(P) = S$, which maps a closed process P with $n_{sid} \notin P$ to a sid-sensitive process $S \in \mathcal{S}_{sid}$, as follows:*

1. *For each protected occurrence of an input $C(x).P'$ in P we replace $C(x).P'$ by*

$$\textit{if } M_{sid} = fst(C) \textit{ then } (\textit{let } y = snd(C) \textit{ in } (M_{sid}, y)(x).P') \textit{ else } C(x).P'$$

2. *For each occurrence of an output in P we proceed analogously.*

Lemma 20. *$\mathcal{S}_{sid}$ is n_{sid}-complete.*

Proof.
- Claim 1: For all processes P we have

$$\text{if } M_{sid} = fst(C) \text{ then } (\text{let } y = snd(C) \text{ in } (M_{sid}, y)(x).P) \text{ else } C(x).P \approxeq C(x).P \tag{2.4}$$

$$
\begin{aligned}
P, Q ::= {} & \mathbf{0} \\
& (M_{sid}, C)(x).P \\
& \overline{(M_{sid}, C)}\langle T\rangle.P \\
& C^*(x).P \\
& \overline{C^*}\langle T\rangle.P \\
& \text{if } M_{sid} = \mathit{fst}(C) \text{ then } P \text{ else } C(x).Q \\
& \text{if } M_{sid} = \mathit{fst}(C) \text{ then } P \text{ else } \overline{C}\langle T\rangle.Q \\
& P \mid Q \\
& !P \\
& \nu a.P \\
& \text{let } x = D \text{ in } P \text{ else } Q
\end{aligned}
$$

Figure 2.4: Syntax of sid-sensitive processes. M_{sid} is the fixed term. C, T range over all terms with $n_{sid} \notin \mathit{fn}(C)$ and $n_{sid} \notin \mathit{fn}(T)$, C^* over all terms with $n_{sid} \notin \mathit{fn}(C^*)$ such that there is no substitution σ with $C^*\sigma =_{\mathrm{E}} (M_{sid}, \Box)$ for some term $\Box$. D is a destructor term with $n_{sid} \notin \mathit{fn}(D)$ and $a \neq n_{sid}$ is a name. Note that in the if-constructions both occurrences of C stand for the same term.

(analogously for outputs). Proof: Let σ be a closing substitution for Equation 2.4. We remember that

$$\text{if } M_{sid} = \mathit{fst}(C) \text{ then } (\text{let } y = \mathit{snd}(C) \text{ in } (M_{sid}, y)(x).P) \text{ else } C(x).P$$

is just syntactic sugar for

$$\text{let } z = \mathit{equals}(M_{sid}, \mathit{fst}(C)) \text{ in } (\text{let } y = \mathit{snd}(C) \text{ in } (M_{sid}, y)(x).P) \text{ else } C(x).P$$

By definition of *equals* we have $\mathit{equals}(M_{sid}, \mathit{fst}(C))\sigma \Downarrow M_{sid}$ iff $\mathit{fst}(C)\sigma \Downarrow M_{sid}$. We distinguish two cases:

– If $\mathit{fst}(C\sigma) \Downarrow M_{sid}$, then by Definition 5 (v) (natural symbolic model) we have that $(M_{sid}, C_2) =_{\mathrm{E}} C\sigma$ for all C_2 with $\mathit{snd}(C\sigma) \Downarrow C_2$. Hence

$$
\begin{aligned}
& \text{if } M_{sid} = \mathit{fst}(C\sigma) \text{ then} \\
& \quad (\text{let } y = \mathit{snd}(C\sigma) \text{ in } (M_{sid}, y)(x).P\sigma) \text{ else } C\sigma(x).P\sigma \\
\overset{(*)}{\approx}\ & \text{if } M_{sid} = \mathit{fst}((M_{sid}, C_2)) \text{ then} \\
& \quad \text{let } y = \mathit{snd}((M_{sid}, C_2)) \text{ in } (M_{sid}, y)(x).P\sigma \\
& \text{else} \\
& \quad (M_{sid}, C_2)(x).P\sigma \\
\overset{(**)}{\approx}\ & \text{let } y = \mathit{snd}((M_{sid}, C_2)) \text{ in } (M_{sid}, y)(x).P\sigma \\
\overset{(**)}{\approx}\ & (M_{sid}, C_2)(x).P\sigma \\
\overset{(*)}{\approx}\ & C(x).P
\end{aligned}
$$

$(*)$ by Lemma 4 (iv) and $(**)$ by Lemma 4 (vii).

  - If $\mathit{fst}(C)\sigma \not\Downarrow M_{\mathit{sid}}$, then the claim follows by Lemma 4 (vi).

- Claim 2: $P \approxeq \Phi(P)$. We prove this by structural induction on P. Since Φ does only affect in- and outputs we can focus on those: If $P = C(x).P'$ then

$$\begin{aligned}
P &= C(x).P' \\
&\overset{(*)}{\approxeq} C(x).\Phi(P') \\
&\overset{(**)}{\approxeq} \text{if } M_{\mathit{sid}} = \mathit{fst}(C) \text{ then (let } y = \mathit{snd}(C) \text{ in } (M_{\mathit{sid}}, y)(x).P') \text{ else } C(x).P' \\
&= \Phi(P)
\end{aligned}$$

  where $(*)$ holds by the induction hypothesis and $(**)$ by Claim 1.

For any closed P we have $P \approxeq \Phi(P)$ by Claim 2. $\Phi(P)$ is closed since P is closed and hence $P \approx \Phi(P)$. For P with $n_{\mathit{sid}} \notin (\mathit{fn}(P) \cup \mathit{bn}(P))$ we have $\Phi(P) \in \mathcal{S}_{\mathit{sid}}$. Thus $\mathcal{S}_{\mathit{sid}}$ is n_{sid}-complete. □

Lemma 21. *For closed $S \in \mathcal{S}_{\mathit{sid}}$ and $S \to S'$ with $n_{\mathit{sid}} \notin \mathit{fn}(S') \cup \mathit{bn}(S')$ we have $S' \in \mathcal{S}_{\mathit{sid}}$.*

Proof. First, we observe that all processes not containing n_{sid} and being structurally equivalent to a sid-sensitive process are sid-sensitive as well. Furthermore $\mathcal{C}[P]$, where $\mathcal{C}$ is an evaluation context and P a process, is sid-sensitive iff $\mathcal{C}[\mathbf{0}]$ and P are sid-sensitive. In all cases w.l.o.g. $n_{\mathit{sid}} \notin \mathit{fn}(\mathcal{C}) \cup \mathit{bn}(\mathcal{C})$ because $\equiv$ does not introduce free names and bound names are w.l.o.g. not n_{sid}. We have the following cases:

- REPL: $S \equiv \mathcal{C}[!P] \to \mathcal{C}[P|!P] \equiv S'$. $!P$ is sid-sensitive, hence P and $P|!P$ are.
- COMM: $S \equiv \mathcal{C}[\overline{C}\langle T\rangle.P|\tilde{C}(x).Q] \to \mathcal{C}[P|Q\{T/x\}] \equiv S'$. Q is sid-sensitive and $n_{\mathit{sid}} \notin \mathit{fn}(T)$ since $n_{\mathit{sid}} \notin \mathit{fn}(S) \cup \mathit{bn}(\mathcal{C})$. We can easily check the grammar of sid-sensitive processes from Figure 2.4 to see that a substitution $\{T/x\}$ with $n_{\mathit{sid}} \notin T$ applied to a sid-sensitive process yields a sid-sensitive process. Therefore $Q\{T/x\}$ and $P|Q\{T/x\}$ are sid-sensitive.
- LET-THEN: $S \equiv \mathcal{C}[\text{let } x = D \text{ in } P \text{ else } Q] \to \mathcal{C}[P\{T/x\}] \equiv S'$ for some term T with $D \Downarrow T$ and $n_{\mathit{sid}} \notin \mathit{fn}(T)$ since $n_{\mathit{sid}} \notin \mathit{fn}(S') \cup \mathit{bn}(\mathcal{C})$. Analogously to the argument in the COMM case, $P\{T/x\}$ is sid-sensitive.
- LET-ELSE: Here, according to the grammar of sid-sensitive processes from Figure 2.4, we distinguish three cases:
  - $S \equiv \mathcal{C}[\text{if } M_{\mathit{sid}} = \mathit{fst}(C) \text{ then } P \text{ else } C(x).Q] \to \mathcal{C}[C(x).Q] \equiv S'$. C is closed since S is closed. $M_{\mathit{sid}} = \mathit{fst}(C)$ is false, i.e., there is no term M such that $\mathit{equals}(M_{\mathit{sid}}, \mathit{fst}(C)) \Downarrow M$. Therefore $\mathit{fst}(C) \not\Downarrow=_{\mathrm{E}} M_{\mathit{sid}}$. This implies $C \neq_{\mathrm{E}} (M_{\mathit{sid}}, X)$ for all terms X by Definition 5 (v) (natural symbolic model). Hence $C(x).Q$ is sid-sensitive (matching the $C^*(x).P$ rule).
  - $S \equiv \mathcal{C}[\text{if } M_{\mathit{sid}} = \mathit{fst}(C) \text{ then } P \text{ else } \overline{C}\langle T\rangle.Q] \to \mathcal{C}[\overline{C}\langle T\rangle.Q] \equiv S'$. Analogously to the previous case.
  - $S \equiv \mathcal{C}[\text{let } x = D \text{ in } P \text{ else } Q] \to \mathcal{C}[Q] \equiv S'$. Q is sid-sensitive by definition.

This concludes our proof. □

Definition 22 (n_{sid}-good)**.** *A process P is n_{sid}-good if it follows the grammar from Figure 2.5.*

$$\begin{aligned}
P, Q ::= {} & \mathbf{0} \\
& C(x).P \\
& \overline{C}\langle T\rangle.P \\
& (n_{sid}, C^*)(x).P \\
& \overline{(n_{sid}, C^*)}\langle T\rangle.P \\
& \text{if } M_{sid} = \mathit{fst}(C) \text{ then } P \text{ else } (n_{sid}, C)(x).Q \\
& \text{if } M_{sid} = \mathit{fst}(C) \text{ then } P \text{ else } \overline{(n_{sid}, C)}\langle T\rangle.Q \\
& P \mid Q \\
& !P \\
& \nu a.P \\
& \text{let } x = D \text{ in } P \text{ else } Q
\end{aligned}$$

Figure 2.5: Syntax of n_{sid}-good processes. M_{sid} is the fixed term. C, T range over all terms with $n_{sid} \notin \mathit{fn}(C)$, $n_{sid} \notin \mathit{fn}(T)$. C^* ranges over all terms with $n_{sid} \notin \mathit{fn}(C^*)$ such that there is no substitution σ with $C^*\sigma =_{\mathrm{E}} (M_{sid}, T)$ for some term T. D is a destructor term with $n_{sid} \notin \mathit{fn}(D)$ and $a \neq n_{sid}$ is a name. Note that in the if-constructions both occurrences of C stand for the same term.

Definition 23 (tag)**.** *We define the function tag on terms:*

$$\begin{aligned}
\mathit{tag}((n_{sid}, C)) :=& C \\
\mathit{tag}(C) :=& (M_{sid}, C) \text{ otherwise}
\end{aligned}$$

Let P be an n_{sid}-good process. Then we write $\mathit{tag}(P)$ for the process that results from replacing any channel identifier C by $\mathit{tag}(C)$ in P.

The function *tag* adds a tag M_{sid} to all channel identifiers in a process. We will see that *tag* returns a sid-sensitive process. We will need that *tag* is a bijective mapping between n_{sid}-good processes and sid-sensitive processes. The special name n_{sid} is needed to cover the corner cases when constructing that bijection.

Lemma 22. *Let P be an n_{sid}-good process. Then $\mathit{tag}(P) \in \mathcal{S}_{sid}$.*

Proof. We do a structural induction over the grammar of n_{sid}-good processes from Figure 2.5. Assume that $\mathit{tag}(P')$ and $\mathit{tag}(Q')$ are in $\mathcal{S}_{sid}$.

- For the communication on a channel C with $n_{sid} \notin \mathit{fn}(C)$ we have $\mathit{tag}(C(x).P')$ $= (M_{sid}, C)(x).\mathit{tag}(P')$ which is obviously in $\mathcal{S}_{sid}$. $\mathit{tag}(\overline{C}\langle T\rangle.P')$ analogous.
- For the communication on a channel $C = (n_{sid}, C^*)$ we have $\mathit{tag}((n_{sid}, C^*)(x).P')$ $= C^*(x).\mathit{tag}(P')$. $C^*(x).\mathit{tag}(P')$ is in $\mathcal{S}_{sid}$ since, by definition of n_{sid}-good, there is no substitution σ with $C^*\sigma =_{\mathrm{E}} (M_{sid}, T)$ for some term T. $\overline{(n_{sid}, C^*)}\langle T\rangle.P'$ analogous.
- For the first pair of if statements we have that

$$\begin{aligned}
&\mathit{tag}(\text{if } M_{sid} = \mathit{fst}(C) \text{ then } P' \text{ else } (n_{sid}, C)(x).Q') \\
=& (\text{if } M_{sid} = \mathit{fst}(C) \text{ then } \mathit{tag}(P') \text{ else } C(x).\mathit{tag}(Q'))
\end{aligned}$$

is in $\mathcal{S}_{sid}$ since $n_{sid} \notin \mathit{fn}(C)$. Analogous for $\overline{(n_{sid}, C)}\langle T\rangle.Q'$ in the ELSE branch.

Checking the remaining rules from Figure 2.5 is a straightforward task. □

Definition 24 (untag)**.** *We define the function untag on terms:*

$$\begin{aligned} untag((M_{sid}, C)) &:= C \\ untag(C) &:= (n_{sid}, C) \text{ otherwise} \end{aligned}$$

Let P be a sid-sensitive process. Then we write $untag(P)$ for the process that results from replacing any channel identifier C by $untag(C)$.

Lemma 23. *Let $P \in \mathcal{S}_{sid}$ be a sid-sensitive process. Then $untag(P)$ is n_{sid}-good.*

Proof. Analogous to the proof of Lemma 22 a straightforward structural induction shows this Lemma. We quickly sketch the interesting cases:

- $untag((M_{sid}, C)(x).P') = C(x).untag(P')$ matches rule $C(x).P$ from Figure 2.5 (note that $n_{sid} \notin fn(C)$). $\overline{(M_{sid}, C)}\langle T\rangle.P'$ analogous.
- $untag(C^*(x).P') = (n_{sid}, C^*)(x).untag(P')$: $untag(C^*) = (n_{sid}, C^*)$ since there is no substitution σ with $C^*\sigma =_{\mathrm{E}} (M_{sid}, \Box)$ for some term $\Box$. The expression matches rule $(n_{sid}, C^*)(x).P$ from Figure 2.5. $\overline{C^*}\langle T\rangle.P$ analogous.
- For the first if-rule we distinguish two cases:
  - $C \neq (M_{sid}, \Box)$. Then

    $$\begin{aligned} &untag(\text{if } M_{sid} = fst(C) \text{ then } P' \text{ else } C(x).Q') \\ = &(\text{if } M_{sid} = fst(C) \text{ then } untag(P') \text{ else } (n_{sid}, C)(x).untag(Q')) \end{aligned}$$

    matches rule (if $M_{sid} = fst(C)$ then P else $(n_{sid}, C)(x).Q$) from Figure 2.5.
  - $C = (M_{sid}, C')$. Then

    $$\begin{aligned} &untag(\text{if } M_{sid} = fst(C) \text{ then } P' \text{ else } C(x).Q') \\ = &(\text{if } M_{sid} = fst((M_{sid}, C')) \text{ then } untag(P') \text{ else } C'(x).untag(Q')) \\ = &(\text{let } y = equals(M_{sid}, fst((M_{sid}, C'))) \text{ in } untag(P') \text{ else } C'(x).untag(Q')) \end{aligned}$$

    $n_{sid} \notin fn(C')$ since $n_{sid} \notin fn(C)$. Hence $C'(x).untag(Q')$ is n_{sid}-good. The process

    $$(\text{let } y = equals(M_{sid}, fst((M_{sid}, C'))) \text{ in } untag(P') \text{ else } C'(x).untag(Q'))$$

    matches rule (let $x = D$ in P else Q) from Figure 2.5 with $D = equals(M_{sid}, fst((M_{sid}, C')))$.

  Analogous for $\overline{C}\langle T\rangle.Q'$ in the ELSE branch.

□

Lemma 24. *Let P be an n_{sid}-good process. Then $untag(tag(P)) \approx P$.*

Proof. We prove this lemma by structural induction over P according to the grammar from Figure 2.5.

- $P = C(x).P'$ where C is a channel identifier with $n_{sid} \notin C$: Then $C \neq (n_{sid}, C')$ for some term C' and thus $tag(C) = (M_{sid}, C)$. Hence $untag(tag(C)) = C$ and $untag(tag(P)) = untag(tag(C(x).P')) = C(x).untag(tag(P')) \approx C(x).P' = P$ by the induction hypothesis and since $\approx$ is closed under the application of contexts (Lemma 1). $P = \overline{C}\langle T\rangle.P'$ analogously.

- $P = (n_{sid}, C^*)(x).P'$ for some term C^* with $n_{sid} \notin fn(C^*)$ and $C^*\sigma \neq_{\mathrm{E}} (M_{sid}, \tilde{C}^*)$ for all substitutions σ and terms $\tilde{C}^*$. Certainly $tag((n_{sid}, C^*)) = C^*$. By assumption $C^* \neq (M_{sid}, \tilde{C}^*)$ and thus $untag(tag((n_{sid}, C^*))) = untag(C^*) = (n_{sid}, C^*)$. The rest of this case, as well as the case for $P = \overline{(n_{sid}, C^*)}\langle T\rangle.P'$, is analogous to the previous case.

- $P =$ if $M_{sid} = fst(C)$ then P' else $(n_{sid}, C)(x).Q'$ where $n_{sid} \notin fn(C)$: Clearly $tag((n_{sid}, C)) = C$. We now distinguish two cases for C:
  - $C = (M_{sid}, C')$ for some term C'. Then $untag(C) = untag((M_{sid}, C')) = C' \neq C$. This is the reason why we cannot have $untag(tag(P)) = P$ in general. However,

$$
\begin{aligned}
& untag(tag(P)) \\
={}& untag(tag(\text{if } M_{sid} = fst(C) \text{ then } P' \text{ else } (n_{sid}, C)(x).Q')) \\
={}& \text{if } M_{sid} = fst((M_{sid}, C')) \text{ then} \\
& \quad untag(tag(P')) \\
& \ \text{else} \\
& \quad untag(tag((n_{sid}, C)(x).Q')) \\
\overset{(*)}{\approx}{}& untag(tag(P')) \overset{(**)}{\approx} P' \\
\overset{(*)}{\approx}{}& \text{if } M_{sid} = fst((M_{sid}, C')) \text{ then } P' \text{ else } (n_{sid}, C)(x).Q' \\
={}& \text{if } M_{sid} = fst(C) \text{ then } P' \text{ else } (n_{sid}, C)(x).Q' = P
\end{aligned}
$$

  In both cases $(*)$ holds by Lemma 4 (vii) and Definition 5 (iv) (natural symbolic model). $(**)$ holds by the induction hypothesis.

  - Otherwise $untag(C) = (n_{sid}, C)$ and it is easy to see that $untag(tag(P)) = P$.

  $P =$ if $M_{sid} = fst(C)$ then P' else $\overline{(n_{sid}, C)}\langle T\rangle.Q'$ analogously.

The missing cases for parallel composition, bang, name restriction and let-statement all work straightforwardly.

□

Lemma 25. *Let P be a sid-sensitive process. Then $tag(untag(P)) = P$.*

Proof. Since *tag* and *untag* do only modify channel identifiers we show $tag(untag(C)) = C$ for the different kinds of channel identifiers that are allowed in an sid-sensitive process by Figure 2.4:

- C is a channel identifier with $C = (M_{sid}, C')$ for some term C' with $n_{sid} \notin fn(C')$: Then $untag(C) = C'$ and $tag(C') = (M_{sid}, C') = C$ since $n_{sid} \notin fn(C)$. Hence $untag(tag(C)) = C$.

- C is a channel identifier C^* with $n_{sid} \notin fn(C^*)$ and $C^*\sigma \neq_{\mathrm{E}} (M_{sid}, \tilde{C}^*)$ for all substitutions σ and terms $\tilde{C}^*$. Then $tag(untag(C)) = tag((n_{sid}, C^*)) = C^* = C$.

- C is a channel identifier with $n_{sid} \notin fn(C)$ in the ELSE-branch of (if $tag = fst(C)$). We distinguish two cases:

- $C = (M_{sid}, C')$ for some term C'. Then $untag(C) = C'$ and $tag(C') = (M_{sid}, C')$ since $n_{sid} \notin fn(C') \subseteq fn(C)$.
- Otherwise $untag(C) = (n_{sid}, C)$ and $tag((n_{sid}, C)) = C$.

In both cases we have $untag(tag(C)) = C$.

□

Definition 25. *We define a relation* $\sim_{\mathcal{S}_{sid}} := \{(P, Q) : P, Q \in \mathcal{S}_{sid}, untag(P) \approx untag(Q)\}$.

Lemma 26. *Assume that* $\sim_{\mathcal{S}_{sid}}$ *is an* $\mathcal{S}_{sid}$*-bisimulation and* $P \approx Q$ *for closed* n_{sid}*-good processes* P *and* Q*. Then* $tag(P) \approx tag(Q)$.

Proof. Note that $tag(P)$ and $tag(Q)$ are sid-sensitive processes by Lemma 22 and thus do not contain n_{sid}. We have

$$\begin{aligned} P \approx Q \Rightarrow & untag(tag(P)) \approx P \approx Q \approx untag(tag(Q)) \text{ (by Lemma 24)} \\ \Rightarrow & tag(P) \sim_{\mathcal{S}_{sid}} tag(Q) \\ \Rightarrow & tag(P) \approx^{n_{sid}}_{\mathcal{S}_{sid}} tag(Q) \\ & \text{(since } \approx^{n_{sid}}_{\mathcal{S}_{sid}} \text{ is the largest } \mathcal{S}_{sid}\text{-bisimulation by Definition 19)} \\ \Rightarrow & tag(P) \approx tag(Q) \text{ (by Lemmas 19, 20)} \end{aligned}$$

□

Lemma 27. *Let* P *be a closed* n_{sid}*-good process with* $P \equiv_{\mathrm{E}} Q \to Q'$ *for some closed processes* Q, Q'*. Then there is a closed* n_{sid}*-good process* P' *such that* $P \to P' \equiv_{\mathrm{E}} Q'$ *and* $tag(P) \to tag(P')$.

Proof. According to Lemma 7 we find a closed process $\tilde{P}'$ such that $P \to \tilde{P}' \equiv_{\mathrm{E}} Q'$ (this holds for any P, not just for n_{sid}-good ones). Now we show that if P is additionally n_{sid}-good, there is a closed n_{sid}-good process P' with $P \to P' \equiv_{\mathrm{E}} \tilde{P}'$ and $tag(P) \to tag(P')$ which proves the Lemma.

First, we make some general observations: For $P \to \tilde{P}'$ we find an evaluation context $\mathcal{C}$ and processes R, R' such that $P \equiv \mathcal{C}[R] \to \mathcal{C}[R'] \equiv \tilde{P}'$ and $R \to R'$ is a direct application of one of the rules for internal reductions from Figure 2.3. Furthermore, it is easy to verify that any process A with $P \equiv A$ and $n_{sid} \notin bn(A)$ is also n_{sid}-good and $tag(P) \equiv tag(A)$. Additionally, $\mathcal{C}[R]$ is n_{sid}-good iff $\mathcal{C}[\mathbf{0}]$ and R are n_{sid}-good. Hence, w.l.o.g. (since $\equiv$ allows for renaming of bound names), we can assume $\mathcal{C}[\mathbf{0}]$ and R to be n_{sid}-good. Since $tag(\mathcal{C}[R]) = tag(\mathcal{C})[tag(R)]$, it remains to show that R' is n_{sid}-good and that $tag(R) \to tag(R')$. We will be able to show this for the REPL, the COMM and the THEN-ELSE rules and have that $P' := \mathcal{C}[R'] \equiv \tilde{P}' \equiv Q'$ in these cases. In the LET-THEN case however, the destructor evaluation might introduce a term T containing a free occurrence of n_{sid}. Fortunately, replacing T with an equivalent term T' will solve the problem and we have that $P' := \mathcal{C}[R'\{T/T'\}] \equiv_{\mathrm{E}} \tilde{P}' \equiv Q'$ for $R'\{T/T'\}$ being n_{sid}-good. In detail:

- REPL: $!R \to \mathcal{C}[R|!R] \equiv \tilde{P}'$ where w.l.o.g. $\mathcal{C}[!R]$ and therefore $\mathcal{C}[R|!R]$ are n_{sid}-good. We set $P' := \mathcal{C}[R|!R]$ and have $tag(P) \equiv tag(\mathcal{C}[!R]) = tag(\mathcal{C})[!tag(R)] \overset{(*)}{\to} tag(\mathcal{C})[tag(R)|!tag(R)] = tag(\mathcal{C}[R|!R]) = tag(P')$. $(*)$ by the REPL rule.

- COMM: Analogously to REPL $P \equiv \mathcal{C}[\overline{C}\langle T\rangle.R|\tilde{C}(x).\tilde{R}] \to \mathcal{C}[R|\tilde{R}\{T/x\}] \equiv \tilde{P}'$ where $C =_{\mathrm{E}} \tilde{C}$ and w.l.o.g. $\mathcal{C}[\overline{C}\langle T\rangle.R|\tilde{C}(x).\tilde{R}]$ and $\mathcal{C}[R|\tilde{R}\{T/x\}]$ are n_{sid}-good. We observe
$$\mathit{tag}(\overline{C}\langle T\rangle.R) = \overline{\mathit{tag}(C)}\langle T\rangle.\mathit{tag}(R) \text{ and } \mathit{tag}(\tilde{C}(x).\tilde{R}) = \mathit{tag}(\tilde{C})(x).\mathit{tag}(\tilde{R})$$
by Definition 23. Analogously to REPL we have to show
$$\mathit{tag}(\mathcal{C})[\overline{\mathit{tag}(C)}\langle T\rangle.\mathit{tag}(R)|\mathit{tag}(\tilde{C})(x).\mathit{tag}(\tilde{R})] \to \mathit{tag}(\mathcal{C})[\mathit{tag}(R)|\mathit{tag}(\tilde{R})\{T/x\}]$$
Note that $\mathit{tag}(\tilde{R})\{T/x\} = \mathit{tag}(\tilde{R}\{T/x\})$ since $n_{sid} \notin \mathit{fn}(T)$. Hence it is necessary and sufficient to show $\mathit{tag}(C) =_{\mathrm{E}} \mathit{tag}(\tilde{C})$. Now we distinguish two cases to show $\mathit{tag}(C) =_{\mathrm{E}} \mathit{tag}(\tilde{C})$:
  - $C = (n_{sid}, C')$ for some term C'. By assumption we have $C =_{\mathrm{E}} \tilde{C}$ and hence $\tilde{C} =_{\mathrm{E}} (n_{sid}, C')$. By the grammar of n_{sid}-good processes (Figure 2.5) we have $n_{sid} \notin \mathit{fn}(\tilde{C})$ or $\tilde{C} = (n_{sid}, C^*)$ for some C^*. Lemma 3 (iii) above excludes the first case and leaves us with $\tilde{C} = (n_{sid}, C^*)$. By Definition 5 (vi) (natural symbolic model) we have $C' =_{\mathrm{E}} C^*$ and hence $\mathit{tag}(\tilde{C}) = C^* =_{\mathrm{E}} C' = \mathit{tag}(C)$.
  - $C \neq (n_{sid}, C')$ for any term C'. By the grammar of n_{sid}-good processes (Figure 2.5) we then have $n_{sid} \notin \mathit{fn}(C)$. $\tilde{C} = (n_{sid}, C')$ for some term C' leads to $C =_{\mathrm{E}} (n_{sid}, C')$ which contradicts Lemma 3 (iii). Hence (again by the grammar of n_{sid}-good processes) $n_{sid} \notin \mathit{fn}(\tilde{C})$. Thus $\mathit{tag}(C) = (M_{sid}, C) =_{\mathrm{E}} (M_{sid}, \tilde{C}) = \mathit{tag}(\tilde{C})$.
- LET-THEN: $P \equiv \mathcal{C}[\text{let } x = D \text{ in } R \text{ else } \tilde{R}] \to \mathcal{C}[R\{T/x\}] \equiv \tilde{P}'$ with $D \Downarrow T$. By Definition 5 (vii) (natural symbolic model) we find T' with $n_{sid} \notin T'$, $D \Downarrow T'$ and $T' =_{\mathrm{E}} T$. Hence we have
$$P \equiv \mathcal{C}[\text{let } x = D \text{ in } R \text{ else } \tilde{R}] \to \mathcal{C}[R\{T'/x\}] =: P'$$
and $P' \equiv_{\mathrm{E}} \tilde{P}' \equiv Q'$. Altogether
$$\begin{aligned} \mathit{tag}(P) &\equiv \mathit{tag}(\mathcal{C}[\text{let } x = D \text{ in } R \text{ else } \tilde{R}]) \\ &= \mathit{tag}(\mathcal{C})[\text{let } x = D \text{ in } \mathit{tag}(R) \text{ else } \mathit{tag}(\tilde{R})] \\ &\to \mathit{tag}(\mathcal{C})[\mathit{tag}(R)\{T'/x\}] \\ &\overset{(*)}{=} \mathit{tag}(\mathcal{C})[\mathit{tag}(R\{T'/x\})] \\ &\equiv \mathit{tag}(P') \end{aligned}$$
$(*)$ since $n_{sid} \notin \mathit{fn}(T')$.
- LET-ELSE is not affected by *tag* and the proof is analogous to that for the REPL rule.

□

Lemma 28. *Let P be a closed sid-sensitive process and P' be a closed process with $n_{sid} \notin \mathit{fn}(P')$. Then there is a process P^* with $\mathit{untag}(P) \to P^*$ and $P^* \approx \mathit{untag}(P')$.*

Proof. The rest of this proof is partially analogous to that of Lemma 27. Similarly, we can focus on the rules from Figure 2.3 directly. The main difference is that, for some sid-sensitive process R and term T with $n_{sid} \notin \mathit{fn}(T)$, $\mathit{untag}(R)\{T/x\} \neq \mathit{untag}(R\{T/x\})$. Instead, we only have $\mathit{untag}(R)\{T/x\} \approxeq \mathit{untag}(R\{T/x\})$ (we are going to prove that first). Therefore the COMM rule and the LET-THEN rule, where substitutions occur, have to be handled differently. The arguments for the REPL rule and the LET-ELSE rule are analogous.

Claim: If R is a sid-sensitive process, $\boldsymbol{untag(R)\{T/x\} \approx untag(R\{T/x\})}$ for all $\boldsymbol{T}$ with $\boldsymbol{n_{sid} \notin fn(T)}$.

For all channel identifiers $C = (M_{sid}, C')$ and $C = C^*$ according Figure 2.4 we obviously have $untag(C)\{T/x\} = untag(C\{T/x\})$ for all substitutions $\{T/x\}$. However, in the ELSE-branch of (if $M_{sid} = fst(C)$), C can be an arbitrary term with $n_{sid} \notin fn(C)$. If $C = (M_{sid}, C')$ for some term C', $untag(C)\{T/x\} = untag(C\{T/x\})$ holds. Otherwise, for a substitution $\{T/x\}$, we distinguish two cases:

- $C\{T/x\} \neq (M_{sid}, C')$ for all terms C'. Then $untag(C)\{T/x\} = (n_{sid}, C\{T/x\}) = untag(C\{T/x\})$.
- Otherwise $C\{T/x\} = (M_{sid}, C')$ for some term C'. Then $untag(C)\{T/x\} = (n_{sid}, C\{T/x\}) \neq C' = untag(C\{T/x\})$. Since $fst(C\{T/x\}) \Downarrow M_{sid}$ the ELSE-branch of R will never be executed and we, analogously to the proof of Lemma 24, replace $(n_{sid}, C\{T/x\})$ by C' to have $untag(R)\{T/x\} \approx untag(R\{T/x\})$.

Note that P' is sid-sensitive by Lemma 21.

We now handle the COMM rule and the LET-THEN rule:

- COMM: Analogously to Lemma 27 we have to prove $untag(C) =_{\mathrm{E}} untag(\tilde{C})$ where C and $\tilde{C}$ are the channel identifiers used for communication. By the grammar of sid-sensitive processes from Figure 2.4 all channel identifiers which occur unrestricted are either of the form (a) (M_{sid}, C') for some term C' or (b) C^* such that $C^*\sigma \neq_{\mathrm{E}} (M_{sid}, C')$ for all substitutions σ and all terms C'. We distinguish two cases
  - $C = (M_{sid}, C')$. $\tilde{C}$ cannot be of form (b) since $C =_{\mathrm{E}} \tilde{C}$. Hence $\tilde{C} = (M_{sid}, \tilde{C}')$ and $C' =_{\mathrm{E}} \tilde{C}$ by Definition 5 (vi) (natural symbolic model). Therefore $untag(C) = C' =_{\mathrm{E}} \tilde{C}' = untag(\tilde{C})$.
  - Otherwise, C is of form (b). Then $\tilde{C}$ cannot be of form (a) since $C =_{\mathrm{E}} \tilde{C}$. We thus have $untag(C) = (n_{sid}, C) =_{\mathrm{E}} (n_{sid}, \tilde{C}) = untag(\tilde{C})$.

  We find

  $$\begin{aligned} &P \equiv \mathcal{C}[\overline{C}\langle T\rangle.R|\tilde{C}(x).\tilde{R}] \to \mathcal{C}[R|\tilde{R}\{T/x\}] \equiv P' \\ \Rightarrow &untag(P) \equiv untag(\mathcal{C})[\overline{untag(C)}\langle T\rangle.untag(R)|untag(\tilde{C})(x).untag(\tilde{R})] \\ &\overset{(*)}{\to} untag(\mathcal{C})[untag(R)|untag(\tilde{R})\{T/x\}] =: P^* \end{aligned}$$

  $(*)$ since $untag(C) =_{\mathrm{E}} untag(\tilde{C})$. Due to the claim above $P^* \approx untag(P')$ which proves the COMM case.

- LET-THEN: We have $P \equiv \mathcal{C}[\text{let } x = D \text{ in } R \text{ else } \tilde{R}] \to \mathcal{C}[R\{T/x\}] \equiv P'$. In contrast to Lemma 27 the evaluation of the destructor may not lead to a term T with $n_{sid} \in fn(T)$ here if $x \in fv(R)$ since we required P' to be sid-sensitive. (Otherwise, if $x \notin fv(R)$, we obviously have $untag(R)\{T/x\} = untag(R\{T/x\})$.) Thus

  $$\begin{aligned} &untag(P) \equiv untag(\mathcal{C})[\text{let } x = D \text{ in } untag(R) \text{ else } untag(\tilde{R})] \\ &\qquad \to untag(\mathcal{C})[untag(R)\{T/x\}] =: P^* \overset{(*)}{\approx} untag(\mathcal{C}[R\{T/x\}]) = untag(P') \end{aligned}$$

  (*) due to the claim above. This proves the LET-THEN case.

Since *untag* dos not affect the REPL and LET-ELSE cases these can be handled exactly like the REPL case in the proof of Lemma 27. □

Lemma 29. *$\sim_{\mathcal{S}_{sid}}$ is an $\mathcal{S}_{sid}$-n_{sid}-bisimulation*

Proof. Let $(P, Q) \in \sim_{\mathcal{S}_{sid}}$. We show the three points of an $\mathcal{S}_{sid}$-n_{sid}-simulation.

- $P \downarrow_C$: We have $P \downarrow_C$ iff $P \downarrow_{\hat{C}}$ for a channel identifier $\hat{C} =_{\mathrm{E}} C$ which occurs in P and thus follows the grammar from Figure 2.4. Since $P \sim_{\mathcal{S}_{sid}} Q$: $untag(P) \approx untag(Q)$ holds by definition. Since $P \downarrow_{\hat{C}}$ we have $untag(P) \downarrow_{untag(\hat{C})}$ and thus $untag(Q) =: \hat{Q}_1 \rightarrow \ldots \rightarrow \hat{Q}_n \downarrow_{untag(\hat{C})}$ for some $n \in \mathbb{N}$ and processes Q_i, $i \in \{1, \ldots, n\}$. By Lemma 23 $\hat{Q}_1 = untag(Q)$ is n_{sid}-good. By Lemma 27 we get a sequence of n_{sid}-good processes $\hat{Q}'_1 \rightarrow \ldots \rightarrow \hat{Q}'_n$ with $\hat{Q}'_1 = \hat{Q}_1$, $\hat{Q}'_i \equiv_{\mathrm{E}} \hat{Q}_i$ and $tag(\hat{Q}'_1) \rightarrow \ldots \rightarrow tag(\hat{Q}'_n)$. Since $\hat{Q}'_1 = \hat{Q}_1 = untag(Q)$ we have $tag(\hat{Q}'_1) = Q$ by Lemma 25. Furthermore, $\hat{Q}'_n \equiv_{\mathrm{E}} \hat{Q}_n \downarrow_{untag(\hat{C})}$ implies $\hat{Q}'_n \downarrow_{untag(\hat{C})}$ (see Footnote 5) and $tag(\hat{Q}'_n) \downarrow_{tag(untag(\hat{C}))}$. Since $\hat{C}$ is a term according to Figure 2.4 we have $tag(untag(\hat{C})) = \hat{C}$ ($=_{\mathrm{E}} C$) (see Lemma 25). Hence $Q = tag(\hat{Q}'_1) \rightarrow^* tag(\hat{Q}'_n) \downarrow_C$.
- $P \rightarrow P'$ with $n_{sid} \notin fn(P') \cup bn(P')$: According to Lemma 28 we find P^* such that $untag(P) \rightarrow P^* \approx untag(P')$. Since $P \sim_{\mathcal{S}_{sid}} Q$ we also have $untag(Q) =: \hat{Q}_1 \rightarrow \ldots \rightarrow \hat{Q}_n \approx P^*$. Analogously to the previous part we find some n_{sid}-good $\hat{Q}'_n$ such that $Q \rightarrow^* tag(\hat{Q}'_n)$ and $\hat{Q}'_n \equiv_{\mathrm{E}} \hat{Q}_n$. By Lemma 24 we have $untag(tag(\hat{Q}'_n)) \approx \hat{Q}'_n$ ($\hat{Q}'_n$ is closed). Thus $untag(tag(\hat{Q}'_n)) \approx \hat{Q}'_n \equiv_{\mathrm{E}} \hat{Q}_n \approx P^* \approx untag(P')$ which implies $untag(tag(\hat{Q}'_n)) \approx untag(P')$ since $\equiv_{\mathrm{E}}$ entails $\approx$ by Lemma 4 (iv). Hence $Q \rightarrow^* tag(\hat{Q}'_n)$ and $P' \sim_{\mathcal{S}_{sid}} tag(\hat{Q}'_n)$.
- Assume $P \sim_{\mathcal{S}_{sid}} Q$ and let $R \in \mathcal{S}_{sid}$ be a process and $\underline{a}$ names. We have $untag(P) \approx untag(Q)$ by definition of $\sim_{\mathcal{S}_{sid}}$ and $\approx$ is closed under the application of evaluation contexts. Hence $untag(\nu\underline{a}.(P \mid R)) = \nu\underline{a}.(untag(P)|untag(R)) \approx \nu\underline{a}.(untag(Q)|untag(R)) = untag(\nu\underline{a}.(Q|R))$. Thus, by definition of $\sim_{\mathcal{S}_{sid}}$, $\nu\underline{a}.(P|R) \sim_{\mathcal{S}_{sid}} \nu\underline{a}.(Q|R)$.

Since $\sim_{\mathcal{S}_{sid}}$ is symmetric it is an $\mathcal{S}_{sid}$-n_{sid}-bisimulation.

□

Lemma 30. *Let P and Q be closed processes and M be an arbitrary closed term. Then $P \approx Q \Rightarrow P(\!(M)\!) \approx Q(\!(M)\!)$.*

Proof. Fix a name $n_{sid} \notin (fn(M) \cup fn(P) \cup bn(P) \cup fn(Q) \cup bn(Q))$and $M_{sid} := M$. Remember that all lemmas in this section were proven for an arbitrary fixed M_{sid} with $n_{sid} \notin fn(M_{sid})$. Now P, Q are n_{sid}-good and $P(\!(M_{sid})\!) = tag(P)$ and $Q(\!(M_{sid})\!) = tag(Q)$. By Lemmas 26,29: $tag(P) \approx tag(Q)$. Hence $P(\!(M)\!) = P(\!(M_{sid})\!) \approx Q(\!(M_{sid})\!) = Q(\!(M)\!)$. □

Lemma 31. *Let P and Q be processes and M be a term with $fv(M) \cap (bv(P) \cup bv(Q)) = \varnothing$. Then $P \approxeq Q \Rightarrow P(\!(M)\!) \approxeq Q(\!(M)\!)$.*

Proof. For all closing substitutions σ we have $P \approxeq Q \Rightarrow P\sigma \approx Q\sigma$. By Lemma 30 we have $P\sigma(\!(M\sigma)\!) \approx Q\sigma(\!(M\sigma)\!)$ for the closed processes $P\sigma$ and $Q\sigma$ and the closed term $M\sigma$. This entails $P(\!(M)\!)\sigma \approx Q(\!(M)\!)\sigma$ since $fv(M) \cap (bv(P) \cup bv(Q)) = \varnothing$. Therefore $P(\!(M)\!) \approxeq Q(\!(M)\!)$. □

Proof of Lemma 18. By Lemma 31 $P(\!(x)\!) \approxeq Q(\!(x)\!)$. According to Definition 17 $!!_x P = C^{x,\underline{np}}_{\mathit{SID}}[P(\!(x)\!)]$ for some names $\underline{np} \cap \mathit{fn}(P) = \varnothing$ and $!!_x Q = C^{x,\underline{nq}}_{\mathit{SID}}[Q(\!(x)\!)]$ for some names $\underline{nq} \cap \mathit{fn}(Q) = \varnothing$. Let $\underline{n}$ be names such that $\underline{n} \cap (\mathit{fn}(P) \cup \mathit{fn}(Q)) = \varnothing$ and $|\underline{n}| \geq \max(|\underline{np}|, |\underline{nq}|)$. We have

$$C^{x,\underline{np}}_{\mathit{SID}}[P(\!(x)\!)] \equiv C^{x,\underline{n}}_{\mathit{SID}}[P(\!(x)\!)] \overset{(*)}{\approxeq} C^{x,\underline{n}}_{\mathit{SID}}[Q(\!(x)\!)] \equiv C^{x,\underline{nq}}_{\mathit{SID}}[Q(\!(x)\!)]$$

(*) since $P(\!(x)\!) \approxeq Q(\!(x)\!)$ and $\approxeq$ is closed under the application of contexts (Lemma 1). Therefore $!!_x P \approxeq !!_x Q$. □

Note that Lemma 18 also implies $P \approxeq Q \Rightarrow !!P \approxeq !!Q$.

Lemma 32. *Let P be a process and n be a name that occurs only in channel identifiers in P. Then $\nu n.!!_x P \approxeq !!_x \nu n.P$ for all variables $x \notin \mathit{bv}(P)$.*

Proof. First, we observe that instances of P with distinct tags cannot communicate with each other. This can be formalized by the following

Claim.

Let $\mathit{id}, \mathit{id}' \in \mathit{SID}$ be distinct IDs and P, Q arbitrary processes. Then $P(\!(\mathit{id})\!) \rightarrow^* \updownarrow_C$ and $Q(\!(\mathit{id}')\!) \rightarrow^* \updownarrow_{C'}$ for terms C, C' implies $C \neq_{\mathrm{E}} C'$. Proof: By Definition 11 every channel identifier in $P(\!(\mathit{id})\!)$ is of the form (id, X) for some term X. Analogously, every channel identifier in $Q(\!(\mathit{id}')\!)$ is of the form (id', Y). Towards contradiction we assume $C = (\mathit{id}, X) =_{\mathrm{E}} (\mathit{id}', Y) = C'$. Then, by Definition 5 (vi) (natural symbolic model), we have $\mathit{id} =_{\mathrm{E}} \mathit{id}'$. However, $\mathit{id} \neq_{\mathrm{E}} \mathit{id}'$ is required for all pairs of distinct IDs $\mathit{id}, \mathit{id}' \in \mathit{SID}$. This proves the claim.

It is now easy to check that

$$\begin{aligned}\mathcal{R} := \{(&\mathcal{C}[\nu n.(P_1(\!(\mathit{id}_1)\!)| \dots |P_\ell(\!(\mathit{id}_\ell)\!)| \prod_{x \in S} P(\!(x)\!))],\\ &\mathcal{C}[\nu n.P_1(\!(\mathit{id}_1)\!)| \dots |\nu n.P_\ell(\!(\mathit{id}_\ell)\!)| \prod_{x \in S} \nu n.P(\!(x)\!)]) :\\ &P_1, \dots, P_\ell \text{ processes where } n \text{ occurs only in channel identifiers},\\ &\mathit{id}_1, \dots, \mathit{id}_l \subseteq \mathit{SID} \setminus S \text{ are distinct, } S \subseteq \mathit{SID} \text{ and } \mathcal{C} \text{ evaluation context}\}\end{aligned}$$

is a bisimulation and thereby prove the lemma. Although the P_i in $\mathcal{R}$ are formally arbitrary processes that contain n only in channel identifiers, they intuitively allow to represent the running instances of P. Note that the claim above holds for any pair $P_i(\!(\mathit{id}_i)\!)$, $P_j(\!(\mathit{id}_j)\!)$ with $i \neq j$. Intuitively, since n occurs only in channel identifiers and thus is never transmitted, no context can tell the difference between a private n that is shared among all instances and an n individual n for each instance. □

Lemma 33. *Let P and Q be processes. Then $!!_x(P|Q) \approxeq !!_x P | !!_x Q$ for all variables $x \notin \mathit{bv}(P) \cup \mathit{bv}(Q)$.*

Proof. We use the semantics of product processes (see Definition 7) for this proof. By Definition 12 and Definition 17 we have $!!_x R \approxeq \prod_{x \in \mathit{SID}} R(\!(x)\!)$ for any process R. Let σ be a closing substitution for $!!_x P$ and $!!_x Q$ (i.e., $\mathit{fv}(P(\!(x)\!)\sigma), \mathit{fv}(Q(\!(x)\!)\sigma) \subseteq \{x\}$). We set $\Pi_P(X) := \prod_{x \in X} P(\!(x)\!)\sigma$ for arbitrary $X \subseteq \mathit{SID}$ and $\sum_P(X) := \sum_{x \in X} P(\!(x)\!)\sigma =$

$P(\!(x_1)\!)\sigma | \ldots | P(\!(x_\ell)\!)\sigma$ for finite $X = \{x_1, \ldots, x_\ell\} \subseteq \mathit{SID}$. Analogously $\Pi_Q(X)$, $\sum_Q(X)$ and $\Pi_{PQ}(X) := \prod_{x \in X} (P(\!(x)\!)\sigma | Q(\!(x)\!)\sigma)$. We then define the relation $\mathcal{R}$:

$$\mathcal{R} := \{(\mathcal{C}[\sum_P(S_P) \mid \sum_Q(S_Q) \mid \Pi_{PQ}(S_{PQ})],\ \mathcal{C}[\Pi_P(S_{PQ} \cup S_P) \mid \Pi_Q(S_{PQ} \cup S_Q)]) :$$
$$\mathcal{C} \text{ evaluation context}, S_P, S_Q, S_{PQ} \subseteq \mathit{SID}, S_P \cap S_{PQ} = \emptyset, S_Q \cap S_{PQ} = \emptyset\}$$

closed under structural equivalence. Note that

$$\left(\prod_{x \in \mathit{SID}} (P(\!(x)\!)\sigma | Q(\!(x)\!)\sigma),\ \prod_{x \in \mathit{SID}} P(\!(x)\!)\sigma \mid \prod_{x \in \mathit{SID}} Q(\!(x)\!)\sigma \right) \in \mathcal{R}$$

for $S_P := \emptyset$, $S_Q := \emptyset$ and $S_{PQ} := \mathit{SID}$ which proves this lemma if $\mathcal{R} \subseteq \approx$. Therefore, we show the three points of a simulation for $\mathcal{R}$ and $\mathcal{R}^{-1}$ respectively. First, we show that $\mathcal{R}$ is a simulation. For $(A, B) \in \mathcal{R}$:

1. $A \downarrow_C$: Product processes do not emit on channels. Three cases remain:
   a) If $\mathcal{C}[\mathbf{0}] \downarrow_C$, then $B \downarrow_C$.
   b) If $P(\!(id)\!)\sigma \downarrow_C$ for some $id \in S_P$, then B can spawn the instance $P(\!(id)\!)\sigma$ from $\Pi_P(S_{PQ} \cup S_P)$ and then emit on C. Hence $B \rightarrow\downarrow_C$.
   c) Analogously for $Q(\!(id)\!)\sigma \downarrow_C$ for some $id \in S_Q$.

   Hence $A \downarrow_C$ entails $B \rightarrow^*\downarrow_C$.
2. $A \rightarrow A'$: We distinguish two cases:
   a) $\rightarrow$ follows the IREPL rule: Then $\rightarrow$ spawns a new instance with id id from $\Pi_{PQ}(S_{PQ})$: We set $\mathcal{C}'[\Box] := \mathcal{C}[P(\!(id)\!)\sigma \mid Q(\!(id)\!)\sigma \mid \Box]$ and $S'_{PQ} := S_{PQ} \setminus \{id\}$. Hence we have $A \rightarrow \mathcal{C}'[\sum_P(S_P) \mid \sum_Q(S_Q) \mid \Pi_{PQ}(S'_{PQ})]$. Additionally, we observe $B \equiv \mathcal{C}[\Pi_P(S_{PQ} \cup S_P) \mid \Pi_Q(S_{PQ} \cup S_Q)] \rightarrow\rightarrow \mathcal{C}'[\Pi_P(S'_{PQ} \cup S_P) \mid \Pi_Q(S'_{PQ} \cup S_Q)]$ by spawning $P(\!(id)\!)\sigma$ from $\Pi_P(S_{PQ} \cup S_P)$ and $Q(\!(id)\!)\sigma$ from $\Pi_Q(S_{PQ} \cup S_Q)$. We have $(\mathcal{C}'[\sum_P(S_P) \mid \sum_Q(S_Q) \mid \Pi_{PQ}(S'_{PQ})], \mathcal{C}'[\Pi_P(S'_{PQ} \cup S_P) \mid \Pi_Q(S'_{PQ} \cup S_Q)]) \in \mathcal{R}$.
   b) $\rightarrow$ follows a rule from Figure 2.3: Then we distinguish two cases:
      i. If we have $\mathcal{C}[\mathbf{0}] \rightarrow \mathcal{C}'[\mathbf{0}]$, $\rightarrow$ translates canonically to $\mathcal{C}$ in $B \rightarrow B'$ such that $(A', B') \in \mathcal{R}$.
      ii. Otherwise, $\rightarrow$ affects instances from $\sum_P(S_P) \mid \sum_Q(S_Q)$. We remove the ids of the affected instances from S_P and S_Q yielding sets S'_P and S'_Q and define a context $\mathcal{C}'$ (including the affected instances) such that $A \rightarrow \mathcal{C}'[\sum_P(S'_P) \mid \sum_Q(S'_Q) \mid \Pi_{PQ}(S_{PQ})]$. We now spawn the corresponding instances in B first and then mimic $\rightarrow$ exactly yielding $B \rightarrow^* \mathcal{C}'[\Pi_P(S_{PQ} \cup S'_P) \mid \Pi_Q(S_{PQ} \cup S'_Q)]$. We have $(\mathcal{C}'[\sum_P(S'_P) \mid \sum_Q(S'_Q) \mid \Pi_{PQ}(S_{PQ})], \mathcal{C}'[\Pi_P(S_{PQ} \cup S'_P) \mid \Pi_Q(S_{PQ} \cup S'_Q)]) \in \mathcal{R}$.
3. By definition $\mathcal{R}$ is closed under the application of evaluation contexts.

Now we show that $\mathcal{R}^{-1}$ is a simulation. For $(A, B) \in \mathcal{R}^{-1}$:

1. $A \downarrow_C$: Since product processes do not emit on channels we have $\mathcal{C}[\mathbf{0}] \downarrow_C$ and thus $B \downarrow_C$.
2. $A \rightarrow A'$: We distinguish two cases:
   a) $\rightarrow$ follows the IREPL rule: We distinguish four cases:
      i. A new instance $P(\!(id)\!)\sigma$ is spawned from $\Pi_P(S_{PQ} \cup S_P)$ with $id \in S_P$: We define the context $\mathcal{C}'[\Box] := \mathcal{C}[P(\!(id)\!)\sigma \mid \Box]$, $S'_P := S_P \setminus \{id\}$ and have $A \rightarrow \mathcal{C}'[\Pi_P(S_{PQ} \cup S'_P) \mid \Pi_Q(S_{PQ} \cup S_Q)]$ and $B \equiv$

$\mathcal{C}'[\sum_P(S'_P) \mid \sum_Q(S_Q) \mid \Pi_{PQ}(S_{PQ})]$. Hence $(\mathcal{C}'[\Pi_P(S_{PQ}\cup S'_P) \mid \Pi_Q(S_{PQ}\cup S_Q)], \mathcal{C}'[\sum_P(S'_P) \mid \sum_Q(S_Q) \mid \Pi_{PQ}(S_{PQ})] \in \mathcal{R}^{-1}$.

ii. A new instance $Q(\!(id)\!)\sigma$ is spawned from $\Pi_Q(S_{PQ}\cup S_Q)$ with $id \in S_Q$: Analogous to the previous case.

iii. A new instance $P(\!(id)\!)\sigma$ is spawned from $\Pi_P(S_{PQ} \cup S_P)$ with $id \in S_{PQ}$: We define the context $\mathcal{C}'[\Box] := \mathcal{C}[P(\!(id)\!)\sigma \mid \Box]$, $S'_{PQ} := S_{PQ} \setminus \{id\}$, $S'_Q := S_Q\cup\{id\}$ and have $A \to \mathcal{C}'[\Pi_P(S'_{PQ}\cup S_P) \mid \Pi_Q(S'_{PQ}\cup S'_Q)]$. Note that $S_{PQ} \cup S_Q = S'_{PQ} \cup S'_Q$. In B we spawn $P(\!(id)\!)\sigma \mid Q(\!(id)\!)\sigma$ from $\Pi_{PQ}(S_{PQ})$ and have $B \to \mathcal{C}'[\sum_P(S_P) \mid \sum_Q(S'_Q) \mid \Pi_{PQ}(S'_{PQ})]$. Hence $(\mathcal{C}'[\Pi_P(S'_{PQ} \cup S_P) \mid \Pi_Q(S'_{PQ} \cup S'_Q)], \mathcal{C}'[\sum_P(S_P) \mid \sum_Q(S'_Q) \mid \Pi_{PQ}(S'_{PQ})]) \in \mathcal{R}^{-1}$.

iv. A new instance $Q(\!(id)\!)\sigma$ is spawned from $\Pi_Q(S_{PQ} \cup S_Q)$ with $id \in S_{PQ}$: Analogous to the previous case.

b) $\to$ follows a rule from Figure 2.3: Then we basically have $\mathcal{C}[\mathbf{0}] \to \mathcal{C}'[\mathbf{0}]$ which translates canonically to $\mathcal{C}$ in $B \to B'$ such that $(A', B') \in \mathcal{R}$.

3. By definition $\mathcal{R}$ is closed under the application of evaluation contexts.

This shows that $\mathcal{R}$ is a bisimulation and hence $\mathcal{R} \subseteq \approx$. □

Alternative definitions of !!.

Of course, our definition of $!!P$ is not the only possible definition of a replication with session ids. For example, one might try to define $!!P$ in such a way that an instance of P is spawned for arbitrary terms as sessions id, not only terms in some fixed set *SID*. In particular, a fresh name could then be used as session id which is not possible with our modeling. (Then, of course, the set of processes to which !! may be applied should be restricted to processes which wait for an input before doing anything. Otherwise processes could spawn spontaneously that use some other process' fresh names as session ids.)

Any definition of !! that satisfies Lemmas 17, 18, 32, and 33 would lead to the same composition theorem. (If that definition !! is applicable only to a certain set $\mathbf{P}$ of processes, we additionally need that $\mathbf{P}$ is closed under parallel composition, restrictions, renaming, and !!, and that the definition of $\leq$ (Definition 10) is with respect to simulators in $\mathbf{P}$.)

The composition theorem.

We can now state and prove the composition theorem. It says that if $P \leq Q$, we can restrict the IO-names, compose in parallel with processes that have disjoint NET-names, rename names (as long as NET- and IO-names are not interchanged), and perform concurrent composition.

Theorem 1 (Composition Theorem)**.** *Let P, Q be processes with $P \leq Q$. Then*

(i) For any list of names $\underline{io} \subseteq \mathrm{IO}$ we have $\nu\underline{io}.P \leq \nu\underline{io}.Q$.

(ii) For any process R with $(fn(R) \cap (fn(P) \cup fn(Q))) \subseteq \mathrm{IO}$ we have $P|R \leq Q|R$.

(iii) For any permutation $\psi : \mathrm{NET} \to \mathrm{NET}$ we have $P\psi \leq Q$ and $P \leq Q\psi$.

(iv) For any permutation $\psi : \mathrm{IO} \to \mathrm{IO}$ we have $P\psi \leq Q\psi$.

(v) If Q is a NET-stable process, $!!_x P \leq !!_x Q$ for all variables $x \notin bv(P) \cup bv(Q)$.

Proof. In the following, let $(S, \varphi, \underline{n})$ be as in Definition 10. (They exist because $P \leq Q$.)

(i) $P \approx \nu\underline{n}.(Q\varphi|S) \overset{(*)}{\Rrightarrow} \nu\underline{io}.P \approx \nu\underline{io}.\nu\underline{n}.(Q\varphi|S) \overset{(**)}{\approx} \nu\underline{n}.((\nu\underline{io}.Q)\varphi|S)$

$(*)$ since $\approx$ is closed under the application of evaluation contexts.

$(**)$ since neither S nor φ contain names from IO

(ii) W.l.o.g. we can assume $\mathit{fn}(R) \cap \underline{n} = \emptyset$ and that φ is the identity on $(\mathit{fn}(R) \cup \mathit{bn}(R)) \cap \mathrm{NET}$. These assumptions guarantee $(*)$ in the upcoming equations. $P \approx \nu\underline{n}.(Q\varphi|S) \Rightarrow P|R \approx \nu\underline{n}.(Q\varphi|S)|R \overset{(*)}{\approx} \nu\underline{n}.((Q|R)\varphi|S)$

(iii) $P \approx \nu\underline{n}.(Q\varphi|S) \Rightarrow P\psi \approx (\nu\underline{n}.(Q\varphi|S))\psi \equiv \nu\underline{n}\psi.(Q(\psi \circ \varphi)|S\psi)$. Therefore, with $(S\psi, \psi \circ \varphi, \underline{n}\psi)$ as simulator, we have $P\psi \leq Q$. With $(S, \varphi \circ \psi^{-1}, \underline{n})$ we have $P \leq Q\psi$.

(iv) $P \approx \nu\underline{n}.(Q\varphi|S) \Rightarrow P\psi \approx (\nu\underline{n}.(Q\varphi|S))\psi \equiv \nu\underline{n}.(Q(\varphi \circ \psi)|S)$ since S, φ and $\underline{n}$ do not contain IO names and thus are not affected by $\psi : \mathrm{IO} \to \mathrm{IO}$.

(v) Note that $Q\varphi$ is NET-stable since Q is NET-stable. Then $P \approx \nu\underline{n}.(Q\varphi|S)$ entails

$$\begin{aligned} !!_x P &\approx\ !!_x \nu\underline{n}.(Q\varphi|S) && \text{(by Lemma 18)} \\ &\approx \nu\underline{n}.!!_x(Q\varphi|S) && \text{(by Lemma 32 since } Q\varphi|S \text{ NET-stable)} \\ &\approx \nu\underline{n}.(!!_x(Q\varphi)|!!_x S) && \text{(by Lemma 33)} \\ &\equiv \nu\underline{n}.((!!_x Q)\varphi|!!_x S) && \text{(by Lemma 17)} \end{aligned}$$

Thus $(!!_x S, \varphi, \underline{n})$ is a proper simulator for $!!_x P \leq\ !!_x Q$. □

2.5 Property preservation

Besides secure composition, the second salient property of the UC framework is the fact that security properties of the ideal functionality $\mathcal{F}$ automatically carry over to any protocol emulating $\mathcal{F}$. For example, a secure channel functionality that takes a message m from Alice and gives it directly to Bob will obviously have the property that m stays secret. Then, if π UC-emulates $\mathcal{F}$, any message given to π will also stay secret. A similar property preservation law holds in our case, the following theorem formalizes it:

Theorem 2 (Property preservation)**.** *Let P, Q be NET-stable processes with $P \leq Q$. Let E_1 and E_2 be contexts whose holes are protected only by parallel compositions $(|)$, restrictions (ν), and indexed replications $(!!_x)$. Assume that E_1, E_2 do not contain NET-names (neither bound nor free). Assume that the number of $!!_x$ (possibly with different x) over the hole is the same in E_1 and E_2.*

If $E_1[Q] \approx E_2[Q]$, then $E_1[P] \approx E_2[P]$.

Proof. Let b denote the number of $!!_x$ over the hole of E_1, E_2. We write $!!^b S$ for $b \geq 0$ applications of $!!$ to S.

Since $P \leq Q$, there are $S, \varphi, \underline{n}$ with $P \approx \nu\underline{n}.(Q\varphi|S)$ and S closed and NET-stable, and $\mathrm{IO} \cap \mathit{fn}(A) = \emptyset$, $\varphi : \mathrm{NET} \to \mathrm{NET}$ a bijection and $\underline{n}$ a list of names $\underline{n} \subseteq \mathrm{NET}$. Without loss of generality, we can assume that $\underline{n} \cap \mathit{fn}(E_1, E_2) = \underline{n} \cap \mathit{bn}(E_1, E_2) = \emptyset$. For $i = 1, 2$, we have

$$\begin{aligned} E_i[P] &\overset{\text{(i)}}{\approx} E_i[\nu\underline{n}.(Q\varphi|S)] \\ &\overset{\text{(ii)}}{\approx} \nu\underline{n}.E_i[(Q\varphi|S)] \\ &\overset{\text{(iii)}}{\approx} \nu\underline{n}.(E_i[Q\varphi]|!!^b S) \\ &\overset{\text{(iv)}}{=} \nu\underline{n}.(E_i[Q]\varphi|!!^b S). \end{aligned}$$

```
free netscstart, netnotify, netdeliver, n1, n2.
fun empty/0.

let FSC = in(netstart,y); in(ioA,x);
          ( out(netnotify,empty) | in(netdeliver,z); out(ioB,x) ).

process new ioA; new ioB; out(ioA,choice[n1,n2]) | in(ioB,z) | FSC
```

Figure 2.6: Proverif code for showing $E_1[\mathcal{F}_{SC}] \approx E_2[\mathcal{F}_{SC}]$ in Lemma 34 (`prop-pres.pv`, see [29]).

Here (i) uses Lemma 1.

And (ii) uses that the names $\underline{n}$ do not occur in E_i, the rules NEW-C and NEW-PAR from Figure 2.2, and Lemma 32 for swapping $!!_x$ in E_i and the names $\underline{n}$ (the preconditions of Lemma 32 are fulfilled because $\underline{n}$ are NET-names and thus do not occur in E_i).

And (iii) uses that the names in E_i (IO-names only) and the names in S (NET-names only) are disjoint, as well as Lemma 33 for moving S over a $!!$ in E_i. (Lemma 33 guarantees $!!_x(R|S) \approx\, !!_x R|!!_x S$, this is why S accumulates on $!!_x$ for each $!!_x$ over the hole of E_i. Since S is closed, we can drop the x from $!!_x$.)

And (iv) uses that E_i does not contain NET-names (bound or free) while φ is a substitution on NET-names.

Furthermore, since $\approxeq$ is closed under renaming of free names, and under application of contexts (Lemma 1), from $E_1[Q] \approxeq E_2[Q]$ it follows that $\nu\underline{n}.(E_1[Q]\varphi|!!^b S) \approxeq \nu\underline{n}.(E_2[Q]\varphi|!!^b S)$ and hence $E_1[P] \approxeq E_2[P]$. □

Thus, any security property that can be expressed by an indistinguishability game of the form "$E_1[P] \approxeq E_2[P]$" with E_1, E_2 as in the theorem will carry over from the ideal functionality Q to the protocol P, given $P \leq Q$. Note that even many trace based properties can be expressed in such a way. E.g., if we want to say that $E_1[P]$ does not raise an event *bad* (modeled by emitting on a special channel), we just define E_2 to be like E_1, but without the event. Then $E_1[P] \approxeq E_2[P]$ implies that $E_1[P]$ does not raise the event.

Example: Strong secrecy.

We illustrate the use of this theorem with an example. Consider the secure channel functionality:

Definition 26 (Secure channel)**.**

$$\mathcal{F}_{SC} := net_{scstart}().io_A(x).(\overline{net_{notify}}\langle\rangle \mid net_{deliver}().\overline{io_B}\langle x\rangle)$$

We want to show:

Lemma 34. *If $P \leq \mathcal{F}_{SC}$, then P has strong secrecy in the following sense: We have $P_1 \approx P_2$ where $P_i := \nu io_A io_B.\overline{io_A}\langle n_i\rangle|io_B()|P$.*

Proof. Let $E_i := \nu io_A io_B.\overline{io_A}\langle n_i\rangle|io_B()|\square$. We use Proverif to show that $E_1[\mathcal{F}_{SC}] \approx E_2[\mathcal{F}_{SC}]$. The Proverif code is given in Figure 2.6.

By Theorem 2 (and using that $\approx$ and $\approxeq$ coincide for closed processes), we have $P_1 = E_1[\mathcal{F}_{SC}] \approx E_2[\mathcal{F}_{SC}] = P_2$. □

Anonymity properties are modeled very similarly, except that instead of different payloads n_1, n_2, different user ids are provided to the two processes.

Example: Unlinkability.

The next example is strong unlinkability [5]. This property requires that the adversary cannot distinguish whether every user runs only one session of a protocol, or whether every user runs many sessions. Formally: $!\nu id.!\nu sid.P \approx !\nu id.\nu sid.P$ if we assume that P contains free names id, sid for the user id and the session id. At a first glance, such a property seems to be excluded by the restriction of Theorem 2 that E_1, E_2 may not have a ! over their hole. This is, however, not the case if protocol P (and the functionality Q) are modeled suitably, namely if P is already a multi-session protocol. For example, if P expects a pair of user id and session id on an IO-channel *init* for each session to be run, then strong unlinkability can be expressed as follows:

Definition 27. *A protocol P has* strong unlinkability *iff*

$$\nu init.(P|!\nu id.!\nu sid.\overline{init}\langle(id, sid)\rangle) \approx \nu init.(P|!\nu id.\nu sid.\overline{init}\langle(id, sid)\rangle).$$

Then Theorem 2 guarantees that if Q has strong unlinkability and $P \leq Q$, then P has strong unlinkability.

Notice that if we model a different session id mechanism, we also need a different definition. For example, if P and Q are constructed using the !! *operator, session ids will be part of the channel name (we would have channels such as $(sid, (id, init))$). The variant described above seems most realistic for unlinkability, though, because* !! *includes session ids in the clear in all network-messages, so constructing unlinkable protocols by concurrent composition of individual sessions using* !! *does not seem to work well.*

In Section 2.9 we show that the various restrictions in Theorem 2 are necessary. In particular, property preservation for contexts E_1, E_2 having a ! over their hole (instead of a !!) does not hold. The reasons are similar to those that forbid ! in the composition theorem (cf. Section 2.4). This is another indication that an operator like !! is more natural in this context.

2.6 Relation to Delaune-Kremer-Pereira

DKP-security.

As mentioned in the introduction, Delaune, Kremer, and Pereira [54] have already presented a variant of the UC model in the applied pi calculus. In this section, we describe the differences between their and our model, and why these differences are necessary to achieve stronger security results.

In [54], security is defined as follows:

Definition 28 (DKP-security)**.** *Let $\preceq$ (*observational preorder*) be the largest simulation (not bisimulation).*

Let P, Q be processes. Then $P \leq^{\mathsf{SS}} Q$ iff there exists a simulator S (a context) such that $P \preceq S[Q]$.

Here a simulator S is an evaluation context subject to certain conditions, see [54], notably that it only binds NET-names.

Notice that in this definition, the main difference to our definition is that P and $S[Q]$ do not have to be observationally equivalent, but only observationally preordered. (Also, the notion of the simulator S is somewhat different from ours, but not in essence.) The effect of this is that the simulator may introduce additional non-determinism. For example, in our model, if the protocol P can take one out of two actions, the simulator needs to simulate the appropriate action, he thus needs to figure out which of the two actions is taken. With respect to DKP-security, the simulator can just non-deterministically choose which action to take; the observational preorder takes care that the right action is taken in the right situation. This makes simulators for DKP-security much easier to construct and DKP-security into a considerably weaker notion.

DKP-security satisfies similar laws as our notion. In particular, $\leq^{\mathsf{SS}}$ is reflexive and transitive and it satisfies a composition theorem (which differs from ours mainly in that $P \leq^{\mathsf{SS}} Q \implies !P \leq^{\mathsf{SS}} !Q$ holds, no need to introduce !!). They do not state a property preservation theorem. We believe, though, that their DPK-security supports property preservation for certain kinds of trace properties.[10]

The problem with observational preorder.

We explain why we believe that a definition based on observational preorder instead of equivalence does not give sufficient security guarantees. We illustrate this by the following example. Consider a simple functionality that is supposed to model an insecure but anonymous channel:

$$\mathcal{F}_{anon} := io_A(x).\overline{net}\langle x\rangle | io_B(x).\overline{net}\langle x\rangle$$

Obviously, this functionality preserves anonymity about whether Alice or Bob sends a message (i.e., whether an input on io_A or io_B occurs). Formally: $\nu io_A io_B.(\overline{io_A}\langle T\rangle | \mathcal{F}_{anon}) \approx \nu io_A io_B.(\overline{io_B}\langle T\rangle | \mathcal{F}_{anon})$. (In fact, we even have $\equiv$.) Now consider a naive protocol in which Alice and Bob send their message over distinct channels net_A, net_B. Formally:

$$P := io_A(x).\overline{net_A}\langle x\rangle | io_B(x).\overline{net_B}\langle x\rangle$$

Obviously, P does not provide anonymity, it is easy to see that $\nu io_A io_B.(\overline{io_A}\langle T\rangle | P) \not\approx \nu io_A io_B.(\overline{io_B}\langle T\rangle | P)$. Consequently (Theorem 2), we have $P \not\leq \mathcal{F}_{anon}$ as we would expect since P gives less security than $\mathcal{F}_{anon}$.

On the other hand, with respect to DKP-security, P is considered as secure as $\mathcal{F}_{anon}$, i.e., $P \leq^{\mathsf{SS}} \mathcal{F}_{anon}$. We use the following simulator: $S := net(x).\overline{net_A}\langle x\rangle \mid net(x).\overline{net_B}\langle x\rangle \mid \square$. Then $P \preceq S[\mathcal{F}_{anon}]$ because S relays messages sent on net onto net_A or net_B, and the definition of $\preceq$ makes sure that the message is non-deterministically delivered on the right channel net_A or net_B. Hence $P \leq^{\mathsf{SS}} \mathcal{F}_{anon}$.

Lemma 35 (with non-rigorous proof)**.** $P \leq^{\mathsf{SS}} \mathcal{F}_{anon}$.

Proof. In this proof, we assume that Lemma 5 also holds for the calculus from [54]. Since that calculus is somewhat different from ours, this makes the present proof non-rigorous. (However, probably the proof of Lemma 5 can be easily adapted to the calculus of [54].)

[10] Probably a law of the following kind holds: Assume $P \leq^{\mathsf{SS}} Q$. Let $c \notin fv(P,Q)$, and E be a context satisfying certain properties. Then $E[Q] \not\downarrow_c \Longrightarrow E[P] \not\downarrow_c$. Compare with Theorem 2 which can deal with indistinguishability properties.

Then we have

$$\begin{aligned} P &\overset{(*)}{\approx} \nu net.(io_A(x).\overline{net}\langle x\rangle|net(x).\overline{net_A}\langle x\rangle) \\ &\qquad | \ \nu net.(io_B(x).\overline{net}\langle x\rangle|net(x).\overline{net_B}\langle x\rangle) \\ &\overset{(**)}{\preceq} io_A(x).\overline{net}\langle x\rangle|net(x).\overline{net_A}\langle x\rangle \\ &\qquad | \ io_B(x).\overline{net}\langle x\rangle|net(x).\overline{net_B}\langle x\rangle \\ &\equiv S[\mathcal{F}_{anon}] \text{ with } S := net(x).\overline{net_A}\langle x\rangle \mid net(x).\overline{net_B}\langle x\rangle \mid \Box. \end{aligned}$$

Here $(*)$ uses two applications of Lemma 5 (in the reverse direction), the first with $n := net$, $t := x$, $x := x$, and $Q := \overline{net_A}\langle x\rangle$, and the second with $n := net$, $t := x$, $x := x$, and $Q := \overline{net_B}\langle x\rangle$. And $(**)$ uses that $\nu c.P \preceq P$ ([53, Lemma 8]).

Since $\approx$ implies $\preceq$ and $\preceq$ is transitive, we have $P \preceq S[\mathcal{F}_{anon}]$. Furthermore, S is a valid simulator for DKP-security. Thus $P \leq^{\mathsf{SS}} \mathcal{F}_{anon}$. □

Thus, the security of a protocol in the sense of [54] does not imply that the protocol has the same anonymity properties as the ideal functionality. The same probably holds for other equivalence properties such as strong secrecy etc. We consider this a strong restriction of the notion and thus believe that a symbolic analogue to UC security should use observational equivalence or a similar notion of equivalence.

Why observational preorder?

The reader may wonder why [54] use observational preorder instead of observational equivalence, in particular since observational equivalence is the more direct analogue to the indistinguishability in the computational UC framework [36]. We explain the reasons as we understand them (this is based both on explanations in [54] and on our own insights while working on the current result), and due to what definitional decisions we managed to get around those reasons:

- It is not possible to model "relays". That is, if we have a process P that outputs on a channel c, then as a technical tool we might wish to construct a process R (a relay) that forwards all message on c to another channel c', i.e., we want $\nu c.(P|R) \approx P\{c'/c\}$. Unfortunately, such a relay does not seem to exist in the applied pi calculus. $R :=!c(x).\overline{c'}\langle x\rangle$ does not work. Consider, e.g., $P := \overline{c}\langle n\rangle.\overline{a}\langle n\rangle$. Then $\nu c.(P|R) \downarrow_a$ but $P\{c'/c\} \not\downarrow_a$. With respect to $\preceq$, however, we can have relays ($P\{c'/c\} \preceq \nu c.(P|R)$).

  Why are relays important? One reason is whether a dummy adversary exists. Such a dummy adversary is an adversary that forwards all messages on NET-channels from the protocol to the environment and vice versa. (So, essentially, a relay.) The existence of the dummy adversary is used implicitly or explicitly in most structural theorems (reflexivity, transitivity, concurrent composition). In fact, it seems that when using observational equivalence in [54], one would not even have reflexivity.

  We get around this problem by using a slightly different definition of adversaries/simulators (Definition 9). In our setting, a dummy can be trivially constructed as $(0, \varphi, \varnothing)$ where φ just renames the protocol's NET-channels to the NET-channels that the environment expects the messages on. This simple trick obviates the need for using relays in the construction of the dummy adversary.

- The second problem is that one does not get a composition theorem that guarantees $P \leq^{\mathsf{SS}} Q \implies !P \leq^{\mathsf{SS}} !Q$ when using observational equivalence. However, we believe that this is a natural limitation because we can show that property preservation does not even hold for equivalence-based security properties that replicate the protocol. Thus we cannot expect to get such a composition theorem and simultaneously have property preservation for equivalence properties. We get around this problem by defining a different notion of concurrent composition, using the !! operator (see Section 2.4).

- Finally, the non-existence of relays is a problem when proving the security of concrete protocols $P \leq \mathcal{F}$: A typical thing a simulator has to do is to take a message m on a NET-channel and somehow rewrite it (e.g., to $\mathit{enc}(k, m)$) before sending it on to the environment. This, of course, is a generalization of the concept of a relay. Thus, if relays are impossible, we can hardly expect to construct sensible simulators. This, however, is not true if we pay some attention in the definition of the functionality and obey the following guideline:

  > **Guideline:** When designing a functionality, use different names for all NET-channels and, whenever sending something on a NET-channel C, use $\overline{C}\langle T\rangle | P'$ instead of $\overline{C}\langle T\rangle.R$.

  In these cases, $R := !c(x).\overline{c'}\langle x\rangle$ will usually work as a relay (e.g., $\nu c.(P|R) \approx P\{c'/c\}$ for $P := \overline{c}\langle n\rangle | \overline{a}\langle n\rangle$).

2.7 Example: Secure channels

In this section we apply symbolic UC hands on. We illustrate how our results from Section 2.4 can be usefully applied in practice to construct a secure channel from the widely known NSL protocol and a PKI. Furthermore, when extending the secure channel to multiple sessions, we present an example for a joint state, i.e., multiple instances of one protocol that jointly use one instance of another functionality. While the original UC model of Canetti [36] requires an additional theorem to handle joint states [39], we can directly use !! in our case. We used Proverif[11] for our proofs as much as possible to show how it helps with the verification of various properties in the context of symbolic UC.

In this section, we only consider an example where we assume all parties to be honest (as the goal of secure channels is to protect from an outside adversary). For an example with corruption, see Section 2.8.

We first define the symbolic model used in this section. The constructors are: $\mathit{penc}/3$, $\mathit{pk}/1$, $\mathit{sk}/1$, $\mathit{senc}/3$, $(\cdot,\cdot)$, $\mathit{hash}/1$, and $\mathit{empty}/0$, representing public-key encryption, public and secret keys, symmetric encryption, pairs, hashing, and empty messages, respectively. Encryption has a third argument modeling randomness used for encrypting. More specifically, $\mathit{penc}(\mathit{pk}(k), m, r)$ models a public key encryption using key $\mathit{pk}(k)$, plaintext m, and randomness r, and $\mathit{senc}(k, m, r)$ a symmetric encryption using key k, plaintext m, and randomness r. We believe that senc without the additional randomness argument r would also work in our setting. However, we introduce this additional nonce to help Proverif, which can then better distinguish ciphertexts (e.g., the proof of `secchan-sc2.pv` fails without r due to Proverif's overapproximation technique). We have no equations in our theory.

[11] Version 1.86pl4

```
fun senc/3. (* senc(key,msg,rand) *)
reduc sdec(k,senc(k,m,r)) = m.
fun empty/0.
fun hash/1.
fun pk/1.
fun sk/1.
fun penc/3. (* penc(pk,msg,rand) *)
reduc pdec(sk(k),penc(pk(k),m,r)) = m.
reduc pkofsk(sk(k)) = pk(k).
reduc pkofenc(penc(p,m,r)) = p.
```

Figure 2.7: Key-exchange example: Proverif code for the symbolic model (`secchan-model.pv`, see [29])

Furthermore we have the destructors $\mathit{pdec}/2$, $\mathit{sdec}/2$, $\mathit{pkofsk}/1$, and $\mathit{pkofenc}/1$, modeling public-key decryption, symmetric decryption, extraction of a public key from a secret key, and extraction of a public key from a ciphertext. (The latter two are not needed in our protocols, but we provide them to make the adversary more realistic.) The behavior of the destructors is specified by the following rewrite rules:

$$\begin{aligned}\mathit{pdec}(\mathit{sk}(x), \mathit{penc}(\mathit{pk}(x), y, z)) &\to y\\ \mathit{sdec}(x, \mathit{senc}(x, y, z)) &\to y\\ \mathit{pkofsk}(\mathit{sk}(x)) &\to \mathit{pk}(x)\\ \mathit{pkofenc}(\mathit{penc}(x, y, z)) &\to x\end{aligned}$$

The Proverif code for this symbolic model is given in Figure 2.7.

2.7.1 Key exchange using NSL

With the symbolic model set up we next show how to tailor a UC-secure key exchange from NSL using a PKI functionality $\mathcal{F}_{\mathit{PKI}}$. Towards this goal we model the ideal key exchange functionality $\mathcal{F}_{\mathit{KE}}$, the PKI $\mathcal{F}_{\mathit{PKI}}$ and the NSL protocol based on $\mathcal{F}_{\mathit{PKI}}$ as follows:

Definition 29 (Key exchange functionality)**.**

$$\mathcal{F}_{\mathit{KE}} := \nu k.\mathit{net}_{\mathit{delA}}().\overline{\mathit{io}_{\mathit{ka}}}\langle k\rangle \mid \mathit{net}_{\mathit{delB}}().\overline{\mathit{io}_{\mathit{kb}}}\langle k\rangle$$

Definition 30 (Public key infrastructure functionality)**.**

$$\begin{aligned}\mathcal{F}_{\mathit{PKI}} := \nu k_a k_b.&\overline{\mathit{io}_{\mathit{pkeA}}}\langle(\mathit{sk}(k_a), \mathit{pk}(k_a), \mathit{pk}(k_b))\rangle\\ \mid\ &\overline{\mathit{io}_{\mathit{pkeB}}}\langle(\mathit{sk}(k_b), \mathit{pk}(k_a), \mathit{pk}(k_b))\rangle\\ \mid\ &\overline{\mathit{net}_{\mathit{pke}}}\langle(\mathit{pk}(k_a), \mathit{pk}(k_b))\rangle\end{aligned}$$

Definition 31 (Needham-Schroeder-Lowe)**.**

$$\begin{aligned}
\mathsf{NSL}_A :=\ & io_{pkeA}((x_{sk}, \_, x_{pk_B})).\nu n_a.\nu r_1.\\
& \quad \overline{net_{nslA}}\langle penc(x_{pk_B}, n_a, r_1)\rangle.net_{nslA}(x_c).\\
& \quad let\ (=n_a, x_{n_b}, =B) = pdec(x_{sk}, x_c)\ in\\
& \qquad \nu r_2.\overline{net_{nslA}}\langle penc(x_{pk_B}, x_{n_b}, r_2)\rangle.\\
& \qquad \overline{io_{ka}}\langle hash((n_a, x_{n_b}))\rangle\\
\mathsf{NSL}_B :=\ & io_{pkeB}((x_{sk}, x_{pk_A}, \_)).net_{nslB}(x_c).\\
& \quad let\ x_{n_a} = pdec(x_{sk}, x_c)\ in\\
& \qquad \nu n_b.\nu r.\overline{net_{nslB}}\langle penc(x_{pk_A}, (x_{n_a}, n_b, B), r)\rangle.\\
& \qquad net_{nslB}(x'_c).if\ n_b = pdec(x_{sk}, x'_c)\ then\\
& \qquad\ \ \overline{io_{kb}}\langle hash((x_{n_a}, n_b))\rangle\\
\mathsf{NSL} :=\ & \nu io_{pkeA} io_{pkeB}.(\mathsf{NSL}_A \mid \mathsf{NSL}_B \mid \mathcal{F}_{PKI})
\end{aligned}$$

The differences to the original NSL protocol [66] are: The original protocol includes A's identity in the first message, and the original protocol does not specify what to do with the nonces n_a, n_b, while we use them to derive a key $hash((n_a, n_b))$. Also, [66] also presents an extended version of the protocol that explicitly communicates with a server S for getting the keys for Alice and Bob. We could get this extended protocol by proving that this retrieval protocol implements $\mathcal{F}_{PKI}$, and then composing our NSL protocol with the retrieval protocol.

We can now state the first result of this section, namely that the NSL is a UC-secure realization of $\mathcal{F}_{KE}$.

Lemma 36. $\mathsf{NSL} \leq \mathcal{F}_{KE}$.

Proof. Let NSL'_A be NSL_A without the initial $io_{pkeA}((x_{sk}, \_, x_{pk_B}))$. NSL'_B analogously. And $\mathsf{NSL}''_A := \mathsf{NSL}'_A\{net_{delA}/io_{ka}, sk(k_a)/x_{sk}, pk(k_b)/x_{pk_B}\}$ and $\mathsf{NSL}''_B := \mathsf{NSL}'_B\{net_{delB}/io_{kb}, sk(k_b)/x_{sk}, pk(k_a)/x_{pk_A}\}$.

We have

$$\begin{aligned}
\mathsf{NSL} \equiv\ & \nu io_{pkeA} io_{pkeB} k_a k_b.\Big(io_{pkeA}((x_{sk}, \_, x_{pk_B})).\mathsf{NSL}'_A \mid io_{pkeA}((x_{sk}, x_{pk_A}, \_)).\mathsf{NSL}'_B\\
& \quad \mid \overline{io_{pkeA}}\langle (sk(k_a), pk(k_a), pk(k_b))\rangle \mid \overline{io_{pkeB}}\langle (sk(k_b), pk(k_a), pk(k_b))\rangle\\
& \quad \mid \overline{net_{pke}}\langle (pk(k_a), pk(k_b))\rangle\Big)\\
\overset{\text{(v)}}{\approx}\ & \nu k_a k_b.\Big(\text{let } (x_{sk}, \_, x_{pk_B}) = (sk(k_a), pk(k_a), pk(k_b)) \text{ in } \mathsf{NSL}'_A\\
& \quad \mid \text{let } (x_{sk}, x_{pk_A}, \_) = (sk(k_b), pk(k_a), pk(k_b)) \text{ in } \mathsf{NSL}'_B \mid \overline{net_{pke}}\langle (pk(k_a), pk(k_b))\rangle\Big)\\
\overset{\text{(vi)}}{\approx}\ & \nu k_a k_b.\Big(\mathsf{NSL}'_A\{sk(k_a)/x_{sk}, pk(k_b)/x_{pk_B}\} \mid \mathsf{NSL}'_B\{sk(k_b)/x_{sk}, pk(k_a)/x_{pk_A}\}\\
& \quad \mid \overline{net_{pke}}\langle (pk(k_a), pk(k_b))\rangle\Big)\\
\overset{\text{(vii)}}{\approx}\ & \nu net_{delA} net_{delB} k_a k_b.\Big(\mathsf{NSL}''_A \mid \mathsf{NSL}''_B \mid \overline{net_{pke}}\langle (pk(k_a), pk(k_b))\rangle\\
& \quad \mid net_{delA}(x).\overline{io_{ka}}\langle x\rangle \mid net_{delB}(x).\overline{io_{kb}}\langle x\rangle\Big)\\
\overset{\text{(viii)}}{\approx}\ & \nu net_{delA} net_{delB} k_a k_b.\Big(\mathsf{NSL}''_A \mid \mathsf{NSL}''_B \mid \overline{net_{pke}}\langle (pk(k_a), pk(k_b))\rangle\\
& \quad \mid \nu k.(net_{delA}(x).\overline{io_{ka}}\langle x\rangle \mid net_{delB}(x).\overline{io_{kb}}\langle x\rangle)\Big) =: \mathsf{NSL}_1
\end{aligned}$$

Here (v) uses two consecutive applications of Lemma 5, the first with $n := io_{pkeA}$ and $C := \square$ and $t := (sk(k_a), pk(k_a), pk(k_b))$, and the second with $n := io_{pkeB}$ and $C := \square$ and $t := (sk(k_b), pk(k_a), pk(k_b))$. Remember also that $io_{pkeA}((x_{sk}, \_, x_{pk_B}))$ is syntactic sugar for $io_{pkeA}(x)$.let $(x_{sk}, \_, x_{pk_B}) = x$.

And (vi) uses two consecutive applications of Lemma 4 (v) and the fact that $\approx$ is closed under evaluation contexts.

And (vii) uses two applications of Lemma 5 (both in the opposite direction), the first with $n := net_{delA}$, $Q := \overline{io_{ka}}\langle x\rangle$, and $t := H((n_a, x_{n_b}))$, and the second with $n := net_{delB}$, $Q := \overline{io_{kb}}\langle x\rangle$, and $t := H((x_{n_a}, n_b))$.

And (viii) uses Lemma 4 (i) to add νk.

Using Proverif, we can show the following observational equivalence:

$$\begin{aligned}\mathsf{NSL}_1 &\overset{(*)}{\approx} \nu net_{delA} net_{delB} k_a k_b.(\mathsf{NSL}''_A \mid \mathsf{NSL}''_B \mid \overline{net_{pke}}\langle(pk(k_a), pk(k_b))\rangle \mid \mathcal{F}_{KE})\\ &\equiv \nu net_{delA} net_{delB}.(\mathcal{F}_{KE}|S)\end{aligned}$$

for $S := \nu k_a k_b.(\mathsf{NSL}''_A|\mathsf{NSL}''_B|\overline{net_{pke}}\langle(pk(k_a), pk(k_b))\rangle)$. The Proverif code for checking $(*)$ is given in Figure 2.8.

Hence $\mathsf{NSL} \leq \mathcal{F}_{KE}$. □

2.7.2 Secure channel from key exchange.

Next, we realize a secure channel. Since we already have a realization of a secure key exchange at hand, we realize the secure channel SC from the idealized key exchange $\mathcal{F}_{KE}$. Later we replace $\mathcal{F}_{KE}$ by NSL. We model $\mathcal{F}_{SC}$ and SC based on $\mathcal{F}_{KE}$ as follows:

Definition 32 (Secure channel)**.** [12]

$$\mathcal{F}_{SC} := net_{scstart}().io_A(x).(\overline{net_{notify}}\langle\rangle \mid net_{deliver}().\overline{io_B}\langle x\rangle)$$

Definition 33 (Secure channel protocol)**.**

$$\begin{aligned}\mathsf{SC}_A &:= io_{ka}(x_k).io_A(x_m).\nu r.\overline{net_A}\langle senc(x_k, x_m, r)\rangle\\ \mathsf{SC}_B &:= io_{kb}(x_k).net_B(x_c).let\ x_m = sdec(x_k, x_c)\ in\ \overline{io_B}\langle x_m\rangle\\ \mathsf{SC} &:= \nu io_{ka} io_{kb}.(\mathsf{SC}_A|\mathsf{SC}_B|\mathcal{F}_{KE})\end{aligned}$$

Lemma 37. $\mathsf{SC} \leq \mathcal{F}_{SC}$.

Proof. We have:

$$\begin{aligned}\mathsf{SC} \equiv\ &\nu io_{ka} io_{kb} k.\Big(io_{ka}(x_k).io_A(x_m).\nu r.\overline{net_A}\langle senc(x_k, x_m, r)\rangle \mid io_{kb}(x_k).net_B(x_c).\\ &\quad \text{let } x_m = sdec(x_k, x_c) \text{ in } \overline{io_B}\langle x_m\rangle \mid net_{delA}().\overline{io_{ka}}\langle k\rangle \mid net_{delB}().\overline{io_{kb}}\langle k\rangle\Big)\\ \overset{(*)}{\approx}\ &\nu k.\Big(net_{delA}().io_A(x_m).\nu r.\overline{net_A}\langle senc(k, x_m, r)\rangle \mid net_{delB}().net_B(x_c).\\ &\quad \text{let } x_m = sdec(k, x_c) \text{ in } \overline{io_B}\langle x_m\rangle\Big) =: \mathsf{SC}_1\end{aligned}$$

Here $(*)$ uses two consecutive applications of Lemma 5, the first with $n := io_{ka}$ and $C := net_{delA}().\square$ and $t := k$, and the second with $n := io_{kb}$ and $C := net_{delB}().\square$

[12] This definition was already given in Section 2.5 (Definition 26) and is repeated here for convenience.

```
free B, netnsla, netnslb, netpke.
free ioka, iokb.

let A =
    new na;
    new r1;
    out(netnsla,penc(pk(kb),na,r1));
    in(netnsla,xc);
    let (=na,xnb,=B) = pdec(sk(ka),xc) in
    new r2;
    out(netnsla,penc(pk(kb),xnb,r2));
    out(netdela,hash((na,xnb))).

let B =
    in(netnslb,xc);
    let xna = pdec(sk(kb),xc) in
    new nb;
    new r;
    out(netnslb,penc(pk(ka),(xna,nb,B),r));
    in(netnslb,xc2);
    if nb = pdec(sk(kb),xc2) then
    out(netdelb,hash((xna,nb))).

let KE =
    new k;
    (in(netdela,x);out(ioka,choice[x,k])) |
    (in(netdelb,x);out(iokb,choice[x,k])).

process
    new netdela; new netdelb;
    new ka; new kb; (A | B | out(netpke,(pk(ka),pk(kb))) | KE)
```

Figure 2.8: Key-exchange example: Proverif code for analyzing NSL (`secchan-nsl.pv`, see [29]). (Has to be prefixed with the code from Figure 2.7.)

and $t := k$. (And it uses Lemma 1, so that we can apply Lemma 5 to a subprocess instead of the whole process.)

We show next:

$$\begin{aligned}\mathsf{SC}_1 \approx \nu s\,k.\Big(&\mathit{net}_{\mathit{delA}}().\mathit{io}_A(x_m).\nu r.(!\overline{(s, \mathit{senc}(k, x_m, r))}\langle x_m\rangle \mid \overline{\mathit{net}_A}\langle \mathit{senc}(k, x_m, r)\rangle) \mid\\ &\mathit{net}_{\mathit{delB}}().\mathit{net}_B(x_c).\text{let } x_m = \mathit{sdec}(k, x_c) \text{ in } (s, x_c)(x'_m).\overline{\mathit{io}_B}\langle x_m\rangle\Big) =: \mathsf{SC}_2\end{aligned}$$

By Lemma 9, to show the above it is sufficient to show that the trace property $\mathit{end}() \Rightarrow \mathit{start}()$ holds in the following event process:

$$\begin{aligned}\nu k.\Big(&\mathit{net}_{\mathit{delA}}().\mathit{io}_A(x_m).\nu r.\mathit{event}\ \mathit{start}(\mathit{senc}(k, x_m, r)).\overline{\mathit{net}_A}\langle \mathit{senc}(k, x_m, r)\rangle \mid\\ &\mathit{net}_{\mathit{delB}}().\mathit{net}_B(x_c).\text{let } x_m = \mathit{sdec}(k, x_c) \text{ in } \mathit{event}\ \mathit{end}(x_c).\overline{\mathit{io}_B}\langle x_m\rangle\Big).\end{aligned}$$

We show this trace property using Proverif, the required code is given in Figure 2.9.

Note: We could also have shown an analogous observational equivalence with s instead of $(s, \mathit{senc}(k, x_m, r))$. Then, however, Proverif fails on the code given in Figure 2.10 because it does not see there is only one message x_m sent over the channel. Thus, it believes that different x_m could be confused. Adding x_c to the channel name helps Proverif to see that x_m is unique (since x_c already determines x_m).

Since we send the message x_m directly to Bob via the channel $(s, \cdot)$ (who receives it as x'_m), we can let Bob output the message x'_m received over that channel instead of using the decrypted value x_m. Since then the plaintext of the ciphertext x_c is then not used any more, we can encrypt *empty* instead of x_m (as the adversary cannot tell the difference). Formally, we show the following observational equivalence:

$$\begin{aligned}\mathsf{SC}_2 \approx \nu s\,k.(&\mathit{net}_{\mathit{delA}}().\mathit{io}_A(x_m).\nu r.(!\overline{(s, \mathit{senc}(k, \mathit{empty}, r))}\langle x_m\rangle|\overline{\mathit{net}_A}\langle \mathit{senc}(k, \mathit{empty}, r)\rangle)\\ &\mid \mathit{net}_{\mathit{delB}}().\mathit{net}_B(x_c).\text{let } x_m = \mathit{sdec}(k, x_c) \text{ in } (s, x_c)(x'_m).\overline{\mathit{io}_B}\langle x'_m\rangle) =: \mathsf{SC}_3.\end{aligned}$$

We show this observational equivalence using Proverif, the required code is given in Figure 2.10.

Then we move the restriction νr to the top and replace the channel $(s, \mathit{senc}(k, \mathit{empty}, r))$ by s:

$$\begin{aligned}\mathsf{SC}_3 \overset{(*)}{\approx} \nu s\,k\,r.(&\mathit{net}_{\mathit{delA}}().\mathit{io}_A(x_m).(!\overline{(s, \mathit{senc}(k, \mathit{empty}, r))}\langle x_m\rangle \mid \overline{\mathit{net}_A}\langle \mathit{senc}(k, \mathit{empty}, r)\rangle)\\ &\mid \mathit{net}_{\mathit{delB}}().\mathit{net}_B(x_c).\text{let } x_m = \mathit{sdec}(k, x_c) \text{ in } (s, x_c)(x'_m).\overline{\mathit{io}_B}\langle x'_m\rangle)\\ \overset{(**)}{\approx} \nu s\,k\,r.(&\mathit{net}_{\mathit{delA}}().\mathit{io}_A(x_m).(!\overline{s}\langle x_m\rangle \mid \overline{\mathit{net}_A}\langle \mathit{senc}(k, \mathit{empty}, r)\rangle) \mid\\ &\mathit{net}_{\mathit{delB}}().\mathit{net}_B(x_c).\text{let } x_m = \mathit{sdec}(k, x_c) \text{ in } s(x'_m).\overline{\mathit{io}_B}\langle x'_m\rangle) =: \mathsf{SC}_4\end{aligned}$$

Here $(*)$ follows from Lemma 4 (ii), and $(**)$ is proven using Proverif. The required code is given in Figure 2.11.

```
free ioa. (* A-input of F_SC *)
free iob. (* B-output of F_SC *)
free neta. (* A-end of insecure channel in P_SC *)
free netb. (* B-end of insecure channel in P_SC *)
free netdela, netdelb.

query ev:end(x) ==> ev:start(x).

let PA =
    in(netdela,x);
    in(ioa,xm);
    new r;
    event start(senc(k,xm,r));
    out(neta,senc(k,xm,r)).

let PB =
    in(netdelb,x);
    in(netb,xc);
    let xm=sdec(k,xc) in
    event end(xc);
    out(iob,xm).

process
    new k;
    PA | PB
```

Figure 2.9: Key-exchange example: Proverif code for analyzing the trace property of SC (`secchan-sc1.pv`, see [29]). (Has to be prefixed with the code from Figure 2.7.)

```
free ioa. (* A-input of F_SC *)
free iob. (* B-output of F_SC *)
free neta. (* A-end of insecure channel in P_SC *)
free netb. (* B-end of insecure channel in P_SC *)
free netdela, netdelb.

let PA =
    in(netdela,x);
    in(ioa,xm);
    new r;
    (!out((s,senc(k,choice[xm,empty],r)),xm)) |
    out(neta,senc(k,choice[xm,empty],r)).

let PB =
    in(netdelb,x);
    in(netb,xc);
    let xm=sdec(k,xc) in
    in((s,xc),xm2);
    out(iob,choice[xm,xm2]).

process
    new s;
    new k;
    PA | PB
```

Figure 2.10: Key-exchange example: Proverif code for analyzing the observation equivalence in SC (`secchan-sc2.pv`, see [29]). (Has to be prefixed with the code from Figure 2.7.)

```
free ioa. (* A-input of F_SC *)
free iob. (* B-output of F_SC *)
free neta. (* A-end of insecure channel in P_SC *)
free netb. (* B-end of insecure channel in P_SC *)
free netdela, netdelb.

let PA =
    in(netdela,x);
    in(ioa,xm);
    (!out(choice[(s,senc(k,empty,r)),s],xm)) |
    out(neta,senc(k,empty,r)).

let PB =
    in(netdelb,x);
    in(netb,xc);
    let xm=sdec(k,xc) in
    in(choice[(s,xc),s],xm2);
    out(iob,xm2).

process
    new s;
    new k;
    new r;
    PA | PB
```

Figure 2.11: Key-exchange example: Proverif code for analyzing the second observation equivalence in SC (`secchan-sc3.pv`, see [29]). (Has to be prefixed with the code from Figure 2.7.)

We continue:

$$\begin{aligned}
\mathsf{SC}_4 &\overset{(*)}{\approx} \nu \mathit{net}_{\mathit{deliver}}\, k\, r.(\mathit{net}_{\mathit{delA}}().\mathit{io}_A(x_m).(\mathit{net}_{\mathit{deliver}}().\overline{\mathit{io}_B}\langle x_m\rangle \mid \overline{\mathit{net}_A}\langle \mathit{senc}(k, \mathit{empty}, r)\rangle)\\
&\qquad \mid \mathit{net}_{\mathit{delB}}().\mathit{net}_B(x_c).\text{let } x_m = \mathit{sdec}(k, x_c) \text{ in } \overline{\mathit{net}_{\mathit{deliver}}}\langle\rangle)\\
&\overset{(**)}{\approx} \nu \mathit{net}_{\mathit{deliver}}\, k\, r\; \mathit{net}_{\mathit{notify}}.(\mathit{net}_{\mathit{delA}}().\mathit{io}_A(x_m).(\mathit{net}_{\mathit{deliver}}().\overline{\mathit{io}_B}\langle x_m\rangle \mid \overline{\mathit{net}_{\mathit{notify}}}\langle\rangle) \mid\\
&\qquad \mathit{net}_{\mathit{delB}}().\mathit{net}_B(x_c).\text{let } x_m = \mathit{sdec}(k, x_c) \text{ in } \overline{\mathit{net}_{\mathit{deliver}}}\langle\rangle \mid\\
&\qquad \mathit{net}_{\mathit{notify}}().\overline{\mathit{net}_A}\langle \mathit{senc}(k, \mathit{empty}, r)\rangle)\\
&\equiv \nu \mathit{net}_{\mathit{deliver}}\; \mathit{net}_{\mathit{notify}}.(\mathcal{F}_{SC}\{\mathit{net}_{\mathit{delA}}/\mathit{net}_{\mathit{scstart}}\}|S)\\
&\qquad \text{with } S := \nu kr.\mathit{net}_{\mathit{delB}}().\mathit{net}_B(x_c).\text{let } x_m = \mathit{sdec}(k, x_c) \text{ in } \overline{\mathit{net}_{\mathit{deliver}}}\langle\rangle\\
&\qquad\qquad\qquad \mid \mathit{net}_{\mathit{notify}}().\overline{\mathit{net}_A}\langle \mathit{senc}(k, \mathit{empty}, r)\rangle
\end{aligned}$$

Here $(*)$ uses Lemma 6 with $Q := \overline{\mathit{io}_B}\langle x'_m\rangle$, $x := x'_m$, $n := s$, and $m := \mathit{net}_{\mathit{deliver}}$.

And $(**)$ uses Lemma 5 with $Q := \overline{\mathit{net}_A}\langle \mathit{senc}(k, \mathit{empty}, r)\rangle$, $n := \mathit{net}_{\mathit{notify}}$, $t := \mathit{empty}$.

So $\mathsf{SC} \approx \nu \underline{n}.(\mathcal{F}_{SC}\sigma|S)$ for $\sigma := \{\mathit{net}_{\mathit{delA}}/\mathit{net}_{\mathit{scstart}}\}$ and $\underline{n} := \mathit{net}_{\mathit{deliver}}\, \mathit{net}_{\mathit{notify}}$. Hence $\mathsf{SC} \leq \mathcal{F}_{SC}$. □

With $\mathsf{NSL} \leq \mathcal{F}_{KE}$ (Lemma 36) and $\mathsf{SC} \leq \mathcal{F}_{SC}$ (Lemma 37) at hand we can now use the compositional capabilities of UC: We define an evaluation context $\mathcal{C}[\square] := \nu \mathit{io}_{ka}\mathit{io}_{kb}.(\mathsf{SC}_A|\mathsf{SC}_B|\square)$ where SC_A and SC_B are the processes from Definition 33. Since $\mathcal{C}$ meets the requirements of Theorem 1 $\mathsf{NSL} \leq \mathcal{F}_{KE}$ implies $\mathcal{C}[\mathsf{NSL}] \leq \mathcal{C}[\mathcal{F}_{KE}]$. Since $\mathcal{C}[\mathcal{F}_{KE}] = \mathsf{SC}$ and $\mathsf{SC} \leq \mathcal{F}_{SC}$ we have, by transitivity of $\leq$ (Lemma 12), $\mathcal{C}[\mathsf{NSL}] \leq \mathcal{F}_{SC}$.

We did construct a secure channel from a PKI using the NSL protocol. More interesting than this result is the way we achieved it: We did not have to analyze the complete system $\mathcal{C}[\mathsf{NSL}]$ in one piece but could replace the NSL protocol with an idealized functionality. This illustrates two striking advantages of the UC approach:

- The fact that NSL realizes an ideal key exchange can be re-used for security proofs of further systems.
- We cannot only plug NSL into $\mathcal{C}$ but any protocol that realizes a secure key exchange (e.g., if no PKI is available and thus NSL is not an option).

Instead of one monolithic security proof for $\mathcal{C}[\mathsf{NSL}]$ we end up with smaller proofs and results which can be used flexibly. Furthermore, to split the security analysis of a complex system into smaller parts might be the only feasible option to tackle it at all.

2.7.3 Generating many keys from one

While the example until now illustrates composition and the power of UC, $\mathcal{C}[\mathsf{NSL}]$ only realizes a single-use secure channel. To transfer multiple messages, we could just use concurrent composition to have $!!\mathcal{C}[\mathsf{NSL}] \leq !!\mathcal{F}_{SC}$. However, the resulting protocol uses one instance of NSL per message, and – since NSL contains $\mathcal{F}_{PKI}$, *another* PKI for each message that is sent. This is clearly unrealistic. To get rid of this overhead we want to have all the instances of SC to jointly use just one key exchange $\mathcal{F}_{KE}$, i.e., we want to use the previously mentions joined state technique here. Towards this goal we model a wrapper protocol KE^* which uses one key exchange to emulate multiple key exchanges (from a key k it derives session keys $\mathit{hash}((\mathit{sid}, k))$ where sid is the session id). Formally, we define KE^* as follows and then show $\mathsf{KE}^* \leq !!\mathcal{F}_{KE}$.

Definition 34.

$$\begin{aligned}
\mathsf{KE}^*_A &:= io'_{ka}(x_k).!!_{x_{sid}}\overline{io_{ka}}\langle hash((x_{sid},x_k))\rangle\\
\mathsf{KE}^*_B &:= io'_{kb}(x_k).!!_{x_{sid}}\overline{io_{kb}}\langle hash((x_{sid},x_k))\rangle\\
\mathsf{KE}^* &:= \nu io'_{ka}io'_{kb}.(\mathsf{KE}^*_A \mid \mathsf{KE}^*_B \mid \mathcal{F}'_{KE})
\end{aligned}$$

where $\mathcal{F}'_{KE} := \mathcal{F}_{KE}\{io'_{ka}/io_{ka}, io'_{kb}/io_{kb}\}$.

Lemma 38. $\mathsf{KE}^* \leq !!\mathcal{F}_{KE}$.

Proof. Let $S := net_{delA}().!!\overline{net'_{delA}}\langle\rangle \mid net_{delB}().!!\overline{net'_{delB}}\langle\rangle$. Here we use the shorthand $\overline{t}\langle\rangle$ for $\overline{t}\langle empty\rangle$. Let $\underline{n} := net'_{delA}net'_{delB}$. Let $\sigma := \{net'_{delA}/net_{delA}, net'_{delB}/net_{delB}\}$.

We have

$$\begin{aligned}
\mathsf{KE}^* &\overset{(i)}{\approx} \nu k.net_{delA}().!!_{x_{sid}}\overline{io_{ka}}\langle hash((x_{sid},k))\rangle \mid net_{delB}().!!_{x_{sid}}\overline{io_{kb}}\langle hash((x_{sid},k))\rangle\\
&\overset{(ii)}{\approx} \nu k.net_{delA}().!!_{x_{sid}}\nu net'_{delA}.(\overline{net'_{delA}}\langle\rangle \mid net'_{delA}().\overline{io_{ka}}\langle hash((x_{sid},k))\rangle)\\
&\qquad \mid net_{delB}().!!_{x_{sid}}\nu net'_{delB}.(\overline{net'_{delB}}\langle\rangle \mid net'_{delB}().\overline{io_{kb}}\langle hash((x_{sid},k))\rangle)\\
&\overset{(iii)}{\approx} \nu k.\nu net'_{delA}.net_{delA}().(!!_{x_{sid}}\overline{net'_{delA}}\langle\rangle \mid !!_{x_{sid}}net'_{delA}().\overline{io_{ka}}\langle hash((x_{sid},k))\rangle)\\
&\qquad \mid \nu net'_{delB}.net_{delB}().(!!_{x_{sid}}\overline{net'_{delB}}\langle\rangle \mid !!_{x_{sid}}net'_{delB}().\overline{io_{kb}}\langle hash((x_{sid},k))\rangle)\\
&\overset{(iv)}{\approx} \nu k.\nu net'_{delA}.(net_{delA}().!!_{x_{sid}}\overline{net'_{delA}}\langle\rangle \mid !!_{x_{sid}}net'_{delA}().\overline{io_{ka}}\langle hash((x_{sid},k))\rangle)\\
&\qquad \mid \nu net'_{delB}.(net_{delB}().!!_{x_{sid}}\overline{net'_{delB}}\langle\rangle \mid !!_{x_{sid}}net'_{delB}().\overline{io_{kb}}\langle hash((x_{sid},k))\rangle)\\
&\overset{(v)}{\approx} \nu\underline{n}.\Big(\nu k.!!_{x_{sid}}\Big(net'_{delA}().\overline{io_{ka}}\langle hash((x_{sid},k))\rangle \mid net'_{delB}().\overline{io_{kb}}\langle hash((x_{sid},k))\rangle\Big) \mid S\Big)\\
&\overset{(vi)}{\approx} \nu\underline{n}.(!!_{x_{sid}}\nu k.(net'_{delA}().\overline{io_{ka}}\langle k\rangle \mid net'_{delB}().\overline{io_{kb}}\langle k\rangle) \mid S)\\
&= \nu\underline{n}.(!!\mathcal{F}_{KE}\,\sigma \mid S)
\end{aligned}$$

Here (i) uses two application of Lemma 5, the first with $C := net_{delA}().\square$, $n := io'_{ka}$, and $t := k$, the second with $C := net_{delB}().\square$, $n := io'_{kb}$, and $t := k$. (And it uses Lemma 1, so that we can apply Lemma 5 to a subprocess instead of the whole process.)

And (ii) uses Lemma 5 with $C := \square$ to show $\overline{io_{ka}}\langle hash((x_{sid},k))\rangle \approx \nu net'_{delA}.(\overline{net'_{delA}}\langle\rangle \mid net'_{delA}().\overline{io_{ka}}\langle hash((x_{sid},k))\rangle$ and $\overline{io_{kb}}\langle hash((x_{sid},k))\rangle \approx \nu net'_{delB}.(\overline{net'_{delB}}\langle\rangle \mid net'_{delB}().\overline{io_{kb}}\langle hash((x_{sid},k))\rangle$.

And (iii) uses Lemma 4 (ii) and Lemma 33 and Lemma 32.

And (iv) uses the following claim (proven below) twice. First with $n := net'_{delA}$, $m := net_{delA}$, $Q := \overline{io_{ka}}\langle hash((x_{sid},k))\rangle$. Then with $n := net'_{delB}$, $m := net_{delB}$, $Q := \overline{io_{kb}}\langle hash((x_{sid},k))\rangle$.

Claim 4. *For names n, m, and for any process Q, we have $\nu n.m().(!!_x\overline{n}\langle\rangle \mid !!_x n().Q) \approx \nu n.((m().!!_x\overline{n}\langle\rangle) \mid !!_x n().Q)$.*

(Intuitively, this claim holds because $!!_x n().Q$ cannot perform any observable actions until $!!_x\overline{n}\langle\rangle$ is executed. So it makes no difference whether both $!!_x n().Q$ and $!!_x\overline{n}\langle\rangle$ wait for the input on m to occur, or whether only $!!_x n().Q$ waits for it.)

And (v) follows from the definition of $\equiv$ and Lemma 33.

Finally, (vi) follows from the following claim (proven below):

Claim 5. *For any process P, we have $\nu k.!!_x P\{hash((x,k))/k\} \approx !!_x\nu k.P$.*

Thus we have derived $\mathsf{KE}^* \approx \nu\underline{n}.(!!\mathcal{F}_{KE}\,\sigma \mid S)$. This shows $\mathsf{KE}^* \leq !!\mathcal{F}_{KE}$. It remains to show the two claims.

To show Claim 4, consider the following relation:

$$\mathcal{R} := \Big\{E[\nu n.m().(\prod_{x\in SID}\overline{n}\langle\rangle \mid \prod_{x\in SID} n().Q(\!(x)\!))],$$
$$E[\nu n.(m().\prod_{x\in SID}\overline{n}\langle\rangle \mid \prod_{x\in SID\setminus S} n().Q(\!(x)\!) \mid \sum_{x\in S} n().Q(\!(x)\!))]\Big\} \cup \approx$$

up to structural equivalence. Here E ranges over evaluation contexts, and S over finite subsets of *SID*. n, m, Q are from the statement of the lemma. $\sum_{x\in S} P$ stands short for $P\{s_1/x\}|\dots|P\{s_k/x\}$ with $S =: \{s_1,\dots,s_k\}$. I.e., $\sum_{x\in S}$ is almost the same as $\prod_{x\in S}$, except that $\sum_{x\in S}$ is syntactic sugar (and only makes sense for finite S) while $\prod_{x\in S}$ is a proper construct in the syntax of product processes.

We show that $\mathcal{R}$ is a bisimulation:

- If $(A,B)\in\mathcal{R}$ and $A\downarrow_M$, then $B\downarrow_M$:

  In the case $A \approx B$, the statement is immediate. We can thus assume $A \equiv E[\nu n.m().(\prod_{x\in SID}\overline{n}\langle\rangle \mid \prod_{x\in SID} n().Q(\!(x)\!))]$ and $B \equiv E[\nu n.(m().\prod_{x\in SID}\overline{n}\langle\rangle \mid \prod_{x\in SID\setminus S} n().Q(\!(x)\!) \mid \sum_{x\in S} n().Q(\!(x)\!))]$.

  In the argument to E, there are no unprotected outputs. Thus the output on M is in E and thus $B\downarrow_M$ trivially follows.

- If $(A,B)\in\mathcal{R}$ and $B\downarrow_M$, then $A\downarrow_M$: Analogous to the previous case.

- If $(A,B)\in\mathcal{R}$ and $A\to A'$, then there is a B' with $B\to^* B'$ and $(A',B')\in\mathcal{R}$:

  In the case $A \approx B$, the statement is immediate. We can thus assume $A \equiv E[\nu n.m().(\prod_{x\in SID}\overline{n}\langle\rangle \mid \prod_{x\in SID} n().Q(\!(x)\!))]$ and $B \equiv E[\nu n.(m().\prod_{x\in SID}\overline{n}\langle\rangle \mid \prod_{x\in SID\setminus S} n().Q(\!(x)\!) \mid \sum_{x\in S} n().Q(\!(x)\!))]$.

  If $A\to A'$ is a reduction within E, then let $B\to B'$ be the corresponding reduction, and then $(A',B')\in\mathcal{R}$.

  Otherwise, $A\to A'$ is a communication on m between E and the input $m()$ in its argument, hence $A' \equiv E'[\nu n.(\prod_{x\in SID}\overline{n}\langle\rangle \mid \prod_{x\in SID} n().Q(\!(x)\!))]$. And $B\to B' := E'[\nu n.(\prod_{x\in SID}\overline{n}\langle\rangle \mid \prod_{x\in SID\setminus S} n().Q(\!(x)\!) \mid \sum_{x\in S} n().Q(\!(x)\!))]$.

  From Lemma 4 (ix), we have $A' \approx B'$, hence $(A',B')\in\mathcal{R}$.

- If $(A,B)\in\mathcal{R}$ and $B\to B'$, then there is a A' with $A\to^* A'$ and $(A',B')\in\mathcal{R}$:

  In the case $A \approx B$, the statement is immediate. We can thus assume $A \equiv E[\nu n.m().(\prod_{x\in SID}\overline{n}\langle\rangle \mid \prod_{x\in SID} n().Q(\!(x)\!))]$ and $B \equiv E[\nu n.(m().\prod_{x\in SID}\overline{n}\langle\rangle \mid \prod_{x\in SID\setminus S} n().Q(\!(x)\!) \mid \sum_{x\in S} n().Q(\!(x)\!))]$.

  If $B\to B'$ is a reduction within E, or if $B\to B'$ is a communication on m between E and $m()$ in its argument, then the reasoning is as in the previous case.

  Otherwise, we have that $B\to B'$ is a reduction of the second product, i.e. $B' \equiv E[\nu n.(m().\prod_{x\in SID}\overline{n}\langle\rangle \mid \prod_{x\in SID\setminus S'} n().Q(\!(x)\!) \mid \sum_{x\in S'} n().Q(\!(x)\!))]$ with $S' := S\setminus\{t\}$ for some $t\in SID\setminus S$. Then $(A',B')\in\mathcal{R}$ with $A' := A$.

- If $(A, B) \in \mathcal{R}$, then $(E[A], E[B]) \in \mathcal{R}$:

  Immediate from the definition of $\mathcal{R}$.

The statement of the claim is equivalent to

$$\begin{aligned} P_1 &:= \nu n.m().\Big(\prod_{x \in \mathit{SID}} \overline{n}\langle\rangle \mid \prod_{x \in \mathit{SID}} n().Q(\!(x)\!)\Big) \\ &\approx \nu n.\Big((m().\prod_{x \in \mathit{SID}} \overline{n}\langle\rangle) \mid \prod_{x \in \mathit{SID}} n().Q(\!(x)\!)\Big) =: P_2. \end{aligned}$$

And this follows from the fact that $\mathcal{R}$ is a bisimulation since $(P_1, P_2) \in \mathcal{R}$. Thus Claim 4 is shown.

To show Claim 5, consider the following relation:

$$\mathcal{R} := \Big\{\Big(\nu\underline{n}k.Q\sigma \mid \prod_{x \in S} P\{\mathit{hash}((x,k))/k\}, \quad \nu\underline{n}\,\underline{k}_\sigma.Q \mid \prod_{x \in S} \nu k.P\Big)\Big\}$$

up to structural equivalence. Here $k \notin \mathit{fn}(S)$ is an arbitrary name, $S \subseteq \mathit{SID}$ is a set of terms, σ is a (finite) substitution mapping names to distinct (with respect to $=_\mathrm{E}$) terms $\mathit{hash}((t,k))$ with $t \in \mathit{SID} \setminus S$, $\underline{k}_\sigma = \operatorname{dom}\sigma$, $\underline{k}_\sigma \cap \mathit{fn}(P,S) = \varnothing$, $\underline{n}$ is a list of names, and Q is an arbitrary process with $k \notin \mathit{fn}(Q)$.

We show that $\mathcal{R}$ is a bisimulation:

- If $(A, B) \in \mathcal{R}$ and $A \downarrow_M$ then $B \downarrow_M$:

  Since k and $\underline{k}_\sigma$ are bound names, we have that M does not contain either of them. But only terms containing k or k_S are different in A and B. Thus $B \downarrow_M$.

- If $(A, B) \in \mathcal{R}$ and $B \downarrow_M$ then $A \downarrow_M$:

  Analogous.

- If $(A, B) \in \mathcal{R}$ and $A \rightarrow A'$, then there is a B' with $B \rightarrow^* B'$ and $(A', B') \in \mathcal{R}$:

  If the reduction is $\prod_{x \in S} P\{\mathit{hash}((x,k))/k\} \rightarrow P\{\mathit{hash}((t,k))/k, t/x\} \mid \prod_{x \in S'} P\{\mathit{hash}((x,k))/k\}$ with $S' := S \setminus \{t\}$, then we have $B \rightarrow^* B'$ and $(A', B') \in \mathcal{R}$ with $B' := \nu\underline{nk}_{\sigma'}.Q \mid P\{k_t/k, t/x\} \mid \prod_{x \in S'} \nu k.P$ and $\sigma' := \sigma \cup \{k_t \mapsto H((t,k))\}$ for some fresh name k_t. Notice that the terms in the range of σ' are still distinct because $S \subseteq \mathit{SID}$ contains only distinct terms, and $t \in \mathit{SID} \setminus S$.

  If the reduction is a reduction of $Q\sigma \rightarrow Q'$, then it is easy to see (by checking, in particular, for all destructors that $f(t_1, \dots, t_n)\sigma = f(t_1\sigma, \dots, t_n\sigma)$) that $Q \rightarrow Q'\sigma^{-1}$. From this it follows that $B \rightarrow^* B'$ and $(A, B) \in \mathcal{R}$ with $B' := \nu\underline{n}\,\underline{k}_\sigma.Q'\sigma^{-1} \mid \prod_{x \in S} \nu k.P$.

- If $(A, B) \in \mathcal{R}$ and $B \rightarrow B'$, then there is a A' with $A \rightarrow^* A'$ and $(A', B') \in \mathcal{R}$:

  If the reduction is $\prod_{x \in S} \nu k.P \rightarrow \nu k.P\{t/x\} \mid \prod_{x \in S'} \nu P$ with $S' := S \setminus \{t\}$, then we have $(A', B') \in \mathcal{R}$ with $A' := \nu\underline{n}k.(Q \mid P\{H((t,k))/k\})\sigma \mid \prod_{x \in S'} P\{H((x,k))/k\}$ and $B' \equiv \nu\underline{nk}_{\sigma'}.Q \mid P\{k_t/k\} \mid \prod_{x \in S'} \nu k.P$ and $\sigma' := \sigma \cup \{k_t \mapsto H((t,k))\}$ and some fresh name k_t. Notice that the terms in the range of σ' are still distinct because $S \subseteq \mathit{SID}$ contains only distinct terms, and $t \in \mathit{SID} \setminus S$.

If the reduction is a reduction of $Q \to Q'$, then it is easy to see (by checking, in particular, for all destructors that $f(t_1, \dots, t_n)\sigma = f(t_1\sigma, \dots, t_n\sigma)$) that $Q\sigma \to Q'\sigma$. From this it follows that $(A, B) \in \mathcal{R}$ with $A' := \nu\underline{n}k.Q'\sigma \mid \prod_{x\in S} P\{\mathit{hash}((x,k))/k\}$.

- If $(A, B) \in \mathcal{R}$ and E is an evaluation context, then $(E[A], E[B]) \in \mathcal{R}$:

  Then $A = \nu\underline{n}k.Q\sigma \mid \prod_{x\in S} P\{\mathit{hash}((x,k))/k\}$ and $B = \nu\underline{n}\,\underline{k}_\sigma.Q \mid \prod_{x\in S} \nu k.P$. Without loss of generality, $k, \underline{k}_\sigma \notin \mathit{fn}(E) \cup \mathit{fn}(E)$. (Otherwise we could replace $k, \underline{k}_\sigma$ by other names in A, B.) There is a process Q' and a list of names $\underline{n}'$ such that $E[P] \equiv \nu\underline{n}'.(P|Q')$ for all P. Then $(E[A], E[B]) \equiv$

$$\Big(\nu\underline{n}'\,\underline{n}k.(Q|Q')\sigma \mid \prod_{x\in S} P\{\mathit{hash}((x,k))/k\}, \quad \nu\underline{n}'\,\underline{n}\,\underline{k}_\sigma.(Q|Q') \mid \prod_{x\in S} \nu k.P\Big) \in \mathcal{R}.$$

Since $(\nu k. \prod_{x\in \mathit{SID}} P\{\mathit{hash}((x,k))/k\}, \prod_{x\in \mathit{SID}} \nu k.P) \in \mathcal{R}$, we have $\nu k.!!_x P\{\mathit{hash}((x,k))/k\} \approx \nu k. \prod_{x\in \mathit{SID}} P(\!(x)\!)\{\mathit{hash}((x,k))/k\} \approx \prod_{x\in \mathit{SID}} \nu k.P(\!(x)\!) \approx\ !!\nu k.P$. This shows Claim 5. □

Analogously to the single session case we define a suitable context $\mathcal{C}^*$ by replacing $\mathcal{F}'_{KE}$ in KE^* with □ and have

$$\mathcal{C}^*[\mathsf{NSL}] \leq \mathcal{C}^*[\mathcal{F}'_{KE}] = \mathsf{KE}^* \leq\ !!\mathcal{F}_{KE}$$

Furthermore, $!!\mathsf{SC} \approx \nu io_{ka} io_{kb}.(!!\mathsf{SC}_A|!!\mathsf{SC}_B|!!\mathcal{F}_{KE})$ (by Lemmas 32,33). Hence

$$\begin{aligned} &\nu io_{ka} io_{kb}.(!!\mathsf{SC}_A|!!\mathsf{SC}_B|\mathcal{C}^*[\mathsf{NSL}]) \\ &\leq \nu io_{ka} io_{kb}.(!!\mathsf{SC}_A|!!\mathsf{SC}_B|!!\mathcal{F}_{KE}) \\ &\leq\ !!\mathsf{SC} \leq\ !!\mathcal{F}_{SC}. \end{aligned}$$

Finally, we have a protocol which realizes multiple secure channels while invoking the NSL protocol and using only one PKI.

2.8 Virtual primitives

In this section, we present a technique for deriving security of protocols in the symbolic UC model that is specific to the symbolic model. No analogue in the computational world seems to exist. The idea is the following: When constructing UC secure protocols, it is often necessary to include specific "trapdoors" that allow the simulator to extract or modify certain information. For example, when constructing a simulator for a commitment scheme, we need to include in the protocol some way for the simulator to extract the value of the commitment when given a commitment by the environment (*extractability*), or to change the content of a commitment when producing a commitment for the environment (*equivocality*), see [37]. These additional trapdoors often make the protocols more complex, and they often also need more complex cryptographic primitives. A simple commitment protocol in which the committer just sends $\mathit{hash}(m,r)$ for message m and randomness r is not UC secure because the simulator cannot extract or equivocate. Instead, one would need to assume a special hash function that takes an additional parameter crs (the common reference string) $\mathit{hash}(\mathit{crs},m,r)$ in such a way that given a suitably chosen "fake" crs, one can find collisions in hash or extract m from $\mathit{hash}(\mathit{crs},m,r)$. With such a hash function, one can construct a UC secure commitment relatively easily (see

Definition 37 below). However, now our protocol uses a considerably more complex primitive than a simple hash function. And certainly common hash functions such as SHA-3 do not have these properties.

This leads to a strange situation: We have a protocol that we can only prove secure using a hash function that has additional weaknesses (namely that given a "bad" crs, one can cheat). One might be tempted to state that if the protocol is secure for such weak hash functions, it should in particular be secure for good hash functions. Unfortunately, such reasoning does not work in the computational setting: We cannot just remove the existence of trapdoors from the hash function – if we do so, we have a completely different hash function and our security proof makes no claims about that function.

In the symbolic world, things are different. Here it turns out that we can indeed first analyze a protocol using a hash function with trapdoors, and then remove these trapdoors in a later step, still preserving security. We call this approach the "virtual primitives" approach, because we use primitives (in this example a hash function with trapdoors) that do not need to actually exist, and that are removed in the final protocol.

In a nutshell, the virtual primitives approach when trying to realize a functionality $\mathcal{F}$ (e.g., a commitment) works as follows:

- First, identify a symbolic model $\mathcal{M}_{real}$ containing cryptographic primitives (e.g. a hash function) that should be used in the final protocol.
- Extend $\mathcal{M}_{real}$ by additional constructors, destructors, or equality rules, call the resulting model $\mathcal{M}_{virt}$. The extension $\mathcal{M}_{virt}$ should be "safe" in the sense that in $\mathcal{M}_{virt}$ an adversary will have at least as much power as in $\mathcal{M}_{real}$ (this will be made formal in Section 2.8.2).
- Design a protocol P. Show that P emulates $\mathcal{F}$ with respect to $\mathcal{M}_{virt}$.
- Compose P with other protocols, leading to a complex protocol $C[P] \leq C[\mathcal{F}] \leq \mathcal{G}$ (with respect to $\mathcal{M}_{virt}$) where $\mathcal{G}$ is some desired final goal, e.g., some crypto-heavy voting protocol.
- Property preservation guarantees that any property $\wp$ that holds for $\mathcal{G}$ also holds for $C[P]$ (with respect to $\mathcal{M}_{virt}$). Since $\mathcal{M}_{virt}$ only makes adversaries stronger, $\wp$ also holds for $C[P]$ with respect to $\mathcal{M}_{real}$.
- Summarizing, we have constructed a protocol $C[P]$ in a modular way such that $C[P]$ uses the symbolic model $\mathcal{M}_{real}$ (without any trapdoors) and has all the security properties of the functionality $\mathcal{G}$.

The virtual primitive approach is not limited to commitments. But in the following sections, we illustrate it in the case of a commitment protocol. Note however, that the main theorem that allows us to conclude that $\mathcal{M}_{virt}$-security implies $\mathcal{M}_{real}$-security is formulated for general safe extensions.

A few words are in order why the virtual primitives approach works in the symbolic setting. What is the specific property of the symbolic model – in contrast to the computational one – that makes it possible? In our interpretation, this is due to the fact that a primitive (like hashes) in the symbolic world is a concrete object (i.e., a particular constructor with certain reduction rules and equalities) while in the computational world it is a class of objects (hash functions) that are described by some negative properties ("functions such that the adversary cannot. . . "). Therefore in the symbolic world, it is possible to formally compare executions using different kinds of a primitive (e.g., hashes with and without trapdoors); executions in one setting can be mapped into executions in the other setting by rewriting the terms sent around. In contrast, in the computational setting, this is not possible: a security result for hash functions with trapdoors has no implications for hash functions without trapdoors – these two are completely different mathematical functions on bitstrings, and it is not possible to rewrite an execution that uses one hash function into an execution using another (in particular if the adversary makes his actions depend on individual bits of the hashes). This difference between the symbolic and the computational setting seems to be the reason why virtual primitives work in the symbolic setting.

Related approaches in the computational model.

Although virtual primitives as described above are restricted to the symbolic setting, somewhat related techniques do exist in the computational model. [69, 12] show how to circumvent UC impossibility results (such as the impossibility of OT, commitment, or general multi-party computation without trusted setup) by giving the simulator additional power. Namely the simulator is allowed to run in (slightly) superpolynomial time. This is in some sense similar to giving the simulator access to additional constructors/destructors for extraction/equivocation as we do. Yet, there are three crucial differences to our setting: First, they can only use primitives that can actually exist computationally. For example, even a superpolynomial-time simulator cannot invert a fixed-length hash function, as part of the input is information-theoretically lost. In contrast, we can add arbitrary properties to, e.g., hash functions by introducing new equations in the symbolic model. Second, their final protocols have to use whatever primitives have been introduced for proof purposes; it is not possible to remove additional properties in the end as done in our approach. Third, their protocols involve advanced cryptographic techniques which makes the protocols considerably more involved and, consequently, inefficient. On the other hand, of course, protocols designed with our techniques are only proven secure in the symbolic model but lack a proof in the computational model – we believe therefore that our and their approaches are incomparable with respect to their advantages and disadvantages.

2.8.1 Realizing commitments

For simplicity, we formulate a commitment functionality where the adversary is not informed that a commitment takes place (when both Alice and Bob are honest). Of course, such a functionality can only be realized if we assume perfectly secure channels between Alice and Bob that do not even allow the adversary to notice or block messages. If our protocols were to use secure channels where the adversary can notice and block communication, we would instead realize a somewhat weaker functionality which notifies the adversary[13] (the resulting changes in the proof are orthogonal to the issues of this chapter).

Definition 35 (Commitment). $\mathcal{F}_{COM} := io_{coma}(x_m).(\overline{io_{comb}}\langle\rangle | io_{opena}().\overline{io_{openb}}\langle x_m\rangle)$.

Symbolic model.

The symbolic model $\mathcal{M}_{real}$ has constructors $hash/2$, $empty/0$, and $(\cdot,\cdot)$ (pairs) – f/n means f has arity n –, has destructors *fst*, *snd*, has no equalities, and has the rewrite rules for *fst*, *snd*, *equals* prescribed by Definition 5. This model $\mathcal{M}_{real}$ is quite

[13] Namely, $\mathcal{F}_{COM} := io_{coma}(x_m).(\overline{net_{coma}}\langle\rangle | net_{comb}().\overline{io_{comb}}\langle\rangle | io_{opena}().(\overline{net_{opena}}\langle\rangle | net_{openb}().\overline{io_{openb}}\langle x_m\rangle))$

```
fun hash/2.
fun empty/0.
fun fake/3.
fun fakeH/2.
fun crseqv/1.
fun crsext/1.
equation hash(crseqv(n),(m,fake(n,m,r))) = fakeH(n,r).
reduc extract(n,hash(crsext(n),(m,r))) = m.
```

Figure 2.12: Virtual primitives example: Proverif code for the symbolic model (`virtprim-model.pv`, see [29])

standard and does not use any cryptography except hash functions (*hash* is binary for convenience only).

As explained above, to construct UC-secure commitments, we need additional "trapdoors" in our equational theory. Let $\mathcal{M}_{\mathit{virt}}$ be the symbolic model $\mathcal{M}_{\mathit{real}}$ with the following additions: Constructors $\mathit{fake}/3$, $\mathit{fakeH}/2$, $\mathit{crseqv}/1$, $\mathit{crsext}/1$, destructor $\mathit{extract}/2$, equation $\mathit{hash}(\mathit{crseqv}(x_n),(x_m,\mathit{fake}(x_n,x_m,x_r))) =_{\mathrm{E}} \mathit{fakeH}(x_n,x_r)$, and rewrite rule $\mathit{extract}(x_n,\mathit{hash}(\mathit{crsext}(x_n),(x_m,x_r))) \to x_m$.

The Proverif code for this symbolic model is given in Figure 2.12.

Notice that if we have a CRS $\mathit{crseqv}(n)$ and know n, we can open $\mathit{fakeH}(n,r)$ to arbitrary values. Similarly, if the CRS is $\mathit{crsext}(n)$ and we know n, we can extract m from $\mathit{hash}(\mathit{crsext}(n),(m,r))$. These two facts allow us to construct a simulator that does equivocation and extraction.

Note that we introduced two different CRS-constructors for faking, *crsext* and *crseqv*. It would be tempting to use only one of them, i.e., use the equation $\mathit{hash}(\mathit{fakecrs}(x),(y,\mathit{fake}(x,y,z))) =_{\mathrm{E}} \mathit{fakeH}(x,z)$ and the reduction rule $\mathit{extract}(x,\mathit{hash}(\mathit{fakecrs}(x),(y,z))) \to y$. But then we would have for any terms k,m,r that $\mathit{extract}(k,\mathit{fakeH}(k,r)) =_{\mathrm{E}} \mathit{extract}(k,\mathit{hash}(\mathit{fakecrs}(k),(m,r))) \to m$, so by computing $\mathit{extract}(k,\mathit{fake}(k,r))$ the adversary can derive any term m, thus the adversary will know *all* secrets. This is clearly not a sensible symbolic model.

The commitment protocol.

The protocol we construct uses a crs, so we first need to define the crs functionality $\mathcal{F}_{\mathit{CRS}}$ that gives a random non-secret value k to Alice, Bob, and the adversary.

Definition 36 (Common reference string)**.** $\mathcal{F}_{\mathit{CRS}} := \nu k.\overline{io_{\mathit{crsa}}}\langle k\rangle \mid \overline{io_{\mathit{crsb}}}\langle k\rangle \mid \overline{net_{\mathit{crs}}}\langle k\rangle$.

Our protocol is then as expected. To commit to a message x_m, Alice fetches the crs x_{crs}, picks a random r, and sends $h := \mathit{hash}(x_{\mathit{crs}},(x_m,r))$ to Bob. To unveil, Alice sends (x_m,r), so that Bob can check whether h indeed contained these values. We call Alice's part of the protocol COM_A and Bob's part COM_B.

Definition 37 (Commitment protocol)**.**

$$\begin{aligned}
\mathsf{COM}_A &:= io_{crsa}(x_{crs}).io_{coma}(x_m).\\
&\quad \nu r.\Big(\overline{net_1}\langle hash(x_{crs},(x_m,r))\rangle\\
&\qquad |io_{opena}().\overline{net_2}\langle(x_m,r)\rangle\Big)\\
\mathsf{COM}_B &:= io_{crsb}(x_{crs}).net_1(x_h).\Big(\overline{io_{comb}}\langle\rangle|net_2((x_m,x_r)).\\
&\quad \mathit{if}\ x_h = hash(x_{crs},(x_m,x_r))\ \mathit{then}\ \overline{io_{openb}}\langle x_m\rangle\Big)\\
\mathsf{COM} &:= \nu io_{crsa}io_{crsb}net_1net_2.(\mathsf{COM}_A|\mathsf{COM}_B|\mathcal{F}_{CRS})
\end{aligned}$$

To show that COM is a secure commitment protocol, we need to show the following lemma (cf. also the discussion on how to model corruptions in Section 2.3):

Lemma 39. *With respect to $\mathcal{M}_{virt}$, we have*

(i) Uncorrupted case: $\mathsf{COM} \le \mathcal{F}_{COM}$.
(ii) Alice corrupted: $\nu io_{crsb}.(\mathsf{COM}_B|\mathcal{F}_{CRS}\{\frac{net_{crsa}}{io_{crsa}}\}) \le \mathcal{F}_{COM}\{\frac{net_{coma}}{io_{coma}},\frac{net_{opena}}{io_{opena}}\}$
(iii) Bob corrupted: $\nu io_{crsa}.(\mathsf{COM}_A|\mathcal{F}_{CRS}\{\frac{net_{crsb}}{io_{crsb}}\}) \le \mathcal{F}_{COM}\{\frac{net_{comb}}{io_{comb}},\frac{net_{openb}}{io_{openb}}\}$.

In the proof, we show the various observational equivalences by a sequence of rewriting steps on the protocol, interspersed with automated Proverif proofs for the steps that actually involve the symbolic model (i.e., we do not have to manually deal with the complex symbolic model $\mathcal{M}_{virt}$).

We split this lemma into the following three lemmas:

Lemma 40 (Commitment – uncorrupted case)**.** $\mathsf{COM} \le \mathcal{F}_{COM}$.

Proof.

$$\begin{aligned}
\mathsf{COM} \equiv\ & \nu io_{crsa} io_{crsb} net_1 net_2\, k\, r.\ \overline{io_{crsa}}\langle k\rangle \mid \overline{io_{crsb}}\langle k\rangle \mid \overline{net_{crs}}\langle k\rangle \\
& \quad \mid io_{crsa}(x_{crs}).io_{coma}(x_m).\big(\overline{net_1}\langle hash(x_{crs},(x_m,r))\rangle \mid io_{opena}().\overline{net_2}\langle (x_m,r)\rangle\big) \\
& \quad \mid io_{crsb}(x_{crs}).net_1(x_h).\big(\overline{io_{comb}}\langle\rangle | net_2((x'_m,x_r)). \\
& \qquad\qquad \text{if } x_h = hash(x_{crs},(x'_m,x_r)) \text{ then } \overline{io_{openb}}\langle x'_m\rangle\big) \\
\overset{(i)}{\approx}\ & \nu\ net_1 net_2\, k\, r.\ \overline{net_{crs}}\langle k\rangle \\
& \quad \mid io_{coma}(x_m).\big(\overline{net_1}\langle hash(k,(x_m,r))\rangle \mid io_{opena}().\overline{net_2}\langle (x_m,r)\rangle\big) \\
& \quad \mid net_1(x_h).\big(\overline{io_{comb}}\langle\rangle | net_2((x'_m,x_r)). \\
& \qquad\qquad \text{if } x_h = hash(k,(x'_m,x_r)) \text{ then } \overline{io_{openb}}\langle x'_m\rangle\big) \\
\overset{(ii)}{\approx}\ & \nu\ net_2\, k\, r.\ \overline{net_{crs}}\langle k\rangle \\
& \quad \mid io_{coma}(x_m).\big(\overline{io_{comb}}\langle\rangle \mid net_2((x'_m,x_r)). \\
& \qquad\qquad \text{if } hash(k,(x_m,r)) = hash(k,(x'_m,x_r)) \text{ then } \overline{io_{openb}}\langle x'_m\rangle \\
& \qquad\qquad \mid io_{opena}().\overline{net_2}\langle (x_m,r)\rangle\big) \\
\overset{(iii)}{=}\ & \nu net_2\, k\, r.\ \overline{net_{crs}}\langle k\rangle \\
& \quad \mid io_{coma}(x_m).\big(\overline{io_{comb}}\langle\rangle \mid net_2(x_{tmp}).\text{let } (x'_m,x_r) = z \text{ in} \\
& \qquad\qquad \text{if } hash(k,(x_m,r)) = hash(k,(x'_m,x_r)) \text{ then } \overline{io_{openb}}\langle x'_m\rangle \\
& \qquad\qquad \mid io_{opena}().\overline{net_2}\langle (x_m,r)\rangle\big) \\
\overset{(iv)}{\approx}\ & \nu\ k\, r.\ \overline{net_{crs}}\langle k\rangle \\
& \quad \mid io_{coma}(x_m).\big(\overline{io_{comb}}\langle\rangle \mid io_{opena}().\ \text{let } (x'_m,x_r) = (x_m,r) \text{ in} \\
& \qquad\qquad \text{if } hash(k,(x_m,r)) = hash(k,(x'_m,x_r)) \text{ then } \overline{io_{openb}}\langle x'_m\rangle\big) \\
\overset{(v)}{\approx}\ & \nu k\, r.\ \overline{net_{crs}}\langle k\rangle \mid io_{coma}(x_m).\big(\overline{io_{comb}}\langle\rangle \mid io_{opena}().\ \overline{io_{openb}}\langle x_m\rangle\big) \\
\equiv\ & \mathcal{F}_{COM} \mid S \quad \text{with} \quad S := \nu k\, r.\overline{net_{crs}}\langle k\rangle
\end{aligned}$$

Here (i) uses two invocations of Lemma 5, one with $n := io_{crsa}$, $t := k$, and $x := x_{crs}$, and one with $n := io_{crsb}$, $t := k$, and $x := x_{crs}$.

And (ii) uses one invocation of Lemma 5 with $n := net_1$, $x := x_h$, and $t := hash(k,(x_m,r))$.

And (iii) uses the fact that $t(p).P$ is syntactic sugar for $t(z).\text{let } p = z \text{ in } P$ for a pattern p and a fresh variable z.

And (iv) uses one invocation of Lemma 5 with $n := net_2$, $x := x_{tmp}$, and $t := (x_m,r)$. (And it uses Lemma 1, so that we can apply Lemma 5 to a subprocess instead of the whole process.)

And (v) uses several invocations of Lemma 4 (v) to evaluate the let- and the if-statement.

So $\mathsf{COM} \approx \mathcal{F}_{COM} \mid S$ for some S with $\mathrm{IO} \cap fn(S) = \varnothing$. Hence $\mathsf{COM} \leq \mathcal{F}_{COM}$. □

Lemma 41 (Commitment – Alice corrupted)**.**
$\nu io_{crsb}.(\mathsf{COM}_B | \mathcal{F}_{CRS}\{\frac{net_{crsa}}{io_{crsa}}\}) \leq \mathcal{F}_{COM}\{\frac{net_{coma}}{io_{coma}}, \frac{net_{opena}}{io_{opena}}\}$

```
free netcrs,netcrsa,net1,net2,iocomb,ioopenb.

process
new k;
out(netcrsa,choice[k,crsext(k)]) |
out(netcrs,choice[k,crsext(k)]) |
in(net1,xh);
out(iocomb,empty) |
in(net2,(xm,xr));
if xh = hash(choice[k,crsext(k)],(xm,xr)) then
out(ioopenb,choice[xm,extract(k,xh)])
```

Figure 2.13: Virtual primitives example: Proverif code for corrupted Alice (`virtprim-acorr.pv`, see [29]). (Has to be prefixed with the code from Figure 2.12.)

Proof. We have

$$
\begin{aligned}
&\nu io_{crsb}.(\mathsf{COM}_B|\mathcal{F}_{CRS}\{\tfrac{net_{crsa}}{io_{crsa}}\})\\
&\overset{(i)}{\approx} \nu k.\overline{net_{crsa}}\langle k\rangle \mid \overline{net_{crs}}\langle k\rangle \mid net_1(x_h).\Big(\overline{io_{comb}}\langle\rangle|\\
&\qquad net_2((x_m,x_r)).\text{if } x_h = hash(k,(x_m,x_r)) \text{ then } \overline{io_{openb}}\langle x_m\rangle\Big)\\
&\overset{(ii)}{\approx} \nu k.\overline{net_{crsa}}\langle crsext(k)\rangle \mid \overline{net_{crs}}\langle crsext(k)\rangle \mid net_1(x_h).\Big(\overline{io_{comb}}\langle\rangle|\\
&\qquad net_2((x_m,x_r)).\text{if } x_h = hash(crsext(k),(x_m,x_r)) \text{ then } \overline{io_{openb}}\langle extract(k,x_h)\rangle\Big)\\
&\overset{(iii)}{\approx} \nu k.\overline{net_{crsa}}\langle crsext(k)\rangle \mid \overline{net_{crs}}\langle crsext(k)\rangle \mid\\
&\qquad net_1(x_h).\nu net_{opena}.\Big(\overline{io_{comb}}\langle\rangle|net_{opena}().\overline{io_{openb}}\langle extract(k,x_h)\rangle|\\
&\qquad net_2((x_m,x_r)).\text{if } x_h = hash(crsext(k),(x_m,x_r)) \text{ then } \overline{net_{opena}}\langle\rangle\Big)\\
&\overset{(iv)}{\approx} \nu net_{opena}\, k.\overline{net_{crsa}}\langle crsext(k)\rangle \mid \overline{net_{crs}}\langle crsext(k)\rangle \mid\\
&\qquad net_1(x_h).\ \Big(\overline{io_{comb}}\langle\rangle|net_{opena}().\overline{io_{openb}}\langle extract(k,x_h)\rangle|\\
&\qquad net_2((x_m,x_r)).\text{if } x_h = hash(crsext(k),(x_m,x_r)) \text{ then } \overline{net_{opena}}\langle\rangle\Big)\\
&\overset{(v)}{\approx} \nu net_{coma}\, net_{opena}k.\overline{net_{crsa}}\langle crsext(k)\rangle \mid \overline{net_{crs}}\langle crsext(k)\rangle \mid\\
&\qquad net_1(x_h).\Big(\overline{net_{coma}}\langle extract(k,x_h)\rangle|\\
&\qquad net_2((x_m,x_r)).\text{if } x_h = hash(crsext(k),(x_m,x_r)) \text{ then } \overline{net_{opena}}\langle\rangle\Big)|\\
&\qquad net_{coma}(x'_m).\Big(\overline{io_{comb}}\langle\rangle|net_{opena}().\overline{io_{openb}}\langle x'_m\rangle\Big)\\
&\equiv \nu net_{coma}\, net_{opena}.(\mathcal{F}_{COM}\{\tfrac{net_{coma}}{io_{coma}},\tfrac{net_{opena}}{io_{opena}}\}|S) \text{ for some } S \text{ with } \mathrm{IO}\cap fn(S)=\varnothing.
\end{aligned}
$$

Here (i) uses Lemma 5 with $n := io_{crsb}$, $C := \nu k.\overline{net_{crsa}}\langle k\rangle \mid \overline{net_{crs}}\langle k\rangle \mid \Box$, $x := x_{crs}$, and $t := k$.

And (ii) is shown using Proverif, the required code is given in Figure 2.13. Note that in the rhs of (ii), we have replaced all occurrences of the CRS k by $crsext(k)$, and instead of outputting x_m in the end, we output $extract(k, x_h)$.

And (iii) uses Lemma 5 (in the opposite direction) with $n := net_{opena}$, $Q :=$

$\overline{io_{openb}}\langle extract(k,x_h)\rangle$, and $C := \overline{io_{comb}}\langle\rangle|net_2((x_m,x_r)).\text{if } x_h = hash(crsext(k),(x_m,x_r)) \text{ then } \square$. (And it uses Lemma 1, so that we can apply Lemma 5 to a subprocess instead of the whole process.)

And (iv) uses Lemma 4 (ii) to swap νnet_{opena} and $net_1(x_h)$. (And Lemma 1 to apply Lemma 4 (ii) to a subprocess.)

And (v) uses Lemma 5 (in the opposite direction) with $n := net_{coma}$, $x := x'_m$, $t := extract(k,x_h)$, and $Q := \overline{io_{comb}}\langle\rangle|net_{opena}().\overline{io_{openb}}\langle x'_m\rangle$.

So we have

$$\nu io_{crsb}.(\mathsf{COM}_B|\mathcal{F}_{CRS}\{\tfrac{net_{crsa}}{io_{crsa}}\}) \approx \nu net_{coma}net_{opena}.(\mathcal{F}_{COM}\{\frac{net_{coma}}{io_{coma}},\frac{net_{opena}}{io_{opena}}\}|S)$$

for some S with $\mathrm{IO}\cap fn(S)=\varnothing$. Hence

$$\nu io_{crsb}.(\mathsf{COM}_B|\mathcal{F}_{CRS}\{\tfrac{net_{crsa}}{io_{crsa}}\}) \leq \mathcal{F}_{COM}\{\frac{net_{coma}}{io_{coma}},\frac{net_{opena}}{io_{opena}}\}$$

. □

Lemma 42 (Commitment – Bob corrupted)**.** $\nu io_{crsa}.(\mathsf{COM}_A|\mathcal{F}_{CRS}\{\tfrac{net_{crsb}}{io_{crsb}}\}) \leq \mathcal{F}_{COM}\{\tfrac{net_{comb}}{io_{comb}},\tfrac{net_{openb}}{io_{openb}}\}$.

Proof. We have

$$\begin{aligned}
&\nu io_{crsa}.(\mathsf{COM}_A|\mathcal{F}_{CRS}\{\tfrac{net_{crsb}}{io_{crsb}}\})\\
&\overset{\text{(i)}}{\approx} \nu k.\overline{net_{crsb}}\langle k\rangle \mid \overline{net_{crs}}\langle k\rangle \mid\\
&\qquad io_{coma}(x_m).\nu r.\Big(\overline{net_1}\langle hash(k,(x_m,r))\rangle|io_{opena}().\overline{net_2}\langle(x_m,r)\rangle\Big)\\
&\overset{\text{(ii)}}{\approx} \nu k.\overline{net_{crsb}}\langle crseqv(k)\rangle \mid \overline{net_{crs}}\langle crseqv(k)\rangle \mid io_{coma}(x_m).\nu r.\\
&\qquad\Big(\overline{net_1}\langle fakeH(k,r)\rangle|io_{opena}().\overline{net_2}\langle(x_m,fake(k,x_m,r))\rangle\Big)\\
&\overset{\text{(iii)}}{\approx} \nu k.\overline{net_{crsb}}\langle crseqv(k)\rangle \mid \overline{net_{crs}}\langle crseqv(k)\rangle \mid io_{coma}(x_m).\nu r.\\
&\qquad \nu net_{openb}.\Big(\overline{net_1}\langle fakeH(k,r)\rangle|io_{opena}().\\
&\qquad \overline{net_{openb}}\langle x_m\rangle|net_{openb}(x'_m).\overline{net_2}\langle(x'_m,fake(k,x'_m,r))\rangle\Big)\\
&\overset{\text{(iv)}}{\approx} \nu net_{openb}\,k\,r.\overline{net_{crsb}}\langle crseqv(k)\rangle \mid \overline{net_{crs}}\langle crseqv(k)\rangle \mid io_{coma}(x_m).\\
&\quad \Big(\overline{net_1}\langle fakeH(k,r)\rangle|io_{opena}().\overline{net_{openb}}\langle x_m\rangle|net_{openb}(x'_m).\overline{net_2}\langle(x'_m,fake(k,x'_m,r))\rangle\Big)\\
&\overset{\text{(v)}}{\approx} \nu net_{comb}\,net_{openb}\,k\,r.\overline{net_{crsb}}\langle crseqv(k)\rangle \mid \overline{net_{crs}}\langle crseqv(k)\rangle \mid io_{coma}(x_m).\\
&\qquad \Big(io_{opena}().\overline{net_{openb}}\langle x_m\rangle|\overline{net_{comb}}\langle\rangle\Big) \mid\\
&\qquad net_{comb}().\Big(\overline{net_1}\langle fakeH(k,r)\rangle|net_{openb}(x'_m).\overline{net_2}\langle(x'_m,fake(k,x'_m,r))\rangle\Big)\\
&\equiv \nu net_{comb}net_{openb}.(\mathcal{F}_{COM}\{\tfrac{net_{comb}}{io_{comb}},\tfrac{net_{openb}}{io_{openb}}\}|S) \text{ for some } S \text{ with } \mathrm{IO}\cap fn(S)=\varnothing.
\end{aligned}$$

Here (i) uses Lemma 5 with $n := io_{crsa}$, $C := \nu k.\overline{net_{crsb}}\langle k\rangle \mid \overline{net_{crs}}\langle k\rangle \mid \square$, $x := x_{crs}$, and $t := k$.

And (ii) is shown using Proverif, the required code is given in Figure 2.14. Note that in the rhs of (ii), we have replaced all occurrences of the CRS k by $crseqv(k)$, and instead of sending the hash value $hash(k,(x_m,r))$ we send $fakeH(k,r)$ which does

```
free netcrs,netcrsb,net1,net2,iocoma,ioopena.

process
new k;
out(netcrs,choice[k,crseqv(k)]) |
out(netcrsb,choice[k,crseqv(k)]) |
in(iocoma,xm);
new r;
out(net1,choice[hash(k,(xm,r)),fakeH(k,r)]) |
in(ioopena,x);
out(net2,(xm,choice[r,fake(k,xm,r)]))
```

Figure 2.14: Virtual primitives example: Proverif code for corrupted Bob (`virtprim-bcorr.pv`, see [29]). (Has to be prefixed with the code from Figure 2.12.)

not depend on x_m, and in the end, instead of sending the randomness r, we send $\mathit{fake}(k, x_m, r)$. Intuitively, this replacement is indistinguishable because our symbolic model contains the equation $\mathit{hash}(\mathit{crseqv}(k), (m, \mathit{fake}(k, m, r))) =_{\mathrm{E}} \mathit{fakeH}(k, r)$.

And (iii) uses Lemma 5 (in the opposite direction) with $n := \mathit{net}_{\mathit{openb}}$, $x := x'_m$, $t := x_m$, $Q := \overline{\mathit{net}_2}\langle(x'_m, \mathit{fake}(k, x'_m, r))\rangle$, and $C := \overline{\mathit{net}_1}\langle \mathit{fakeH}(k, r)\rangle \mid \mathit{io}_{\mathit{opena}}().\Box$. (And it uses Lemma 1, so that we can apply Lemma 5 to a subprocess instead of the whole process.)

And (iv) uses Lemma 4 (ii) to swap νr and $\nu \mathit{net}_{\mathit{openb}}$ with $\mathit{io}_{\mathit{coma}}(x_m)$. (And Lemma 1 to apply Lemma 4 (ii) to a subprocess.)

And (v) uses Lemma 5 (in the opposite direction) with $n := \mathit{net}_{\mathit{comb}}$, $t := \mathit{empty}$, and $Q := \overline{\mathit{net}_1}\langle \mathit{fakeH}(k, r)\rangle \mid \mathit{net}_{\mathit{openb}}(x'_m).\overline{\mathit{net}_2}\langle(x'_m, \mathit{fake}(k, x'_m, r))\rangle$.

So we have

$$\nu \mathit{io}_{\mathit{crsa}}.(\mathsf{COM}_A|\mathcal{F}_{\mathit{CRS}}\{\tfrac{\mathit{net}_{\mathit{crsb}}}{\mathit{io}_{\mathit{crsb}}}\}) \approx \nu \mathit{net}_{\mathit{comb}}\mathit{net}_{\mathit{openb}}.(\mathcal{F}_{\mathit{COM}}\{\tfrac{\mathit{net}_{\mathit{comb}}}{\mathit{io}_{\mathit{comb}}}, \tfrac{\mathit{net}_{\mathit{openb}}}{\mathit{io}_{\mathit{openb}}}\}|S)$$

for some S with $\mathrm{IO} \cap \mathit{fn}(S) = \varnothing$. Hence $\nu \mathit{io}_{\mathit{crsa}}.(\mathsf{COM}_A|\mathcal{F}_{\mathit{CRS}}\{\tfrac{\mathit{net}_{\mathit{crsb}}}{\mathit{io}_{\mathit{crsb}}}\}) \leq \mathcal{F}_{\mathit{COM}}\{\tfrac{\mathit{net}_{\mathit{comb}}}{\mathit{io}_{\mathit{comb}}}, \tfrac{\mathit{net}_{\mathit{openb}}}{\mathit{io}_{\mathit{openb}}}\}$. □

2.8.1.1 A note on adaptive corruption

We have only modeled static corruption in our examples, i.e., it is fixed in the beginning of the execution which parties are corrupted. If we were to model adaptive corruption where parties may be corrupted during the protocol execution, we would face an additional challenge (besides the fact that the descriptions of the processes would be much more complex): Since the simulator may have to provide the CRS before he knows whether Alice or Bob will be corrupted, he will not know whether he should use $\mathit{crseqv}(k)$ or $\mathit{crsext}(k)$ as CRS. And on page 84 we explained why we cannot just replace both *crseqv* and *crsext* by a single constructor *fakecrs* because then the adversary would be able to deduce any term. However, this problem can be solved using the conditional destructors supported by Proverif 1.87: we can make sure that the rewrite rule $\mathit{extract}(x_n, \mathit{hash}(\mathit{crsext}(x_n), (x_m, x_r))) \to x_m$ only triggers if $\mathit{hash}(\mathit{crsext}(x_n), (x_m, x_r)) \neq_{\mathrm{E}} \mathit{fakeH}(M, M')$ for all M, M'. The resulting symbolic model is shown in Figure 2.15. We can show Lemmas 40, 41, and 42 also using this

```
(* Needs proverif1.87 beta *)

fun hash(bitstring,bitstring):bitstring.
const empty:bitstring.
fun fake(bitstring,bitstring,bitstring):bitstring.
fun fakeH(bitstring,bitstring):bitstring.
fun fakecrs(bitstring):bitstring.

equation forall n:bitstring,m:bitstring,r:bitstring;
  hash(fakecrs(n),(m,fake(n,m,r))) = fakeH(n,r).

fun extract(bitstring,bitstring):bitstring
reduc forall n:bitstring,r:bitstring;
    extract(n,fakeH(n,r)) = empty
otherwise forall n:bitstring,m:bitstring,r:bitstring;
    extract(n,hash(fakecrs(n),(m,r))) = m.
```

Figure 2.15: Virtual primitives example: Proverif code for the symbolic model when using *fakecrs* constructor (`virtprim-model-x.pv`, see [29]). Note that we use the typed Proverif syntax here because Proverif 1.87 does not support conditional destructors in the untyped syntax.

symbolic model by replacing all occurrences of *crseqv* and *crsext* in the simulators by *fakecrs*. Proverif still shows all the necessary equivalences. Although this does not show adaptive security, it shows that the simulator does not need to choose the CRS depending on who is corrupted, giving hope for the adaptive case. We leave that case for future work.

2.8.2 Removing the virtual primitives

In this section, we will consider different symbolic models. Since the relation symbols $\rightarrow, \Downarrow, \approx, \downarrow, =_{\mathrm{E}}$ etc. do not explicitly specify the symbolic model, we use the following convention: When referring to a symbolic model $\mathcal{M}_i$, we write $\rightarrow_i, \Downarrow_i, \approx_i, \downarrow^i$, $=_{E_i}$ etc. We say a term (or destructor term) is an *$\mathcal{M}$-term* (or $\mathcal{M}$-destructor term) if it contains only constructors (and destructors) from $\mathcal{M}$. We call a process an *$\mathcal{M}$-process* if it contains only $\mathcal{M}$-terms and $\mathcal{M}$-destructor terms.

We have now shown that COM is a secure commitment protocol with respect to $\mathcal{M}_{\mathit{virt}}$. However, we would like to deduce security of protocols using COM with respect to $\mathcal{M}_{\mathit{real}}$. For this, we first need to formalize what it means that $\mathcal{M}_{\mathit{virt}}$ is a safe extension of $\mathcal{M}_{\mathit{real}}$:

Definition 38 (Safe extension). *We call a symbolic model $\mathcal{M}_1 = (\Sigma_1, \mathrm{E}_1, \mathrm{R}_1)$ a* safe extension *of a symbolic model $\mathcal{M}_2 = (\Sigma_2, \mathrm{E}_2, \mathrm{R}_2)$ iff the following holds:*

(i) $\Sigma_1 \supseteq \Sigma_2$.

(ii) If D is an $\mathcal{M}_2$-destructor term, and M is an $\mathcal{M}_1$-term, and $D \Downarrow_1 M$, then there exists an $\mathcal{M}_2$-term $M' =_{E_1} M$ with $D \Downarrow_2 M'$.

(iii) For all $\mathcal{M}_2$-destructor terms D and $\mathcal{M}_2$-terms M, we have $D \Downarrow_2 M \Rightarrow D \Downarrow_1 M$.

(iv) For all $\mathcal{M}_2$-terms M, M' we have $M =_{E_1} M' \Leftrightarrow M =_{E_2} M'$.

The following lemma is relatively easy to show:

Lemma 43. *$\mathcal{M}_{virt}$ is a safe extension of $\mathcal{M}_{real}$.*

Proof. Obviously, $\Sigma_{virt} \supseteq \Sigma_{real}$. So Definition 38 (i) is satisfied.

We show that Definition 38 (ii) is satisfied: Let D be an $\mathcal{M}_{real}$-destructor term and M be an $\mathcal{M}_{virt}$-term. Since $\mathcal{M}_{real}$ contains no destructors, D is an $\mathcal{M}_{real}$-term. Thus $D \Downarrow_{virt} M$ implies $D = M$. This implies that $M' := M$ is an $\mathcal{M}_{real}$-term and $D \Downarrow_{real} M'$.

We show that Definition 38 (iii) is satisfied: Let D be an $\mathcal{M}_{real}$-destructor term and M be an $\mathcal{M}_{real}$-term. Since $\mathcal{M}_{real}$ contains no destructors, D is an $\mathcal{M}_{real}$-term. Thus $D \Downarrow_{virt} M$ implies $D = M$ which implies $D \Downarrow_{real} M$.

We show that Definition 38 (iv) is satisfied: For $\mathcal{M}_{real}$-terms M, M', obviously $M =_{E_{real}} M'$ implies $M =_{E_{virt}} M'$. We show the opposite direction: The only equation in E_{virt} (namely $hash(crseqv(k), (m, fake(k, m, r))) =_{\mathrm{E}} fakeH(k, r)$) only allows us to rewrite terms containing *crseqv* or *fakeH*. Since M, M' are $\mathcal{M}_{real}$-terms, they do not contain these constructors. Hence $M =_{E_{virt}} M'$ only if $M = M'$. So $M =_{E_{virt}} M'$ implies $M =_{E_{real}} M'$. □

The following theorem justifies the above definition of safe extensions:

Theorem 3. *Assume that $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$. Then for all $\mathcal{M}_2$-processes P, P' we have $P \approx_1 P' \Rightarrow P \approx_2 P'$.*

Proof. We first show some auxiliary claims:

Claim 1. *For all $\mathcal{M}_2$-processes P, P', we have $P \rightarrow_2 P' \Rightarrow P \rightarrow_1 P'$.*

We show this claim by induction over the derivation of $P \rightarrow_2 P'$. We distinguish the following cases:

- *Closure under structural equivalence:* In this case $P \rightarrow_2 P'$ has been derived from $P \equiv \hat{P} \rightarrow_2 \hat{P}' \equiv P'$ for $\mathcal{M}_2$-processes $\hat{P}, \hat{P}'$, and the induction hypothesis implies $\hat{P} \rightarrow_1 \hat{P}'$. Thus $P \equiv \hat{P} \rightarrow_1 \hat{P}' \equiv P'$ which implies $P \rightarrow_1 P'$. The claim follows.
- *Closure under evaluation contexts:* In this case $P \rightarrow_2 P'$ has been derived from $P = E[\hat{P}]$, $P' = E[\hat{P}']$, and $\hat{P} \rightarrow_2 \hat{P}'$ for some $\mathcal{M}_2$-processes $\hat{P}, \hat{P}'$ and some $\mathcal{M}_2$-evaluation context E. The induction hypothesis implies $\hat{P} \rightarrow_1 \hat{P}'$. Hence $P = E[\hat{P}] \rightarrow_1 E[\hat{P}'] = P'$.
- *REPL:* In this case $P = !\hat{P}$ and $P' = \hat{P}|!\hat{P}$. Hence $P \rightarrow_1 P'$.
- *COMM:* In this case $P = \overline{C}\langle T\rangle.\hat{P} \mid C'(x).\hat{Q}$ and $P' = \hat{P} \mid Q\{T/x\}$ and $C =_{E_2} C'$. Since P is an $\mathcal{M}_2$-process, C, C' are $\mathcal{M}_2$-terms. Since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$, $C =_{E_2} C'$ implies $C =_{E_1} C'$. Thus $P \rightarrow_1 P'$. The claim follows.
- *LET-THEN:* In this case $P = (\text{let } x = D \text{ in } \hat{P} \text{ else } \hat{Q})$ and $P' = \hat{P}\{M/x\}$ for some $\mathcal{M}_2$-processes $\hat{P}, \hat{Q}$, and some $\mathcal{M}_2$-destructor term D and $\mathcal{M}_2$-term M with $D \Downarrow_2 M$. Since P is an $\mathcal{M}_2$-process, D is an $\mathcal{M}_2$-destructor term. Since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$, $D \Downarrow_2 M$ implies that $D \Downarrow_1 M$. Thus $P \rightarrow_1 P'$. The claim follows.
- *LET-ELSE:* In this case $P = (\text{let } x = D \text{ in } \hat{P} \text{ else } \hat{Q})$ and $P' = \hat{Q}$ and for all $\mathcal{M}_2$-terms M we have $D \not\Downarrow_2 M$. Since P is an $\mathcal{M}_2$-process, D is an $\mathcal{M}_2$-destructor term. If we had $D \Downarrow_1 M$ for some $\mathcal{M}_1$-term M, we would have $D \Downarrow_2 M'$ for some $\mathcal{M}_2$-term M' since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$. This

contradicts $D \not\Downarrow_2 M$ for all $\mathcal{M}_2$-terms M. Thus $D \not\Downarrow_1 M$ for all $\mathcal{M}_1$-terms M. Hence $P \to_1 P'$. The claim follows.

Claim 2. *For all $\mathcal{M}_2$-processes P, and all $\mathcal{M}_1$-processes P'' with $P \to_1 P''$, there exists an $\mathcal{M}_2$-process P' such that $P \to_2 P' \equiv_{E_1} P''$.*

We show this claim by induction over the derivation of $P \to_1 P''$. We distinguish the following cases:

- *Closure under structural equivalence:* In this case $P \to_1 P''$ has been derived from $P \equiv \hat{P} \to_1 \hat{P}'' \equiv P''$ for $\mathcal{M}_1$-processes $\hat{P}, \hat{P}''$, and the induction hypothesis (Claim 2) holds for $\hat{P} \to_1 \hat{P}''$. Since structural equivalence does not rewrite terms, the fact that P is an $\mathcal{M}_2$-process implies that $\hat{P}$ is an $\mathcal{M}_2$-process. Thus $\hat{P} \to_1 \hat{P}''$ implies together with the induction hypothesis that $\hat{P} \to_2 P' \equiv_{E_1} \hat{P}''$ for some $\mathcal{M}_2$-process P'. Thus $P \equiv \hat{P} \to_2 P'$ which implies $P \to_2 P'$ and we have $P' \equiv_{E_1} \hat{P}'' \equiv P''$ which implies $P' \equiv_{E_1} P''$. The claim follows.
- *Closure under evaluation contexts:* In this case $P \to_1 P''$ has been derived from $P = E[\hat{P}]$, $P'' = E[\hat{P}'']$, and $\hat{P} \to_1 \hat{P}''$ for some $\mathcal{M}_1$-processes $\hat{P}, \hat{P}''$ and some $\mathcal{M}_1$-evaluation context E. And the induction hypothesis holds for $\hat{P} \to_1 \hat{P}''$. Since P is an $\mathcal{M}_2$-process and $P = E[\hat{P}]$, we have that $\hat{P}$ is an $\mathcal{M}_2$-process and E and $\mathcal{M}_2$-evaluation context. Thus by induction hypothesis, there exists an $\mathcal{M}_2$-process $\hat{P}'$ such that $\hat{P} \to_2 \hat{P}' \equiv_{E_1} \hat{P}''$. Let $P' := E[\hat{P}']$. Obviously P' is an $\mathcal{M}_2$-process. And $P = E[\hat{P}] \to_2 E[\hat{P}'] = P'$ and $P'' = E[\hat{P}''] \equiv_{E_1} E[\hat{P}'] = P'$. The claim follows.
- *REPL:* In this case $P = !\hat{P}$ and $P'' = \hat{P}|!\hat{P}$. Since P is an $\mathcal{M}_2$-process, so is $\hat{P}$, and hence also $P' := P''$ is an $\mathcal{M}_2$-process. Then $P \to_2 P'$ and $P'' \equiv_{E_1} P'$ and the claim follows.
- *COMM:* In this case $P = \overline{C}\langle T\rangle.\hat{P} \mid C'(x).\hat{Q}$ and $P'' = \hat{P} \mid \hat{Q}\{T/x\}$ and $C =_{E_1} C'$. Since P is an $\mathcal{M}_2$-process, C, C' are $\mathcal{M}_2$-terms and $\hat{P}, \hat{Q}$ are $\mathcal{M}_2$-processes. Since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$, $C =_{E_1} C'$ implies $C =_{E_2} C'$. Thus $P \to_2 P''$. With $P' := P''$, the claim follows.
- *LET-THEN:* In this case $P = (\text{let } x = D \text{ in } \hat{P} \text{ else } \hat{Q})$ and $P'' = \hat{P}\{M/x\}$ for some $\mathcal{M}_1$-processes $\hat{P}, \hat{Q}$, and some $\mathcal{M}_1$-destructor term D and $\mathcal{M}_1$-term M with $D \Downarrow_1 M$. Since P is an $\mathcal{M}_2$-process, $\hat{P}, \hat{Q}$ are $\mathcal{M}_2$-processes and D is an $\mathcal{M}_2$-destructor term. Since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$, $D \Downarrow_1 M$ implies that $D \Downarrow_2 M'$ for some $\mathcal{M}_2$-term $M' =_{E_1} M$. Let $P' := \hat{P}\{M/x\}$. Then $P'' = \hat{P}\{M/x\} \equiv_{E_1} \hat{P}\{M'/x\} = P'$ and $P \to_2 P'$. The claim follows.
- *LET-ELSE:* In this case $P = (\text{let } x = D \text{ in } \hat{P} \text{ else } \hat{Q})$ and $P'' = \hat{Q}$ and for all $\mathcal{M}_1$-terms M we have $D \not\Downarrow_1 M$. Since P is an $\mathcal{M}_2$-process, $\hat{P}, \hat{Q}$ are $\mathcal{M}_2$-processes and D is an $\mathcal{M}_2$-destructor term. Since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$, for all $\mathcal{M}_2$-terms M, $D \not\Downarrow_1 M$ implies that $D \not\Downarrow_2 M$. With $P' := \hat{Q} = P''$, we thus have $P'' \equiv_{E_1} P'$ and $P \to_2 P'$. The claim follows.

Claim 3. *For all $\mathcal{M}_2$-processes P, and all $\mathcal{M}_1$-processes P'' with $P \to_1^* P''$, there exists an $\mathcal{M}_2$-process P' such that $P \to_2^* P' \equiv_{E_1} P''$.*

Proof. To show this claim, we show that for all $n \geq 0$, all $\mathcal{M}_2$-processes P, and all $\mathcal{M}_1$-processes P'' with $P \to_1^n P''$, there exists an $\mathcal{M}_2$-process P' such that $P \to_2^* P' \equiv_{E_1} P''$. Here $\to_1^n$ means exactly n applications of $\to$. We show this by induction over n. For $n = 0$, the statement is trivial. Assume the statement holds for n, we show it for $n+1$: We have $P \to_1^{n+1} P''$ hence $P \to_1^n \hat{P}'' \to_1 P''$ for some $\mathcal{M}_1$-process

$\hat{P}''$. By induction hypothesis there exists an $\mathcal{M}_2$-process $\hat{P}'$ with $P \to_2^* \hat{P}' \equiv_{E_1} \hat{P}''$. Since $\hat{P}' \equiv_{E_1} \hat{P}'' \to_1 P''$, by Lemma 7, we have $\hat{P}' \to_1 P_2 \equiv_{E_1} P''$ for some $\mathcal{M}_1$-process P_2. Since $\hat{P}'$ is an $\mathcal{M}_2$-process and $\hat{P}' \to_1 P_2$, by Claim 2, there is an $\mathcal{M}_2$-process P' such that $\hat{P}' \to_2 P' \equiv_{E_1} P_2$. Combining all this, we have

$$P \to_2^* \hat{P}' \to_2 P' \equiv_{E_1} P_2 \equiv_{E_1} P''.$$

Thus $P \to_2^* P' \equiv_{E_1} P''$. □

We are now ready to show Theorem 3. Let $\mathcal{R} := \{(P,Q) : P, Q\ \mathcal{M}_2\text{-processes}, P \approx_1 Q\}$. We show that $\mathcal{R}$ is an $\mathcal{M}_2$-simulation (and due to its symmetry also an $\mathcal{M}_2$-bisimulation):

- If $(P,Q) \in \mathcal{R}$ and $P \downarrow_M^2$ for some $\mathcal{M}_2$-term M, then $Q \to_2^* Q' \downarrow_M^2$ for some $\mathcal{M}_2$-process Q'.

  $P \downarrow_M^2$ implies (see Footnote 5) $P \equiv_{E_2} E[\overline{M}\langle T\rangle.P']$ for some evaluation context E not binding $\mathit{fn}(M)$. This implies $P \equiv_{E_1} E[\overline{M}\langle T\rangle.P']$ (since $M_1 =_{E_2} M_2$ implies $M_1 =_{E_1} M_2$ for $\mathcal{M}_2$-terms M_1, M_2). Thus $P \downarrow_M^1$. Since $(P,Q) \in \mathcal{R}$, we have that $P \approx_1 Q$ and thus $Q \to_1^* Q'' \downarrow_M^1$ for some $\mathcal{M}_1$-process Q''. By Claim 3, this implies that $Q \to_2^* Q' \equiv_{E_1} Q''$ for some $\mathcal{M}_2$-process Q'. Since $Q'' \equiv_{E_1} Q'' \downarrow_M^1$, we have $Q' \downarrow_M^1$ (this follows immediately using the characterization from Footnote 5). Since $Q' \downarrow_M^1$, by definition of $\downarrow$, we have $Q' \equiv E[\overline{M'}\langle T'\rangle.\tilde{Q}]$ for some $\mathcal{M}_1$-terms M', T' with $M' =_{E_1} M$ and $\mathcal{M}_1$-process $\tilde{Q}$, and some evaluation context not binding $\mathit{fn}(M)$. Since Q' is an $\mathcal{M}_2$-process, $E[\overline{M'}\langle T'\rangle.\tilde{Q}]$ is an $\mathcal{M}_2$-process, hence M' is an $\mathcal{M}_2$-term. Thus M, M' are $\mathcal{M}_2$-terms, and $M' =_{E_1} M$. Since $\mathcal{M}_1$ is a safe extension of $\mathcal{M}_2$, this implies $M' =_{E_2} M$. Thus $Q' \equiv E[\overline{M'}\langle T'\rangle.\tilde{Q}]$ implies $Q' \downarrow_M^2$. So we have $Q \to_2^* Q' \downarrow_M^2$ and Q' is an $\mathcal{M}_2$-process.

- If $(P,Q) \in \mathcal{R}$ and $P \to_2 P'$ for an $\mathcal{M}_2$-process P', then there exists an $\mathcal{M}_2$-process Q' with $(P',Q') \in \mathcal{R}$ and $Q \to_2^* Q'$:

  Since P, P' are $\mathcal{M}_2$-processes, and $P \to_2 P'$, by Claim 1 we have $P \to_1 P'$. Since $(P,Q) \in \mathcal{R}$, we have $P \approx_1 Q$ and thus $Q \to_1^* Q''$ for some $\mathcal{M}_1$-process $Q'' \approx_1 P'$. By Claim 3, there is an $\mathcal{M}_2$-process Q' such that $Q'' \equiv_{E_1} Q'$ and $Q \to_2^* Q'$. Furthermore, by Lemma 4 (iv), we have $=_{E_1} \subseteq \approx_1$ and trivially $\equiv \subseteq \approx_1$, hence $\equiv_{E_1} \subseteq \approx_1$. Thus $Q'' \equiv_{E_1} Q'$ implies $Q'' \approx_1 Q'$. Together with $Q'' \approx_1 P'$, we have $P' \approx_1 Q'$ and thus $(P',Q') \in \mathcal{R}$.

- If $(P,Q) \in \mathcal{R}$ and E is an $\mathcal{M}_2$-evaluation context, then $(E[P], E[Q]) \in \mathcal{R}$.

  Since $(P,Q) \in \mathcal{R}$, we have $P \approx_1 Q$. Furthermore, since E is an $\mathcal{M}_2$-evaluation context, E is also an $\mathcal{M}_1$-evaluation context. Hence $E[P] \approx_1 E[Q]$ and thus $(E[P], E[Q]) \in \mathcal{R}$.

Since $\mathcal{R}$ is a $\mathcal{M}_2$-bisimulation, $\mathcal{R} \subseteq \approx_2$. Thus for $\mathcal{M}_2$-terms P, P' we have $P \approx_1 P' \Rightarrow (P,P') \in \mathcal{R} \Rightarrow P \approx_2 P'$. Theorem 3 follows. □

Now we can finally state the following result that derives security of COM with respect to $\mathcal{M}_{\mathit{real}}$ in any context (we state it generally, though):

Lemma 44. *Let $P, \mathcal{F}$ be $\mathcal{M}_{real}$-processes (representing a protocol and an ideal functionality, e.g., $P = \mathsf{COM}$ and $\mathcal{F} = \mathcal{F}_{COM}$). Let $\mathcal{M}_{virt}$ be a safe extension of $\mathcal{M}_{real}$. Assume that $P \leq_{virt} \mathcal{F}$.*

Let C be an $\mathcal{M}_{real}$-context whose hole is protected only by νio for IO-names io, by parallel compositions, and by !, and that does not contain any NET-names in $fn(P, \mathcal{F})$. Assume that $C[\mathcal{F}] \leq_{virt} \mathcal{G}$ for some $\mathcal{M}_{real}$-process $\mathcal{G}$.

Let E_1, E_2 be $\mathcal{M}_{real}$-contexts satisfying the conditions of Theorem 2 (property preservation).

If $E_1[\mathcal{G}] \approx_{virt} E_2[\mathcal{G}]$ then $E_1[C[P]] \approx_{real} E_2[C[P]]$.

Proof. By the composition theorem (Theorem 1), $P \leq_{virt} \mathcal{F}$ implies $C[P] \leq_{virt} C[\mathcal{F}]$. With transitivity and $C[\mathcal{F}] \leq_{virt} \mathcal{G}$, this implies $C[P] \leq_{virt} \mathcal{G}$. Then by the property preservation theorem (Theorem 2), $E_1[\mathcal{G}] \approx_{virt} E_2[\mathcal{G}]$ implies $E_1[C[P]] \approx_{virt} E_2[C[P]]$. Since $\mathcal{M}_{virt}$ is a safe extension of $\mathcal{M}_{real}$, this implies $E_1[C[P]] \approx_{real} E_2[C[P]]$ by Theorem 3. □

2.8.3 On removing the CRS

Using virtual primitives, we have managed to get rid of the need for trapdoors in our commitment protocol. However, we still use a common reference string. This leads to the question whether the CRS can also be removed from the protocol. We do not answer that question here, but we give some indications as to how it might be possible to remove the CRS, also.

First, the question is whether we can construct a UC secure commitment protocol without using a CRS in the first place (i.e., instead of the protocol from Section 2.8.1). We know that this is impossible in the computational UC setting (no matter what primitives we use) [37]. Unfortunately, their impossibility result carries over to the symbolic setting:

Lemma 45. *There are no closed processes A, B and NET-names $\underline{net}$ with the following three properties:*

(i) $\nu\underline{net}.(A|B) \leq \mathcal{F}_{COM}$. (Uncorrupted case.)
(ii) $A \leq \mathcal{F}_{COM}\{\frac{net_{comb}}{io_{comb}}, \frac{net_{openb}}{io_{openb}}\}$. (Bob corrupted.)
(iii) $B \leq \mathcal{F}_{COM}\{\frac{net_{coma}}{io_{coma}}, \frac{net_{opena}}{io_{opena}}\}$. (Alice corrupted.)

Thus, a UC secure commitment protocol has to be of the form $\nu\underline{net}.(A|B|\mathcal{F})$ for some functionality $\mathcal{F}$, e.g., $\mathcal{F}_{CRS}$.

Proof. Assume that there are such processes A, B and NET-names $\underline{net}$.

Then there are simulators $(S_0, \varphi_0, \underline{n}_0)$, $(S_A, \varphi_A, \underline{n}_A)$, and $(S_B, \varphi_B, \underline{n}_B)$ such that

$$\nu\underline{net}.(A|B) \approx \nu\underline{n}_0.(\mathcal{F}_{COM}\varphi_0|S_0) = \nu\underline{n}_0.(\mathcal{F}_{COM}|S_0) \tag{2.5}$$

$$A \approx \nu\underline{n}_A.(\mathcal{F}_{COM}\{\tfrac{net_{comb}}{io_{comb}}, \tfrac{net_{openb}}{io_{openb}}\}\varphi_A|S_A) = \nu\underline{n}_A.(\mathcal{F}_{COM}\{\tfrac{net'_{comb}}{io_{comb}}, \tfrac{net'_{openb}}{io_{openb}}\}|S_A) \tag{2.6}$$

$$B \approx \nu\underline{n}_B.(\mathcal{F}_{COM}\{\tfrac{net_{coma}}{io_{coma}}, \tfrac{net_{opena}}{io_{opena}}\}\varphi_B|S_B) = \nu\underline{n}_B.(\mathcal{F}_{COM}\{\tfrac{net'_{coma}}{io_{coma}}, \tfrac{net'_{opena}}{io_{opena}}\}|S_B) \tag{2.7}$$

for suitable names $net'_{coma}, net'_{opena}, net'_{comb}, net'_{openb}$. The equalities use the fact that $\mathcal{F}_{COM}$ does not contain any NET-names.

Let

$$\begin{aligned}E := \nu io_{coma} io_{comb} io_{opena} io_{openb}.\\ \Big(\Big(\nu r.\Big(\overline{io_{coma}}\langle r\rangle | io_{comb}().(\overline{io_{opena}}\langle\rangle | io_{openb}(x).\text{if } x = r \text{ then } \bar{c}\langle\rangle)\Big)\Big) | \Box\Big)\end{aligned}$$

where c is a fresh name. Intuitively, this context commits to a fresh nonce r, waits until the commit succeeds, then opens the commitment and checks whether the unveiled value is indeed r. For a "good" commitment scheme, this should always be the case. Indeed: By definition of $\mathcal{F}_{COM}$ (and using that $\underline{n}_0$ does not contain IO-names), we have that $E[\nu\underline{n}_0.(\mathcal{F}_{COM}|S_0)] \to^* \downarrow_c$. By (Equation 2.5) we have $E[\nu\underline{net}.(A|B)] \approx E[\nu\underline{n}_0.(\mathcal{F}_{COM}|S_0)]$ and thus $E[\nu\underline{net}.(A|B)] \to^* \downarrow_c$.

We now use (Equation 2.6) and (Equation 2.7) to transform $E[\nu\underline{net}.(A|B)]$ into a process that does not use the commitment protocol $A|B$ any more, but instead uses *two* instances of $\mathcal{F}_{COM}$:

$$\begin{aligned}&E[\nu\underline{net}.(A|B)] \\ \overset{(2.6,2.7)}{\approx}&E[\nu\underline{net}.(\nu\underline{n}_A.(\mathcal{F}_{COM}\{\tfrac{net'_{comb}}{io_{comb}}, \tfrac{net'_{openb}}{io_{openb}}\}|S_A)|\nu\underline{n}_B.(\mathcal{F}_{COM}\{\tfrac{net'_{coma}}{io_{coma}}, \tfrac{net'_{opena}}{io_{opena}}\}|S_B))]\end{aligned} \tag{2.8}$$

By moving all restrictions up (and potentially renaming names to avoid clashes of bound variables), we get:

$$E[\nu\underline{net}.(A|B)] \approx \nu\underline{net}'.E[\mathcal{F}_{COM}\{\tfrac{net''_{comb}}{io_{comb}}, \tfrac{net''_{openb}}{io_{openb}}\}|\mathcal{F}_{COM}\{\tfrac{net''_{coma}}{io_{coma}}, \tfrac{net''_{opena}}{io_{opena}}\}|S_{AB}] =: P$$

Here $\underline{net}'$ is the list of all names that were moved up. net''_{coma} etc are potentially renamed names, and $S_{AB} := S_A|S_B$ potentially up to renamings. Note that S_{AB} does not contain IO-names.

We now use several application of Lemma 5 to simplify P. Each of the following observational equivalences corresponds to one application of Lemma 5.

$$\begin{aligned}
P \equiv{} & \nu \underline{net}\, io_{coma} io_{comb} io_{opena} io_{openb} r. \\
& \quad \overline{io_{coma}}\langle r\rangle \mid io_{comb}().(\overline{io_{opena}}\langle\rangle \mid io_{openb}(x).\text{if } x = r \text{ then } \bar{c}\langle\rangle) \\
& \quad \mid \mathcal{F}_{COM}\{\tfrac{net''_{coma}}{io_{coma}}, \tfrac{net''_{opena}}{io_{opena}}\} \mid \mathcal{F}_{COM}\{\tfrac{net''_{comb}}{io_{comb}}, \tfrac{net''_{openb}}{io_{openb}}\} \mid S_{AB} \\
={} & \nu \underline{net}\, io_{coma} io_{comb} io_{opena} io_{openb} r. \\
& \quad \overline{io_{coma}}\langle r\rangle \mid io_{comb}().(\overline{io_{opena}}\langle\rangle \mid io_{openb}(x).\text{if } x = r \text{ then } \bar{c}\langle\rangle) \\
& \quad \mid \mathcal{F}_{COM}\{\tfrac{net''_{coma}}{io_{coma}}, \tfrac{net''_{opena}}{io_{opena}}\} \mid \colorbox{lightgray}{$io_{coma}(x_m).(\overline{net''_{comb}}\langle\rangle \mid io_{opena}().\overline{net''_{openb}}\langle x_m\rangle)$} \mid S_{AB} \\
\overset{\text{(i)}}{\approx}{} & \nu \underline{net}\, \colorbox{lightgray}{}\, io_{comb} io_{opena} io_{openb} r. \\
& \quad \colorbox{lightgray}{$\overline{net''_{comb}}\langle\rangle \mid io_{opena}().\overline{net''_{openb}}\langle r\rangle$} \mid io_{comb}().(\overline{io_{opena}}\langle\rangle \mid io_{openb}(x).\text{if } x = r \text{ then } \bar{c}\langle\rangle) \\
& \quad \mid \mathcal{F}_{COM}\{\tfrac{net''_{coma}}{io_{coma}}, \tfrac{net''_{opena}}{io_{opena}}\}\, \colorbox{lightgray}{} \mid S_{AB} \\
\overset{\text{(ii)}}{\approx}{} & \nu \underline{net}\, io_{comb}\, \colorbox{lightgray}{}\, io_{openb} r. \\
& \quad \overline{net''_{comb}}\langle\rangle\, \colorbox{lightgray}{} \mid io_{comb}().(\colorbox{lightgray}{$\overline{net''_{openb}}\langle r\rangle$} \mid io_{openb}(x).\text{if } x = r \text{ then } \bar{c}\langle\rangle) \\
& \quad \mid \mathcal{F}_{COM}\{\tfrac{net''_{coma}}{io_{coma}}, \tfrac{net''_{opena}}{io_{opena}}\} \mid S_{AB} \\
={} & \nu \underline{net}\, io_{comb} io_{openb} r. \\
& \quad \overline{net''_{comb}}\langle\rangle \mid io_{comb}().(\overline{net''_{openb}}\langle r\rangle \mid io_{openb}(x).\text{if } x = r \text{ then } \bar{c}\langle\rangle) \\
& \quad \mid \colorbox{lightgray}{$net''_{coma}(x_m).(\overline{io_{comb}}\langle\rangle \mid net''_{opena}().\overline{io_{openb}}\langle x_m\rangle)$} \mid S_{AB} \\
\overset{\text{(iii)}}{\approx}{} & \nu \underline{net}\, \colorbox{lightgray}{}\, io_{openb} r.\ \overline{net''_{comb}}\langle\rangle\, \colorbox{lightgray}{} \\
& \quad \mid net''_{coma}(x_m).(\colorbox{lightgray}{$\overline{net''_{openb}}\langle r\rangle \mid io_{openb}(x).\text{if } x = r \text{ then } \bar{c}\langle\rangle$} \\
& \quad \mid net''_{opena}().\overline{io_{openb}}\langle x_m\rangle) \mid S_{AB} \\
\overset{\text{(iv)}}{\approx}{} & \nu \underline{net}\, \colorbox{lightgray}{}\, r.\ \overline{net''_{comb}}\langle\rangle \\
& \quad \mid net''_{coma}(x_m).(\overline{net''_{openb}}\langle r\rangle\, \colorbox{lightgray}{} \mid net''_{opena}().\colorbox{lightgray}{$\text{if } x_m = r \text{ then } \bar{c}\langle\rangle$}) \mid S_{AB}
\end{aligned}$$

Here (i) uses Lemma 5 with $n := io_{coma}$, $t := r$, and $x := x_m$.

And (ii) uses Lemma 5 with $n := io_{opena}$.

And (iii) uses Lemma 5 with $n := io_{comb}$.

And (iv) uses Lemma 5 with $n := io_{openb}$, $t := x_m$, and $x := x$ (and Lemma 4 (ii) to move the νio_{openb} below the $net''_{coma}(x_m)$ first, and Lemma 1, so that we can apply Lemma 5 to a subprocess instead of the whole process.)

Thus we have

$$\begin{aligned}
& E[\nu \underline{net}.(A|B)] \approx P \approx \\
& \nu \underline{net}\, r.\ \overline{net''_{comb}}\langle\rangle \mid net''_{coma}(x_m).(\overline{net''_{openb}}\langle r\rangle \mid net''_{opena}().\text{if } x_m = r \text{ then } \bar{c}\langle\rangle) \mid S_{AB} =: P_2
\end{aligned}$$

Note that in P_2, x_m is received before the fresh nonce r is revealed. Thus we expect that the comparison $x_m = r$ will always fail. Indeed:

$$\begin{aligned}
P_2 \overset{(*)}{\equiv}{} & \nu \underline{net}\, \colorbox{lightgray}{}.\overline{net''_{comb}}\langle\rangle \mid net''_{coma}(x_m).\colorbox{lightgray}{$\nu r.$}(\overline{net''_{openb}}\langle r\rangle \\
& \quad \mid net''_{opena}().\text{if } x_m = r \text{ then } \bar{c}\langle\rangle) \mid S_{AB} \\
\overset{(**)}{\approx}{} & \nu \underline{net}.\overline{net''_{comb}}\langle\rangle \mid net''_{coma}(x_m).\nu r.(\overline{net''_{openb}}\langle r\rangle \mid net''_{opena}().\colorbox{lightgray}{0}) \mid S_{AB} =: P_3
\end{aligned}$$

Here $(*)$ uses Lemma 4 (ii) with $x := x_m$ to move the restriction νr down, and $(**)$ uses Lemma 10 to replace the if-statement by its else-branch (which is 0).

Thus we have that $E[\nu\underline{net}.(A|B)] \approx P_2 \approx P_3$.

Furthermore, we showed above that $E[\nu\underline{net}.(A|B)] \to^*\downarrow_c$. But since c does not occur in P_3 (we chose it as a fresh name, thus it also does not occur in S_{AB}), we have that $P_3 \to^*\downarrow_c$ cannot hold. This is a contradiction to the observational equivalence $E[\nu\underline{net}.(A|B)] \approx P_3$. Thus our assumption was wrong that processes A, B and NET-names $\underline{net}$ as in the statement of the lemma exist. □

However, Lemma 45 does not exclude that an approach similar to the virtual primitives approach might work: We first construct a UC secure commitment protocol (again, commitments are just one example), build a complex protocol from it using the composition theorem, and then show that security of the complex protocol implies (non-UC) security of a modification that does not use the CRS. It is likely that this works as the CRS returned by the CRS functionality is just a fresh public name, so instead of the CRS we should be able to just use some fresh (non-restricted) name a.

There is one subtlety, though: When composing the commitment protocol P, we end up with a complex protocol $C[P]$ that may use multiple instances of $\mathcal{F}_{CRS}$. In particular, if $C[P]$ contains $!!P$, then $C[P]$ will contain an unbounded number of $\mathcal{F}_{CRS}$-instances. So we cannot replace $\mathcal{F}_{CRS}$ just by a single name, we will need a way to generate an arbitrary number of fresh values. The obvious way for this is to use something like $\mathit{hash}(a, \mathit{sid})$ instead of the CRS that we get from the $\mathcal{F}_{CRS}$-instance with session-id sid (here a is a fresh name).

A lemma roughly like the following conjecture should therefore lead to a method for removing the CRS from a protocol that was produced by UC composition:

Conjecture 2.8.1. *Let hash be a free constructor (i.e., not occurring in any equations or rewrite rules in the symbolic models). Let P be a process. Let E_1, E_2 be contexts. Assume that hash does not occur in E_1, E_2, P. Let $a \notin \mathit{fn}(E_1, E_2, P) \cup \mathit{bn}(E_1, E_2, P)$.*

(i) Let P' result from P by replacing all subterms "$net_{crsa}(x).Q$" by "let $x = a$ in Q". Then $E_1[\nu net_{crsa}.(P|\mathcal{F}_{CRS})] \approxeq E_2[\nu net_{crsa}.(P|\mathcal{F}_{CRS})]$ implies $E_1[\nu net_{crsa}.(P')] \approxeq E_2[\nu net_{crsa}.(P')]$.

(ii) Let P' result from P by replacing all subterms "$(M_{sid}, net_{crsa})(x).Q$" by "let $x = \mathit{hash}(a, M_{sid})$ in Q". Then $E_1[\nu net_{crsa}.(P|!!\mathcal{F}_{CRS})] \approxeq E_2[\nu net_{crsa}.(P|!!\mathcal{F}_{CRS})]$ implies $E_1[\nu net_{crsa}.(P')] \approxeq E_2[\nu net_{crsa}.(P')]$.

Proving (i) is probably considerably simpler than proving (ii). An alternative to proving (ii) could be to make sure that $C[P]$ does not contain $\mathcal{F}_{CRS}$ under a !!. This could be achieved if we design a commitment protocol P that does not implement $\mathcal{F}_{COM}$, but $!!\mathcal{F}_{COM}$ (compare with Section 2.7.3). Then a single copy of P would be sufficient in $C[P]$.

We leave further exploration of approaches to get rid of the CRS to future research.

2.9 Limits for composition and property preservation

In this section, we show that the restrictions of the composition theorem are necessary. More precisely, we show that if $P \leq Q$, then not necessarily $!P \leq\ !Q$

```
fun empty/0.

free net2, net3.

let Q = new n; out(io1,n) |
    (in(io2,x); if x=n then out(net2,empty)) |
    (in(io3,x); if x=n then out(net3,empty)).

process new io1; new io2; new io3; in(io1,x1); in(io1,x2);
                          out(io2,x1) | out(io3,choice[x1,x2]) | !Q
```

Figure 2.16: Proverif code for showing $E_1[Q] \approx E_2[Q]$ in Lemma 46 (prop-pres-bang1.pv, see [29]).

or $io(x).P \leq io(x).Q$ or $\overline{io}\langle t\rangle.P \leq \overline{io}\langle t\rangle.Q$ or $\nu net.P \leq \nu net.Q$ or $P|R \leq Q|R$ (for R that has NET-names in common with P, Q). We show that this is not just a limitation of the composition theorem, we show that similar limitations also apply to property preservation. More precisely, property preservation $P \leq Q, E_1[Q] \approx E_2[Q] \implies E_1[P] \approx E_2[P]$ does not necessarily hold if E_1, E_2 contain a bang (!) over their hole, or an input/output over their hole, or an if/let over their hole, or a different number of !!'s over their respective holes, or restrict NET-names over their holes, or use NET-names.

Example 2.9.1.

$$P := \nu n\, m.\ \overline{io_1}\langle n\rangle \mid io_2(x).\mathit{if}\ x = n\ \mathit{then}\ \overline{net_2}\langle m\rangle \mid io_3(x).\mathit{if}\ x = n\ \mathit{then}\ \overline{net_3}\langle m\rangle$$

$$Q := \nu n\ \ .\ \overline{io_1}\langle n\rangle \mid io_2(x).\mathit{if}\ x = n\ \mathit{then}\ \overline{net_2}\langle \mathit{empty}\rangle \mid io_3(x).\mathit{if}\ x = n\ \mathit{then}\ \overline{net_3}\langle \mathit{empty}\rangle$$

$$E_1 := \nu io_1\, io_2\, io_3.\ io_1(x_1).io_1(x_2).(\overline{io_2}\langle x_1\rangle \mid \overline{io_3}\langle x_1\rangle) \mid !\Box$$

$$E_2 := \nu io_1\, io_2\, io_3.\ io_1(x_1).io_1(x_2).(\overline{io_2}\langle x_1\rangle \mid \overline{io_3}\langle x_2\rangle) \mid !\Box$$

Lemma 46. *Using the notation from Example 2.9.1, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Proof. We show $P \leq Q$: We have $P \approx \nu net_2' net_3'.(Q\{\frac{net_2'}{net_2}, \frac{net_3'}{net_3}\}|S)$ for $S := \nu m.(net_2'(x).\overline{net_2}\langle m\rangle | net_3'(x).\overline{net_3}\langle m\rangle)$ by two invocations of Lemma 5 (first with $n := net_2'$, $x := x$, and $t := \mathit{empty}$, second with $n := net_3'$, $x := x$, and $t := \mathit{empty}$). Hence $P \leq Q$.

The claim $E_1[Q] \approx E_2[Q]$ is shown using Proverif. The Proverif code is given in Figure 2.16

We now show $E_1[P] \not\approx E_2[P]$. Let $D := net_2(y_1).net_3(y_2).\text{if } y_1 = y_2 \text{ then } \overline{c}\langle \mathit{empty}\rangle$. Then $D \mid E_1[P] \rightarrow^* D \mid \cdots \mid \nu m.(\overline{net_2}\langle m\rangle \mid \overline{net_3}\langle m\rangle) \rightarrow^* \nu m.(\cdots \mid \text{if } m = m \text{ then } \overline{c}\langle\rangle) \rightarrow^* \downarrow_c$. Using Proverif, we show that $D \mid E_2[P] \rightarrow^* \downarrow_c$ does not hold (for any context D not containing c). The Proverif code is given in Figure 2.17. $E_1[P] \approx E_2[P]$ would imply $D \mid E_1[P] \approx D \mid E_2[P]$ which together with $D \mid E_1[P] \rightarrow^* \downarrow_c$ would imply the wrong fact $D \mid E_2[P] \rightarrow^* \downarrow_c$. Thus $E_1[P] \not\approx E_2[P]$. □

Lemma 47. *Using the notation from Example 2.9.1, we have $P \leq Q$ but not $!P \leq !Q$.*

```
fun empty/0.

free net2, net3.
private free c.

query mess:c,c.

let P = new n; new m; out(io1,n) |
    (   in(io2,x); if x=n then out(net2,m))
     | (in(io3,x); if x=n then out(net3,m)).

let E2P = new io1; new io2; new io3; in(io1,x1); in(io1,x2);
                                     out(io2,x1) | out(io3,x2) | !P.

let D = in(net2,y1); in(net3,y2); if y1=y2 then out(c,empty).

process D | E2P
```

Figure 2.17: Proverif code for showing that $D|E_2[P] \rightarrow^* \downarrow_c$ does not hold in the proof of Lemma 46 (`prop-pres-bang2.pv`, see [29]).

Proof. From Lemma 46 we have $P \leq Q$ and $E_1[Q] \approx E_2[Q]$. Assume $!P \leq !Q$. We can write $E_1 = E_1'[!\Box]$ and $E_2 = E_2'[!\Box]$ for NET-free evaluation contexts E_1, E_2. Then $E_1'[!Q] = E_1[Q] \approx E_2[Q] = E_2'[!Q]$ and thus by Theorem 2, we have $E_1[P] = E_1'[!P] \approx E_2'[!P] = E_2[P]$. This is a contradiction to Lemma 46. Thus the assumption $!P \leq !Q$ was wrong. □

Example 2.9.2.

$$\begin{aligned} P &:= \overline{net}\langle empty\rangle \\ Q &:= 0 \\ E_1 &:= \nu io.\ (io().\Box \mid \overline{io}\langle empty\rangle) \\ E_2 &:= \nu io.\ (io().\Box) \end{aligned}$$

Lemma 48. *Using the notation from Example 2.9.2, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Proof. Obviously, $P \approx Q|S$ with $S := \overline{net}\langle empty\rangle$. Hence $P \leq S$.

We show $E_1[Q] \approx E_2[Q]$: We have $E_1[Q] = \nu io.\ (io().0 \mid \overline{io}\langle empty\rangle) \approx 0$ by Lemma 5 with $n := io$ and $C := \Box$. And $E_2[Q] = \nu io.io().0 \approx 0$ by Lemma 5 with $n := io$ and $C := 0$. Hence $E_1[Q] \approx E_2[Q]$.

We show $E_1[P] \not\approx E_2[P]$: We have $E_1[P] \rightarrow^* \nu io.\overline{net}\langle empty\rangle \downarrow_{net}$. But $E_2[P] \not\downarrow_{net}$, and $E_2[P]$ does not reduce. Thus there is no successor of $E_2[P]$ that emits on *net*. This contradicts $E_1[P] \approx E_2[P]$ by definition of observational equivalence. □

Lemma 49. *Using the notation from Example 2.9.2, we have $P \leq Q$ but not $io().P \leq io().Q$.*

Proof. From Lemma 48 we have $P \leq Q$ and $E_1[Q] \approx E_2[Q]$. Assume $io().P \leq io().Q$. We can write $E_1 = E_1'[io().\Box]$ and $E_2 = E_2'[io().\Box]$ for NET-free evaluation contexts E_1, E_2. Then $E_1'[io().Q] = E_1[Q] \approx E_2[Q] = E_2'[io().Q]$ and thus by Theorem 2, we have $E_1[P] = E_1'[io().P] \approx E_2'[io().P] = E_2[P]$. This is a contradiction to Lemma 48. Thus the assumption $io().P \leq io().Q$ was wrong. □

Example 2.9.3. *Let P, Q be as in Example 2.9.2.*

$$E_1 := \nu io.\ (\overline{io}\langle \mathit{empty}\rangle.\Box \mid io())$$
$$E_2 := \nu io.\ (\overline{io}\langle \mathit{empty}\rangle.\Box)$$

Lemma 50. *Using the notation from Example 2.9.3, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Lemma 51. *Using the notation from Example 2.9.3, we have $P \leq Q$ but not $\overline{io}\langle \mathit{empty}\rangle.P \leq \overline{io}\langle \mathit{empty}\rangle.Q$.*

The proofs of Lemmas 50 and 51 are identical to those of Lemmas 50 and 51, except that $io()$ and $\overline{io}\langle \mathit{empty}\rangle$ are exchanged.

Example 2.9.4. *Let P, Q be as in Example 2.9.2.*

$$E_1 := \mathit{if\ true\ then}\ \Box$$
$$E_2 := \mathit{if\ false\ then}\ \Box$$

Here true is an equality $t = t$ for an arbitrary closed t (e.g., $\mathit{empty} = \mathit{empty}$), and false is an equality $t = t'$ for arbitrary closed t, t' with $t \neq_{\mathrm{E}} t'$ (e.g., $\mathit{empty} = (\mathit{empty}, \mathit{empty})$).

Remember that if $x = y$ is syntactic sugar for let $z = \mathit{equals}(x, y)$. So this example is a counterexample for let-statements.

Lemma 52. *Using the notation from Example 2.9.4, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Proof. $P \leq Q$ was already shown in Lemma 48. By Lemma 4 (v) we have that $E_1[P] \approx P$ and $E_1[Q] \approx Q = 0$ and by Lemma 4 (v) we have that $E_1[P] \approx 0$ and $E_2[Q] \approx 0$. Obviously, $P \not\approx 0$. $E_1[P] \not\approx E_2[P]$, but $E_1[Q] \approx E_2[Q]$. □

Example 2.9.5. *Let P, Q be as in Example 2.9.2.*

$$E_1 := !!\Box$$
$$E_2 := \Box$$

Lemma 53. *Using the notation from Example 2.9.5, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Proof. $P \leq Q$ was already shown in Lemma 48. Let $t \in \mathit{SID}$ be arbitrary. We have $E_1[P] \approx \prod_{x \in \mathit{SID}} \overline{(x, \mathit{net})}\langle \mathit{empty}\rangle \to^* \downarrow_{(t,\mathit{net})}$. But no successor of $E_2[P] = \overline{\mathit{net}}\langle \mathit{empty}\rangle$ emits on $(t, \mathit{net}) \neq_{\mathrm{E}} \mathit{net}$. Thus $E_1[P] \not\approx E_2[P]$.

It is easy to see that $0 \approx \prod_{x \in \mathit{SID}} 0$ (by showing that $\mathcal{R} := \{(R, R | \prod_{x \in \mathit{SID} \setminus S} 0)\}$ up to structural equivalence is a bisimulation). Thus

$$E_1[Q] = !!0 \approx \prod_{x \in \mathit{SID}} 0 \approx 0 = E_2[Q].$$

□

Example 2.9.6.

$$
\begin{aligned}
P &:= \mathit{net}().\mathit{io}().\overline{\mathit{io'}}\langle\rangle\\
Q &:= \mathit{net'}().\mathit{io}().\overline{\mathit{io'}}\langle\rangle\\
E_1 &:= \nu \mathit{io}.(\overline{\mathit{io}}\langle\rangle \mid \nu \mathit{net'}.\square)\\
E_2 &:= \nu \mathit{io}.(\nu \mathit{net'}.\square)
\end{aligned}
$$

Lemma 54. *Using the notation from Example 2.9.6, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Proof. $P \leq Q$ holds with simulator $S := 0$, $\varphi := (\mathit{net'} \mapsto \mathit{net})$, $\underline{n} := \varnothing$.

It is easy to see that $\nu \mathit{net'}.Q \approx 0$. Hence $E_1[Q] \approx \nu \mathit{io}.\overline{\mathit{io}}\langle\rangle$ and $E_2[Q] \approx \nu \mathit{io}.0$. Thus $E_1[Q] \approx E_2[Q]$.

But $E_1[P] \rightarrow^* \downarrow_{\mathit{io'}}$ and $E_2[P] \not\rightarrow^* \downarrow_{\mathit{io'}}$. Hence $E_1[P] \not\approx E_2[P]$. □

Lemma 55. *Using the notation from Example 2.9.1, we have $P \leq Q$ but not $\nu \mathit{net'}.P \leq \nu \mathit{net'}.Q$.*

Proof. From Lemma 54 we have $P \leq Q$ and $E_1[Q] \approx E_2[Q]$. Assume $\nu \mathit{net'}.P \leq \nu \mathit{net'}.Q$. We can write $E_1 = E_1'[\nu \mathit{net'}.\square]$ and $E_2 = E_2'[\nu \mathit{net'}.\square]$ for NET-free evaluation contexts E_1, E_2. Then $E_1'[\nu \mathit{net'}.Q] = E_1[Q] \approx E_2[Q] = E_2'[\nu \mathit{net'}.Q]$ and thus by Theorem 2, we have $E_1[P] = E_1'[\nu \mathit{net'}.P] \approx E_2'[\nu \mathit{net'}.P] = E_2[P]$. This is a contradiction to Lemma 54. Thus the assumption $\nu \mathit{net'}.P \leq \nu \mathit{net'}.Q$ was wrong. □

Example 2.9.7.

$$
\begin{aligned}
P &:= \mathit{io}().\overline{\mathit{net}}\langle\rangle\\
Q &:= \mathit{io}().\overline{\mathit{net'}}\langle\rangle\\
E_1 &:= \nu \mathit{io}.(\overline{\mathit{io}}\langle\rangle \mid \square \mid !\overline{\mathit{net'}}\langle\rangle)\\
E_2 &:= (\nu \mathit{io}.\square \mid !\overline{\mathit{net'}}\langle\rangle)
\end{aligned}
$$

Lemma 56. *Using the notation from Example 2.9.6, we have $P \leq Q$, and $E_1[Q] \approx E_2[Q]$, but $E_1[P] \not\approx E_2[P]$.*

Proof. $P \leq Q$ holds with simulator $S := 0$, $\varphi := (\mathit{net'} \mapsto \mathit{net})$, $\underline{n} := \varnothing$.

By Lemma 5, we have $E_1[Q] \approx \overline{\mathit{net'}}\langle\rangle \mid !\overline{\mathit{net'}}\langle\rangle$. And by Lemma 4 (viii), $\overline{\mathit{net'}}\langle\rangle \mid !\overline{\mathit{net'}}\langle\rangle \approx !\overline{\mathit{net'}}\langle\rangle$. Finally $E_2[Q] \approx 0 \mid !\overline{\mathit{net'}}\langle\rangle$. Hence $E_1[Q] \approx E_2[Q]$.

But $E_1[P] \rightarrow^* \downarrow_{\mathit{net}}$ and $E_2[P] \not\rightarrow^* \downarrow_{\mathit{net}}$. Hence $E_1[P] \not\approx E_2[P]$. □

Lemma 57. *Using the notation from Example 2.9.1, we have $P \leq Q$ but not $P \mid !\overline{\mathit{net'}}\langle\rangle \leq Q \mid !\overline{\mathit{net'}}\langle\rangle$.*

Proof. From Lemma 56 we have $P \leq Q$ and $E_1[Q] \approx E_2[Q]$. Assume $P \mid !\overline{\mathit{net'}}\langle\rangle \leq Q \mid !\overline{\mathit{net'}}\langle\rangle$. We can write $E_1 = E_1'[\square \mid !\overline{\mathit{net'}}\langle\rangle]$ and $E_2 = E_2'[\square \mid !\overline{\mathit{net'}}\langle\rangle]$ for NET-free evaluation contexts E_1, E_2. Then $E_1'[Q \mid !\overline{\mathit{net'}}\langle\rangle] = E_1[Q] \approx E_2[Q] = E_2'[Q \mid !\overline{\mathit{net'}}\langle\rangle]$ and thus by Theorem 2, we have $E_1[P] = E_1'[P \mid !\overline{\mathit{net'}}\langle\rangle] \approx E_2'[P \mid !\overline{\mathit{net'}}\langle\rangle] = E_2[P]$. This is a contradiction to Lemma 56. Thus the assumption $P \mid !\overline{\mathit{net'}}\langle\rangle \leq Q \mid !\overline{\mathit{net'}}\langle\rangle$ was wrong. □

3. Composable Computational Soundness

In this chapter we present the results of [25] with minor changes. Section 3.2.1 is not contained in [25] and discusses differences between the symbolic models in Chapter 2 and this chapter.

The Boundaries of Deduction Soundness. . .

Compositionality for deduction soundness as introduced in [48] is limited, and the authors present rather compelling evidence that the notion may not compose primitives other than encryption. The problem is that deduction soundness does not seem to preclude implementations that leak partial information about their inputs. In turn, this leak of information may impact the security of other primitives that one may want to include later.

More concretely, assume that one has established soundness of a deduction system that covers hash, but for an implementation of the hash function that reveals half of its input: $h(m_1 \| m_2) = m_1 \| g(m_2)$ where g is a standard hash function. If g is a “good" hash function then so is h. Now consider a signature scheme which duplicates signatures: $\mathit{sig}(\mathit{sk}, m) = \mathit{sig}'(\mathit{sk}, m) \| \mathit{sig}'(\mathit{sk}, m)$ where sig' is some standard signature scheme. It is easy to see that if $\mathit{sig}'(\mathit{sk}, m)$ is a secure signature scheme, then so is $\mathit{sig}(\mathit{sk}, m)$. Yet, given $h(\mathit{sig}(\mathit{sk}, m))$ an adversary can easily compute $\mathit{sig}(\mathit{sk}, m)$ without breaking the signature scheme nor the hash: the hash function leaks sufficient information to be able to recover the underlying signature.

. . . Revisited.

In this chapter, we prove that to any deduction sound implementation of a set of primitives, one can add signatures, as long as the implementation for the signature satisfies a standard notion of security. This theorem refutes the counterexample above and provides evidence that deduction soundness is a more powerful (and demanding) security notion than previously understood. In particular, a corollary of the theorem is that there are no deduction sound abstractions for implementations that are “too leaky” (as the hash function from the counterexample).

The new level of understanding facilitates further compositionality proofs for deduction soundness: to any deduction sound system one can add any of the (remaining) standard cryptographic primitives: symmetric encryption, message authentication codes, and hash functions while preserving deduction soundness. The theorems hold under standard security assumptions for the implementation of encryption and MACs and require random oracles for adding hash functions.

As a consequence, we obtain the first soundness result that encompasses all standard primitives: symmetric and asymmetric encryption, signatures, MACs, and hashes. In addition, our composition results allow for a settings where multiple schemes (that implement the same primitive) are used simultaneously, provided that each implementation fulfills our assumptions. Moreover, composition provides a stronger result: whenever deduction soundness is shown for some particular primitive, our result ensures that all standard primitives can be added for free, without any further proof.

Limits of Our Result.

Our compositionality results hold under several restrictions most of which are quite common in soundness proofs, e.g. adversaries can corrupt keys only statically. Less standard is that we demand for secret keys to be used only for the cryptographic task for which they are intended. Quite reasonable most of the time, the restriction does not allow, for example, for the adversary to see encryptions of symmetric keys under public keys. The restriction is related to the signature-hash counterexample. If f is a primitive with a deduction sound system that leaks some information about its input and *enc* is a secure encryption function it is not clear that $(f(k), \mathit{enc}(k, m))$ hides m. Unfortunately, the technique that we used to bypass the signature-hash counterexample does not seem to apply here. At a high level, the difficulty is that in a potential reduction to the security of the encryption scheme, we are not be able to simulate $f(k)$ consistently.

One way to relax the restriction is to employ encryption schemes that are secure even when some (or even most) of the encryption key leaks [55, 68]. Current instantiations for such schemes are highly inefficient and we prefer the following alternative solution which, essentially, allows for other uses of symmetric keys, as long as these uses do not reveal information about the keys. In a bit more detail, we say that a function is *forgetful* for some argument if the function hides (computationally) all of the information about that input. The notion is a generalization for the security of encryption schemes: these can be regarded as forgetful with respect to their plaintext.

More Freedom for Keys.

We then show that a forgetful deduction sound implementation can be extended with symmetric encryption under more relaxed restrictions: soundness is preserved if encryption keys are used for encryption, or appear only in forgetful positions of other functions from the implementation we are extending.

Finally, we show that, in addition to soundness, forgetfulness is preserved as well. Hence we can flexibly and add several layers of asymmetric/symmetric key encryption such that the keys of each layer may appear in any forgetful position of underlying layers. We feel that this allows us to capture almost every hierarchical encryption mechanism in practical protocols.

3.1 Preliminaries

Throughout this chapter, η denotes the *security parameter*. A function $f : \mathbb{N} \to \mathbb{R}$ is *negligible* if it vanishes faster than the inverse of any polynomial (i.e., if $\forall c \in \mathbb{R}\ \exists n_0 \in \mathbb{N}$ s.t. $\forall n \in \mathbb{N}\ |f(n)| < 1/n^c$). For a finite set R, we denote by $r \leftarrow R$ the process of sampling r uniformly from R. For a probabilistic algorithm A, we denote with $y := A(x; r)$ the process of running A on input x and with randomness r, and assigning y the result. If R denotes the randomness space of A, we write $y \leftarrow A(x)$ for $y := A(x; r)$ with $r \leftarrow R$. If A's running time is polynomial in η, then A is called probabilistic polynomial-time (PPT).

3.2 The symbolic model

This section introduces the notion for symbolic models used throughout this chapter. It diverges in some points from the definition of symbolic models for the applied pi calculus used in Chapter 2. Naturally, the general idea of a symbolic model to use cryptography in an abstract way and based on formal deduction rules remains unchanged. At the end of this section we motivate and explain the differences between the two notions in detail and show how they can be reconciled (Section 3.2.1).

Analogously to Section 2.1 our abstract models for the symbolic world consist of term algebras defined on a typed first-order signature.

Specifically we have a set of *data types* $\mathcal{T}$ with a subtype relation ($\preccurlyeq$) which we require to be a preorder. We assume that $\mathcal{T}$ always contains a base type $\top$ such that every other type $\tau \in \mathcal{T}$ is a subtype of $\top$ ($\tau \preccurlyeq \top$).

The *signature* Σ is a set of *function symbols* together with arities of the form $\mathrm{ar}(f) = \tau_1 \times \ldots \times \tau_n \to \tau$, $n \geq 0$ for $\tau_i, \tau \in \mathcal{T}$. We refer to τ as the type of f and require $\tau \neq \top$ for all f except for garbage of basetype $g_\top$. Function symbols with $n = 0$ arguments are called *constants*. We distinguish *deterministic* function symbols, e.g., for pairs, and *randomized* function symbols, e.g., for encryption.

For all symbolic models we fix an infinite set of typed *variables* $\{x, y \ldots\} \in \mathcal{V}$ and an infinite set of *labels* $\mathsf{labels} = \mathsf{labelsH} \cup \mathsf{labelsA}$ for infinite, disjoint sets of *honest labels* ($\mathsf{labelsH}$) and *adversarial labels* ($\mathsf{labelsA}$). Since labels are used to specify randomness, distinguishing honest and adversarial labels (randomness) is important.

The set of *terms of type* τ is defined inductively by

$$\begin{array}{lll} t ::= & & \text{term of type } \tau \\ \quad | & x & \text{variable } x \text{ of type } \tau \\ \quad | & f(t_1, \ldots, t_n) & \text{application of deterministic } f \in \Sigma \\ \quad | & f^l(t_1, \ldots, t_n) & \text{application of randomized } f \in \Sigma \end{array}$$

where for the last two cases, we further require that each t_i is a term of some type τ_i' with $\tau_i' \preccurlyeq \tau_i$ for $\mathrm{ar}(f) = \tau_1 \times \ldots \times \tau_n \to \tau$ and for the last case that $l \in \mathsf{labels}$. The set of terms is denoted by $\mathsf{Terms}(\Sigma, \mathcal{T}, \preccurlyeq)$ and is the union over all sets of terms of type τ for all $\tau \in \mathcal{T}$. For ease of notation we often write $\mathsf{Terms}(\Sigma)$ for the same set of terms, and refer to general terms as $t = f^l(t_1, \ldots, t_n)$ even if f could be a deterministic function symbol which doesn't carry a label.

Intuitively, for nonces, we use randomized constants. For example, assume that $n \in \Sigma$ is a constant. Then usual nonces can be represented by $n^{r_1}, n^{r_2}, \ldots$ where $r_1, r_2 \in \mathsf{labels}$ are labels. Labels in $\mathsf{labelsH}$ will be used when the function has been applied by an honest agent (thus the randomness has been honestly generated) whereas labels in $\mathsf{labelsA}$ will be used when the randomness has been generated by the adversary. Often when the label for a function symbol is clear from the context (e.g. when there is only one label that suits a particular function symbol) we may omit this label.

We require Σ to contain randomized constants g_τ of type τ for any $\tau \in \mathcal{T}$ that will be used for representing garbage of type τ. Garbage will typically be the terms associated to bitstrings produced by the adversary which cannot be parsed as a meaningful term (yet). If garbage can at some point be parsed as the application of a deterministic function symbol, the label is dropped.

Substitutions are written $\sigma = \{x_1 = t_1, \ldots, x_n = t_n\}$ with *domain* $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$. We only consider *well-typed* substitutions, that is substitutions $\sigma = \{x_1 = t_1, \ldots, x_n = t_n\}$ for which t_i is of a subtype of x_i. The application of a substitution σ to a term t is written $\sigma(t) = t\sigma$.

Function symbols in Σ are intended to model cryptographic primitives, including generation of random data like e.g. nonces or keys. Identities will typically be represented by constants (deterministic function symbols without arguments). The symbolic model is equipped with a *deduction relation* $\vdash \subseteq 2^{\mathsf{Terms}} \times \mathsf{Terms}$ that models the information available to a symbolic adversary. $T \vdash t$ means that a formal adversary can build t out of T, where t is a term and T a set of terms. We say that t is *deducible* from T. Deduction relations are typically defined through deduction systems.

Definition 39. *A* deduction system $\mathcal{D}$ *is a set of rules* $\frac{t_1 \quad \cdots \quad t_n}{t}$ *such that* $t_1, \ldots, t_n$, $t \in \mathsf{Terms}(\Sigma, \mathcal{T}, \preccurlyeq)$. *The deduction relation* $\vdash_{\mathcal{D}} \subseteq 2^{\mathsf{Terms}} \times \mathsf{Terms}$ *associated to* $\mathcal{D}$ *is the smallest relation satisfying:*

- $T \vdash_{\mathcal{D}} t$ *for any* $t \in T \subseteq \mathsf{Terms}(\Sigma, \mathcal{T}, \preccurlyeq)$
- *If* $T \vdash_{\mathcal{D}} t_1\sigma, \ldots T \vdash_{\mathcal{D}} t_n\sigma$ *for some substitution* σ *and* $\frac{t_1 \quad \cdots \quad t_n}{t} \in \mathcal{D}$ *then* $T \vdash_{\mathcal{D}} t\sigma$.

We may omit the subscript $\mathcal{D}$ in $\vdash_{\mathcal{D}}$ when it is clear from the context. For all deduction systems $\mathcal{D}$ in this paper we require $\frac{}{g^l_\tau}$ for all garbage symbols $g_\tau \in \Sigma$ and $l \in \mathsf{labelsA}$.

Let σ be a substitution. We say that $\frac{t_1\sigma \quad \cdots \quad t_n\sigma}{t\sigma}$ is an *instantiation* of a rule $\frac{t_1 \quad \cdots \quad t_n}{t} \in \mathcal{D}$. Since we require the deduction relations in this paper to be efficiently decidable, we can, if we have $T \vdash t$, w.l.o.g. always find a sequence $\pi = T \overset{\alpha_1}{\to} T_1 \overset{\alpha_2}{\to} \cdots \overset{\alpha_n}{\to} T_n$ such that for all $i \in \{1, \ldots, n\}$: (i) $\alpha_i = \frac{t_1 \quad \cdots \quad t_n}{t'}$ is an instantiation of rules from $\mathcal{D}$, (ii) $t_1, \ldots, t_n \in T_{i-1}$, (iii) $t' \notin T_{i-1}$, (iv) $t' \in T_i$ and (v) $t \in T_n$. We call π a *deduction proof* for $T \vdash t$.

From now on we denote a symbolic model $\mathcal{M}$ as a tuple $(\mathcal{T}, \preccurlyeq, \Sigma, \mathcal{D})$ where $\mathcal{T}$ is the set of data types, $\preccurlyeq$ the subtype relation, Σ signature and $\mathcal{D}$ the deduction system. For all symbolic models defined in this paper we omit the garbage symbols and the corresponding reduction rules for the sake of brevity.

3.2.1 Reconciling the notions for symbolic models

In this section we motivate and explain the differences between the notion of a symbolic model just introduced and the one for the applied pi calculus from Section 2.1.

Data Types.

In contrast to symbolic models from Section 2.1, the notion used here features data types and a subtype relation. The reasons for this are many-fold. While an untyped symbolic model sets no limits as to how function symbols can be composed, there are a lot of combinations that do not make sense from a practical point of view. E.g., we do not want to use a ciphertext as verification key. Furthermore, the goal of this chapter is to compose implementations of different symbolic models and maintain soundness. The implementation of some function symbol may be sound if used appropriately together with other function symbols from the same model but unsound if applied to other data. Hence we need restrictions for the sake of composability. Data types are a natural way to restrict the usage of functions symbols this way. Note that from a conceptional point of view, we do only restrict the behavior of honest parties, not the behavior of the adversary. An alternative to the introduction of data types is stating the necessary restrictions explicitly, as done in CoSP [7] for example (see "Protocol conditions", [7], page 17).

To extend the symbolic model for the applied calculus with data types is possible and even the way Proverif actually works [21]. Furthermore, results from Chapter 2 canonically carry over to a typed calculus like that of Proverif.

Labels vs. Names.

Another difference between symbolic models in this chapter and symbolic models in Chapter 2 is that labels and constant function symbols take the role of names.

Names in the original pi calculus are very versatile and thus are used for various purposes which differ in nature. They are used as identifiers for parties or channel; or they are used as sources of entropy (nonces) without distinguishing whether the source of the randomness is adversarial or honest. To get computational soundness, the different use cases of names have to be distinguished. The notion for symbolic models in this section achieves this by using (honest and dishonest) labels as the only sources of randomness. Constant function symbols take the role of identifiers.

Again, instead of using labels we could also use names together with appropriate protocol conditions as in CoSP [7] for example. The two approaches are convertible. We picked labels to have less additional conditions and to be compatible with prior work [48].

Deduction Relation vs. Equational Theory and Rewrite Rules.

Finally, the two notions of symbolic models use different approaches to define a semantic for terms. While symbolic models in Chapter 2 feature constructors, destructors an equational theory and rewrite rules, the symbolic models in this chapter come equipped with a deduction system only.

This, in contrast to the aforementioned dissimilarities, constitutes a real restriction of symbolic models in this chapter in comparison with symbolic models from Chapter 2. Deduction soundness is only concerned with the knowledge of the adversary. That is, we do not explicitly specify protocols that the adversary can attack but instead let the adversary decide which terms it can observe.

To get a soundness result for protocols, we would have to augment the symbolic model of this section with rewrite rules or an equational theory. This is briefly discussed in [48]. In addition to being deduction sound, the implementation would then have to respect the equational theory, i.e., $T =_{\mathrm{E}} T'$ implies that the computational interpretations are indistinguishable as well. To extend deduction soundness accordingly would be an interesting direction for future work (see Chapter 4).

3.3 Implementation

An *implementation* $\mathcal{I}$ of a symbolic model is a family of tuples $(M_\eta, [\![\cdot]\!]_\eta, len_\eta, \mathsf{open}_\eta, \mathsf{valid}_\eta)_\eta$ for $\eta \in \mathbb{N}$. We usually omit the security parameter and just write $(M, [\![\cdot]\!], len, \mathsf{open}, \mathsf{valid})$ for an implementation.

M is a Turing machine which provides concrete algorithms working on bitstrings for the function symbols in the signature. $[\![\cdot]\!] : \mathcal{T} \to 2^{\{0,1\}^*}$ is a function that maps each type to a set of bitstrings. $len : \mathsf{Terms} \to \mathbb{N}$ computes the length of a term if interpreted as a bitstring. With open the implementation provides an algorithm to interpret bitstrings as terms. valid is a predicate which states whether a concrete use of the implementation is valid. For example, a correct use of an implementation might exclude the creation of key cycles or dynamic corruption of keys from the valid use cases. More precisely we require the following from an implementation:

We assume a non-empty set of bitstrings $[\![\tau]\!] \subseteq \{0,1\}^\eta$ for each type $\tau \in \mathcal{T}$. For the base type $\top$, we assume $[\![\top]\!] = \{0,1\}^*$ and for any pair of types $\tau, \tau' \in \mathcal{T}$ with $\tau \preccurlyeq \tau'$ we require $[\![\tau]\!] \subsetneq [\![\tau']\!]$ and $[\![\tau]\!] \cap [\![\tau']\!] = \emptyset$ otherwise (i.e., if $\tau \not\preccurlyeq \tau'$). We write $[\![\mathcal{T}]\!]$ for $\cup_{\tau \in \mathcal{T}\setminus\{\top\}} [\![\tau]\!]$. Later, we often make use of a function $\langle c_1, \ldots, c_n, \tau \rangle$ that takes a list of bitstrings $c_1, \ldots, c_n$ and a type τ and encodes $c_1, \ldots, c_n$ as a bitstring $c' \in [\![\tau]\!]$. We assume that this encoding is bijective, i.e., we can uniquely parse c' as $\langle c_1, \ldots, c_n, \tau \rangle$ again.

We require the Turing machine M itself to be deterministic. However, each time it is run, it is provided with a random tape $\mathcal{R}$. More specifically, we require for each $f \in \Sigma$ with $\mathrm{ar}(f) = \tau_1 \times \ldots \times \tau_n \to \tau$ that is not a garbage symbol that for input f M calculates a function $(M\ f)$ with domain $[\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!] \times \{0,1\}^*$ and range $[\![\tau]\!]$. The runtime of M and $(M\ f)$ has to be polynomial in the length of its input. Intuitively, to generate a bitstring for a term $t = f^l(t_1, \ldots, t_n)$ we apply $(M\ f)$ to the bitstrings generated for the arguments t_i and some randomness (which might not be used for deterministic function symbols). We call the resulting bitstring *concrete interpretation* of t. The randomness is provided by the $\mathsf{generate}$ function introduced in Section 3.3.2.

3.3.1 Interpretations

In cryptographic applications functions are often randomized and the same random coins may occur in different places within the same term. This is the case for instance when the same nonce occurs twice in the same term. We use a (partially defined) mapping $L : \{0,1\}^* \to \mathsf{HTerm}$ from bitstrings to *hybrid terms* to record this information. A hybrid term is either a garbage term or $f^l(c_1, \ldots, c_n)$ where $f \in \Sigma$ is a function symbol of arity n applied to bitstrings $c_i \in \{0,1\}^*$. By $\mathrm{dom}(L) \subseteq 2^{\{0,1\}^*}$ we denote the domain of L, i.e. the set of bitstrings for which L is defined. The mapping L induces an interpretation of bitstrings as terms. We define the *interpretation of bitstring* $c \in \mathrm{dom}(L)$ *with respect to a mapping* L as $L[[c]] := f^l(L[[c_1]], \ldots, L[[c_n]])$ if $L(c) = f^l(c_1, \ldots, c_n)$ and $L[[c]] := L(c)$ if $L(c)$ is a garbage term. We say that a mapping L is complete, if for all $(c, f^l(c_1, \ldots, c_n)) \in L$ $c_1, \ldots, c_n \in \mathrm{dom}(L)$. Note that $L[[c]]$ is only defined if L is complete.

3.3.2 Generating function

Given a mapping L we define a *generate function* that associates a concrete semantics for terms (given the terms already interpreted in L).

This function uses a random tape $\mathcal{R}$ as a source of randomness for M when generating the concrete interpretation of terms. We assume that there is an algorithm

```
$\mathsf{generate}_{M,\mathcal{R}}(t, L)$:
    if for some $c \in \mathrm{dom}(L)$ we have $L[[c]] = t$ then
        return $c$
    else
        for $i \in \{1, \ldots, n\}$ let $(c_i, L) := \mathsf{generate}_{M,\mathcal{R}}(t_i, L)$
        let $r := \mathcal{R}(t)$
        let $c := (M\ f)(c_1, \ldots, c_n; r)$
        let $L(c) := f^l(c_1, \ldots, c_n))$ $(l \in \mathsf{labelsH})$
        return $(c, L)$
```

Figure 3.1: The generate function (t is of the form $f^l(t_1, \ldots, t_n)$ (with possibly $n = 0$ and no label l for deterministic function symbols f)).

$\mathcal{R}(t)$ which maps a term t to a bitstring $r \in \{0,1\}^\eta$ that should be used as the randomness when t is generated. Even changing only one label in t leads to a changed term t' for which different randomness will be used. Figure 3.1 defines the generate function given a closed term $t = f^l(t_1, \ldots, t_n)$ and a mapping L.

Note that $\mathsf{generate}_{M,\mathcal{R}}(t, L)$ not only returns a bitstring c associated to t but also updates L (to remember, for example, the value associated to t). Note also that $\mathsf{generate}_{M,\mathcal{R}}$ depends on M and the random tape $\mathcal{R}$. When needed, we explicitly show this dependency, but in general we avoid it for readability. If a mapping L is complete, then for $(c, L') := \mathsf{generate}(t, L)$, L' is complete. Furthermore, the $\mathsf{generate}$ function requires that, for given inputs t, L, the following holds: For all $t' := f^l(t_1, \ldots, t_n) \in st(t)$ where $l \in \mathsf{labelsA}$ we find a $c \in \mathrm{dom}(L)$ s.t. $L[[c]] = t'$ and t doesn't contain garbage symbols carrying honest labels. This guarantees that all bitstrings introduced by the $\mathsf{generate}$ function correspond to the application of non-garbage function symbols carrying honest labels.

3.3.3 Parsing function

Conversely, we require the implementation to define a function parse to convert bitstrings into terms. The function takes a bitstring c and a mapping L as input and returns a term t and an extended mapping L.

For parsing functions we require the concrete structure in Figure 3.2 (where $\mathsf{open} : \{0,1\}^* \times \mathsf{libs} \to \{0,1\}^* \times \mathsf{HTerm}$ a function that on call $\mathsf{open}(c, L)$ parses the bitstring c in presence of the library L and returns its hybrid interpretation).

The exact definition of parse is left unspecified, as it depends on the particular behavior of open which is provided by a concrete implementation. We require this structure for the parsing function to provide a concrete context in which the open function of different implementations can be composed. Note that the open function is only allowed to use honestly generated bitstrings when dealing with a term. We will furthermore only use open functions later that ignore “foreign” bitstrings in the given library, i.e., bitstrings that are of a data type that is not part of the implementation open belongs to. Due to these properties the composition of open functions is commutative. This is important for our composition theorems later. Furthermore, we think that it meets the intuition that the composition of different implementations should be commutative.

```
$\mathsf{parse}(c, L)$:
  if $c \in \mathrm{dom}(L)$ then
    return $(L[[c]], L)$
  else
    let $L_h := \{(\overline{c}, f^l(\dots)) \in L : l \in \mathsf{labelsH}\}$
    let $L := \left(\bigcup_{(\overline{c},\cdot)\in L} \mathsf{open}(\overline{c}, L_h)\right)$
    let $G := \left\{(\overline{c}, g_{\mathsf{T}}^{l(\overline{c})})\right\}$ $(l(\overline{c}) \in \mathsf{labelsA})$
    do
      let $L := (L \setminus G) \cup \left(\bigcup_{(\overline{c},\cdot)\in G} \mathsf{open}(\overline{c}, L_h)\right)$
      let $G := \left\{(\overline{c}, g_{\mathsf{T}}^{l(\overline{c})}) : (\overline{c}', f(\dots, \overline{c}, \dots)) \in L \text{ and } \overline{c} \notin \mathrm{dom}(L)\right\}$
    while $G \neq \emptyset$
    return $(L[[c]], L)$
```

Figure 3.2: The parsing function.

```
$\mathsf{generate}'_{M,\mathcal{R}}(t, L)$:
    if for some $c \in \mathrm{dom}(L)$ we have $L[[c]] = t$ then
        return $c$
    else
        for $i \in \{1, n\}$ let $(c_i, L) := \mathsf{generate}'_{M,\mathcal{R}}(t_i, L)$
        let $r := \mathcal{R}(t)$
        let $c := (M\ f)(c_1, \dots, c_n; r)$
        if $c \in \mathrm{dom}(L)$ then
            exit game with return value 1 (collision)
        let $L(c) := f^l(c_1, \dots, c_n))$ $(l \in \mathsf{labelsH})$
        return $(c, L)$
```

Figure 3.3: A collision-aware generate function.

3.3.4 Good implementation

Until now we have not restricted the behavior of implementations in any way. However, there are some properties we will need to hold for every implementation. We describe these properties in this section and say that a *good implementation* is one that satisfies all of them.

We stipulate that a good implementation is *length regular*, i.e., $len(f^l(t_1, \dots, t_n)) := |(M\ f)(c_1, \dots, c_n; r)|$ depends only on the length of the arguments c_i (which are the computational interpretations of the symbolic arguments t_i). Having such a length function is equivalent to having a set of length functions $len_f : \mathbb{N}^n \to \mathbb{N}$ for each function symbol $f \in \Sigma$ with n arguments. We need this to generically compose length functions of different implementations in Section 3.5.

We now explain what it means for an implementation to be *collision free*. A collision occurs if during a call of $\mathsf{generate}_{M,\mathcal{R}}(t, L)$ an execution of M yields a bitstring c that is already in the domain of L. Since the library L has to be well-defined, we can either overwrite the old value $L(c)$ with the new one or discard the new value. Both variants are problematic:

Overwriting changes the behavior of parse (i.e., bitstrings may now be parsed differently). This might have severe consequences. Imagine that the overwritten bitstring was an honestly signed message. Now this signature looks like the signature of a different message symbolically; possibly like a forgery. Note that this would not be a weakness of signatures but of the fact that collisions can be found for bitstrings corresponding to the signed terms. Discarding means that a bitstring c generated for a term t might not be parsed as t later which might wrongfully prevent the adversary from winning the soundness game.

Since we also need transparent implementations to be collision free we and still want the notion of collision freeness to be composable later, we need to fix a set of functions that reflect the capability of the adversary to pick arbitrary bitstrings for arguments of $\top$.

Definition 40 (Supplementary transparent functions)**.** *For a set of bitstrings $\mathcal{B} \subseteq \{0,1\}^*$ we define the transparent model $\mathcal{M}^{\mathsf{tran}}_{\mathsf{supp}}(\mathcal{B})$ as follows:*

- $\mathcal{T}^{\mathsf{tran}}_{\mathsf{supp}} := \{\top, \tau^{\mathsf{tran}}_{\mathsf{supp}}\}$. *$\tau^{\mathsf{tran}}_{\mathsf{supp}}$ is a subtype of $\top$.*
- $\Sigma^{\mathsf{tran}}_{\mathsf{supp}} := \{f_c : c \in \mathcal{B}\}$ *(all function symbols are deterministic)*
- $\mathcal{D}^{\mathsf{tran}}_{\mathsf{supp}} := \{\frac{}{f_c()} : c \in \mathcal{B}\}$

and an implementation $\mathcal{I}^{\mathsf{tran}}_{\mathsf{supp}}(\mathcal{B})$ as follows:

- $[\![\tau^{\mathsf{tran}}_{\mathsf{supp}}]\!] := \mathcal{B}$
- $(M^{\mathsf{tran}}_{\mathsf{supp}}\ f_c)()$ *returns c*
- $(M^{\mathsf{tran}}_{\mathsf{supp}}\ \mathsf{func})(c)$ *returns f_c if $c \in \mathcal{B}$, $\bot$ otherwise*

Now we are ready to formally define what collision freeness means.

Definition 41 (Collision-free implementation)**.** *Let $\mathsf{DS}'_{\mathcal{M},\mathcal{I},\mathcal{A}}(\eta)$ be the deduction soundness game from Figure 3.7 where we replace the* generate *function by the function* generate′ *from Figure 3.3. We say that an implementation $\mathcal{I}$ is* collision-free *if for all PPT adversaries $\mathcal{A}$*

$$\mathbb{P}\left[\mathsf{DS}_{\mathcal{M}\cup\mathcal{M}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}]\!]),\mathcal{I}\cup\mathcal{I}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}]\!]),\mathcal{A}}(\eta)=1\right]$$
$$-\mathbb{P}\left[\mathsf{DS}'_{\mathcal{M}\cup\mathcal{M}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}]\!]),\mathcal{I}\cup\mathcal{I}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}]\!]),\mathcal{A}}(\eta)=1\right]$$

is negligible.

When we compose implementations later we will need that their open functions do not interfere. Intuitively, each open function should stick to opening the bitstrings it is responsible for (i.e., that are of types belonging to the same implementation the open function belongs to). This is reflected in the following definition.

Definition 42 (type-safe implementation)**.** *We say that an implementation $\mathcal{I}$ of a symbolic model $\mathcal{M}$ is* type safe *if*

(i) $\mathsf{open}(c, L) = (c, g^l_\top)$ *for $l \in$* labelsA *if $c \notin [\![\mathcal{T}]\!]$. (“*open *must not deal with foreign bitstrings.”)*

(ii) $\mathsf{open}(c, L) = \mathsf{open}(c, L|[\![\mathcal{T}]\!])$ *where $L|[\![\mathcal{T}]\!] := \{(c,h) \in L : \exists \tau \in \mathcal{T} \setminus \{\top\} : c \in [\![\tau]\!]\}$. (“The behavior of* open *must not be affected by foreign bitstrings in the library.”)*

Since we need to simulate parsing later, we require $\mathsf{parse}(c, L)$ (based on open) to run in polynomial time in the size of the library.

Restrictions for valid.

Usually, to have computational soundness, we have to restrict the use of the implementation. For example we may only allow static corruption of keys. The purpose of the valid function is exactly this. It gets a trace of queries and outputs a boolean value which states whether the trace is valid or not. To be able to compose the valid functions of different implementations in a meaningful way we require valid to meet the following requirements.

A *trace* $\mathbb{T}$ is a list of queries q. A *query* is either “init T, H” where T, H are lists of terms, “sgenerate t”[1], or “generate t” where t is a term.

(i) If $\mathsf{valid}(\mathbb{T} + q) = \mathsf{true}$, then $\mathsf{valid}(\mathbb{T} + \hat{q}) = \mathsf{true}$ where $\hat{q}$ is a *variation* of q: If $q =$“generate t”, then $\hat{q} =$“generate $\hat{t}$” (analogously for “sgenerate t”). Here, $\hat{t}$ is a variation of t according to the following rule: Any subterm $f^l(t_1, \ldots, t_n)$ of t where f is a *foreign function symbol* (i.e., $f \notin \Sigma$) may be replaced by $\hat{f}^{\hat{l}}(\hat{t}_1, \ldots, \hat{t}_m)$ where $\hat{f} \notin \Sigma$ is a foreign function symbol and $\hat{t}_i = t_j$ for some $j \in \{1, \ldots, n\}$ (where each t_j may only be used once) or $\hat{t}_i$ does not contain function symbols from Σ. As a special case we may also replace $f^l(t_1, \ldots, t_n)$ with a term $\hat{t}_1$ (i.e., $\hat{f}$ is “empty”). If $q =$ ‘“init T, H” then $\hat{q} =$“init $\hat{T}, \hat{H}$” where $T = (t_1, \ldots, t_n)$ and $\hat{T} = (\hat{t}_1, \ldots, \hat{t}_n)$ and $\hat{t}_i$ is a variation of t_i ($\hat{H}$ analogously).

(ii) If $\mathsf{valid}(\mathbb{T} + q) = \mathsf{true}$ and t is a term occurring in q, then $\mathsf{valid}(\mathbb{T}+$“sgenerate t'“$) = \mathsf{true}$ for any subterm t' of t.

(iii) $\mathsf{valid}(\mathbb{T})$ can be evaluated in polynomial time (in the length of the trace $\mathbb{T}$).

Why are these restrictions necessary?

(i) This allows us to replace function symbols with transparent functions and even add or drop arguments during the simulation of a primitive using transparent functions. Intuitively, this requirement is justified since we don’t know the semantics of foreign function symbols valid should not: (a) look at the concrete symbols (i.e., function symbols may be replaced), (b) look at the order of arguments (since it doesn’t know what the foreign function does, valid shouldn’t make decisions based on the order of arguments; also, if the reader accepts (a) a function symbol could be replaced by a semantically equivalent function symbol which just accepts arguments in a different order), (c) depend on the existence of own terms: since the foreign function might just ignore an argument it wouldn’t be meaningful to require its existence, (d) the existence of additional arguments for a foreign function (those could also be hardcoded).

(ii) If a term is valid in general, then any subterm should be valid at least if the adversary doesn’t learn it. We need this when we add something to an implementation that features arguments that are hidden from the adversary (i.e., encryptions under honest keys). We cannot simulate those arguments with transparent functions and therefore need to generate them at some point.

(iii) This is needed to efficiently compute $\mathsf{valid}(\mathbb{T})$ during simulations.

[1] The s in sgenerate stands for “silent”; the adversary does not see any output for this kind of request. See Section 3.6 for details.

3.4 Transparent functions

Typical primitives that are usually considered in soundness results include encryption, signatures, hash functions, etc.. Intuitively, such functions are efficiently invertible, and the type of their output can be efficiently determined. An example for such functions are data structures (i.e., pairs, lists, XML documents, etc.). We define and study soundness of such primitives when they are used together with a class of functions which we call *transparent functions*.

Towards this goal we define *transparent symbolic models* and the corresponding *transparent implementation* and show how to extend symbolic models and their implementations with transparent functions in a generic way.

A transparent symbolic model $\mathcal{M}_{\mathsf{tran}} = (\mathcal{T}_{\mathsf{tran}}, \preccurlyeq_{\mathsf{tran}}, \Sigma_{\mathsf{tran}}, \mathcal{D}_{\mathsf{tran}})$ is a symbolic model where the deduction system is defined as follows (the label is omitted for deterministic function symbols):

$$\mathcal{D}_{\mathsf{tran}} = \left\{ \begin{array}{l} \frac{t_1 \ \cdots \ t_n}{f^l(t_n,\ldots,t_n)} \ l \in \mathsf{labelsA}, f \in \Sigma_{\mathsf{tran}} \\ \\ \frac{f^l(t_1,\ldots,t_n)}{t_i} \ 1 \leq i \leq n, l \in \mathsf{labels}, f \in \Sigma_{\mathsf{tran}} \end{array} \right\}$$

Formally, a transparent implementation of a transparent symbolic model $\mathcal{M} = (\mathcal{T}, \preccurlyeq, \Sigma, \mathcal{D})$ is an implementation (and thus adhering to the requirements from Section 3.3.4) $\mathcal{I}_{\mathsf{tran}} = (M_{\mathsf{tran}}, [\![\cdot]\!], len, \mathsf{open}_{\mathsf{tran}}, \mathsf{valid}_{\mathsf{tran}})$ where $\mathsf{open}_{\mathsf{tran}}$ and $\mathsf{valid}_{\mathsf{tran}}$ are defined explicitly below. We require two additional modes of operation, func and proj, for the Turing machine M_{tran} such that for all $f \in \Sigma$ with $\mathrm{ar}(f) = \tau_1 \times \ldots \times \tau_n \to \tau$

$$\begin{array}{l} (M_{\mathsf{tran}}\ \mathsf{func}) : \{0,1\}^* \to \Sigma \cup \{\bot\} \\ (M_{\mathsf{tran}}\ \mathsf{proj}\ f\ i) : \{0,1\}^* \to \{0,1\}^* \cup \{\bot\} \end{array}$$

and we have for any $c_i \in [\![\tau_i]\!]$, $1 \leq i \leq n$, $r \in \{0,1\}^\eta$

$$\begin{array}{l} (M_{\mathsf{tran}}\ \mathsf{func})((M_{\mathsf{tran}}\ f)(c_1,\ldots,c_n;r)) = f \\ (M_{\mathsf{tran}}\ \mathsf{proj}\ f\ i)((M_{\mathsf{tran}}\ f)(c_1,\ldots,c_n;r)) = c_i \end{array}$$

Furthermore, we require $(M_{\mathsf{tran}}\ \mathsf{func})(c) = \bot$ for all $c \notin [\![\mathcal{T}]\!]$. As expected, M_{tran} is required to run in polynomial time in η for this modes of operation as well.

For transparent implementations we explicitly define the open function $\mathsf{open}_{\mathsf{tran}}$ as in Figure 3.4. Note that a transparent implementation is automatically type safe according to Definition 42: Property (i) is required above and property (ii) holds since L is not used by $\mathsf{open}_{\mathsf{tran}}$.

We define $\mathsf{valid}_{\mathsf{tran}}(\mathbb{T}) = \mathsf{true}$ for all traces $\mathbb{T}$, i.e., the use of transparent functions is not restricted in any way.

3.5 Composition

We next explain how to generically compose two symbolic models and their corresponding implementations.

Let $\mathcal{M}_1 = (\mathcal{T}_1, \preccurlyeq_1, \Sigma_1, \mathcal{D}_1)$ and $\mathcal{M}_2 = (\mathcal{T}_2, \preccurlyeq_2, \Sigma_2, \mathcal{D}_2)$ be symbolic models and $\mathcal{I}_1 = (M_1, [\![\cdot]\!]_1, len_1, \mathsf{open}_1, \mathsf{valid}_1)$ and $\mathcal{I}_2 = (M_2, [\![\cdot]\!]_2, len_2, \mathsf{open}_2, \mathsf{valid}_2)$ implementations of $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively.

We say that that $(\mathcal{M}_1, \mathcal{I}_1)$ and $(\mathcal{M}_2, \mathcal{I}_2)$ are *compatible* if $\mathcal{M}_1$ and $\mathcal{M}_2$ as well as $\mathcal{I}_1$ and $\mathcal{I}_2$ meet the requirements for compositions of symbolic models and implementations stated below respectively.

```
$\mathsf{open}_{\mathsf{tran}}(c, L)$:
  if $c \in [\![\mathcal{T}]\!] \cap \mathrm{dom}(L)$ then
    return $(c, L(c))$
  else if $(M_{\mathsf{tran}}\ \mathsf{func})(c) = \bot$ then
    find unique $\tau \in \mathcal{T}$ s.t. $c \in [\![\tau]\!]$ and
      $c \notin [\![\tau']\!]$ for all $\tau' \in \mathcal{T}$ with $[\![\tau']\!] \subsetneq [\![\tau]\!]$
    return $(c, g_{\tau}^{l(c)})$ $(l(c) \in \mathsf{labelsA})$
  else
    let $f := (M_{\mathsf{tran}}\ \mathsf{func})(c)$ $(\mathrm{ar}(f) = \tau_1 \times \cdots \times \tau_n \to \tau)$
    if $(M_{\mathsf{tran}}\ \mathsf{proj}\ f\ i)(c) = \bot$ for some $i \in \{1, \ldots, n\}$ then
      return $(c, g_{\tau}^{l(c)})$ $(l(c) \in \mathsf{labelsA})$
    else
      for $i \in \{1, \ldots, n\}$ do
        let $\widetilde{c}_i := (M_{\mathsf{tran}}\ \mathsf{proj}\ f\ i)(\widetilde{c})$
      return $(c, f^{l(c)}(c_1, \ldots, c_n))$ $(l(c) \in \mathsf{labelsA})$
```

Figure 3.4: Parsing algorithm for a transparent implementation.

Requirements for Symbolic Model Composability.

For symbolic models $\mathcal{M}_1$ and $\mathcal{M}_2$ to be composable, we require

(i) $\Sigma_1 \cap \Sigma_2 = \{g_\top\}$

(ii) $\mathcal{T}_1 \cap \mathcal{T}_2 = \{\top\}$

We then define the composition $\mathcal{M}' := \mathcal{M}_1 \cup \mathcal{M}_2$ and have $\mathcal{T}' := \mathcal{T}_1 \cup \mathcal{T}_2$, $\preccurlyeq' := \preccurlyeq_1 \cup \preccurlyeq_2$, $\Sigma' := \Sigma_1 \cup \Sigma_2$ and $\mathcal{D}' := \mathcal{D}_1 \cup \mathcal{D}_2$.

Requirements for Implementation Composability.

The corresponding implementations $\mathcal{I}_1 = (M_1, [\![\cdot]\!]_1, \mathit{len}_1, \mathsf{open}_1, \mathsf{valid}_1)$ and $\mathcal{I}_2 = (M_2, [\![\cdot]\!]_2, \mathit{len}_2, \mathsf{open}_2, \mathsf{valid}_2)$ can be composed if the following requirements are met:

(i) For all types $\tau_1 \in \mathcal{T}_1 \setminus \{\top\}$, $\tau_2 \in \mathcal{T}_2 \setminus \{\top\}$ we have $[\![\tau_1]\!] \cap [\![\tau_2]\!] = \emptyset$.

(ii) The composition of $\mathcal{I}_1$ and $\mathcal{I}_2$ (as defined below) is a collision-free implementation of $\mathcal{M}'$ (Definition 41).

We then define the composition $\mathcal{I}' = \mathcal{I}_1 \cup \mathcal{I}_2$ as follows:

The Turing machine $(M'\ f)$ returns $(M_1\ f)$ for $f \in \Sigma_1$ and $(M_2\ f)$ if $f \in \Sigma_2$. This is non-ambiguous due to requirement for symbolic model composability (i).[2] Similarly, for all $\tau \in \mathcal{T}_1$ we set $[\![\tau]\!]' := [\![\tau]\!]_1$ and analogously for $\tau \in \mathcal{T}_2$. Note that $[\![\top]\!] = [\![\top]\!]_1 = [\![\top]\!]_2 = \{0,1\}^*$. Since implementations are required to be length regular we can also compose the length functions len_1 and len_2 in a straightforward way to get len'.

To compose the open functions we define

[2] Here we assume that the membership problem is efficiently decidable (since M' has to run in polynomial time). This can be achieved w.l.o.g. with a suitable encoding for the function symbols.

```
$(\mathsf{open}_1 \circ \mathsf{open}_2)(c, L)$:
  let $(c, t) := \mathsf{open}_1(c, L)$
  if $t = g^l_\top$ for some $l \in \mathsf{labelsA}$ then
    return $\mathsf{open}_2(c, L)$
  else
    return $(c, t)$
```

and consequently $\mathsf{open}' := \mathsf{open}_1 \circ \mathsf{open}_2$. Furthermore we set $\mathsf{valid}'(\mathbb{T}) := \mathsf{valid}_1(\mathbb{T}) \wedge \mathsf{valid}_2(\mathbb{T})$ where $\wedge$ is the conjunction.

Finally, we have to show that the composed implementation $\mathcal{I}'$ is a good implementation of the composed symbolic model $\mathcal{M}'$ by checking the requirements from Section 3.3: Requirements for types hold since they hold on $\mathcal{T}_1$ and $\mathcal{T}_2$ and by requirement (ii) for the composition of symbolic models. The latter furthermore implies $[\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!] = \emptyset$. Due to this and since $\mathcal{I}_1$ and $\mathcal{I}_2$ are type safe, $\mathsf{open}' = \mathsf{open}_1 \circ \mathsf{open}_2 = \mathsf{open}_2 \circ \mathsf{open}_1$. Furthermore, $\mathcal{I}'$ is type safe since $\mathcal{I}_1$ and $\mathcal{I}_2$ are. The requirements for valid carry over obviously as well as the length regularity.

Unfortunately, it is not always straightforward to check requirement (ii) for the composition of implementations. However, we are going to show that (ii) is satisfied if, additionally to the other requirements some additional requirement for the valid functions of $\mathcal{I}_1$ and $\mathcal{I}_2$ hold. This is reflected in the following Lemma 58. We note that the valid predicates of the primitives we introduce later (public key encryption, signatures, secret key encryption, MACs and hashes) all meet the additional requirements of Lemma 58. Hence we do not need to proof collision freeness separately when composing those.

Lemma 58. *Let $\mathcal{M}_1$, $\mathcal{M}_2$ be symbolic models with implementations $\mathcal{I}_1$ and $\mathcal{I}_2$ respectively. If in addition to requirements (i), (ii) for symbolic model composability and (i) for implementation composability the following requirements for* $\mathsf{valid}'(\mathbb{T}) := \mathsf{valid}_1(\mathbb{T}) \wedge \mathsf{valid}_2(\mathbb{T})$ *hold:*

1. *Let $\hat{\mathbb{T}}$ be $\mathbb{T}$ with all silent generate queries "$\mathsf{sgenerate}\ t$" replaced with normal generate queries "$\mathsf{generate}\ t$". Then* $\mathsf{valid}'(\mathbb{T}) \Rightarrow \mathsf{valid}(\hat{\mathbb{T}})$.

2. *Let $x \in \{1, 2\}$. If* valid_x*("$\mathsf{init}\ T, H$"), then for each $t \in T \cup H$ all function symbols in t are from Σ_x or no function symbol in t is from Σ_x.*

3. *Let $x \in \{1, 2\}$. Let $\hat{\mathbb{T}}$ be an expansion of $\mathbb{T} = q_1 + \cdots + q_n$ in the following sense: A $q_i =$"$\mathsf{generate}\ t$" for $i \in \{1, \ldots, n\}$ is be replaced with $q_i^1 + \cdots + q_i^m$ where $q_i^j =$"$\mathsf{generate}\ t_j$", $t_j \in st(t)$ and t_j does not contain function symbols from Σ_x for $j \in \{1, \ldots, m\}$. Then* $\mathsf{valid}_x(\mathbb{T}) \Rightarrow \mathsf{valid}_x(\hat{\mathbb{T}})$.

Then $(\mathcal{M}_1, \mathcal{I}_1)$ and $(\mathcal{M}_2, \mathcal{I}_2)$ are compatible.

Proof. Note that $\mathcal{I}_1$ and $\mathcal{I}_2$ are collision free since we are only dealing with good implementations. We prove the lemma with a sequence of games:

Game 0

In Game 0 $\mathcal{A}$ plays $\mathsf{DS}'_{\mathcal{M}' \cup \mathcal{M}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}']\!]), \mathcal{I}' \cup \mathcal{I}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}']\!]), \mathcal{A}}(\eta)$ from Definition 41.

Game 1

In Game 1 $\mathcal{A}$ plays Game 0 where the $\mathsf{generate}$ functions aborts only for collisions of bitstrings from $[\![\mathcal{T}_1]\!]$, i.e., we use a function $\mathsf{generate}'$ similar to that from Figure 3.3 with:

```
if $c \in \mathrm{dom}(L) \cap [\![\mathcal{T}_1]\!]$ then
    exit game with return value 1 (collision)
```

Claim: Game 1 and Game 0 are indistinguishable

Since the only difference between the games is the changed exit condition, it suffices to look at the probability of a 1-output: We show for any adversary $\mathcal{A}$ that wins Game 0 but not Game 1 with non-negligible probability that it can be used to break the collision-freeness of $\mathcal{I}_2$. Concretely, we provide a simulator $\mathcal{B}$ that plays

$$\mathsf{DS}'_{\mathcal{M}_2\cup\mathcal{M}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}_2]\!]),\mathcal{I}_2\cup\mathcal{I}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}_2]\!]),\mathcal{A}}(\eta)$$

and simulates Game 0 for $\mathcal{A}$.

Setup. $\mathcal{B}$ maintains a couple of global states: $S := \emptyset$ to keep track of the terms generated for $\mathcal{A}$, $L := \emptyset$ to simulate the library for $\mathcal{I}_1$, $\Lambda := \emptyset$ is a partially defined mapping from transparent functions f_c ($c \in [\![\mathcal{T}_1]\!]$) to terms, $\mathcal{R} \leftarrow \{0,1\}^*$ is the random tape of and $\mathbb{T}$ the trace of queries received from $\mathcal{A}$.

Queries. Using the two helper functions $\mathsf{generate}_{\mathcal{B}}$ and $\mathsf{parse}_{\mathcal{B}}$, which will be defined below, $\mathcal{B}$ deals with the queries of $\mathcal{A}$ as follows (note that $\mathcal{A}$ doesn't send parameters in the collision game according to def Definition 41):

- "init T, H": $\mathcal{B}$ adds "init T, H" to $\mathbb{T}$ and returns 0 if $\mathsf{valid}'(\mathbb{T}) = \mathsf{false}$. Otherwise, it computes $C := \{\mathsf{generate}_{\mathcal{B}}(t) : t \in T\}$ and $\mathsf{generate}_{\mathcal{B}}(t)$ for all $t \in H$ and sends C to $\mathcal{A}$.

- "$\mathsf{generate}$ t": $\mathcal{B}$ adds "$\mathsf{generate}$ t" to $\mathbb{T}$ and returns 0 if $\mathsf{valid}'(\mathbb{T}) = \mathsf{false}$. Otherwise, it adds t to S and sends $\mathsf{generate}_{\mathcal{B}}(t)$ to $\mathcal{A}$.

- "$\mathsf{sgenerate}$ t": $\mathcal{B}$ returns 0 if $\mathsf{valid}'(\mathbb{T}+$"$\mathsf{sgenerate}$ t"$) = \mathsf{false}$. Otherwise, it computes $\mathsf{generate}_{\mathcal{B}}(t)$ (but does not send the result to $\mathcal{A}$).

- "parse c": $\mathcal{B}$ computes $t := \mathsf{parse}_{\mathcal{B}}(c)$. If $S \vdash' t$, it sends t to $\mathcal{A}$. Otherwise it returns 1.

The $\mathsf{generate}_{\mathcal{B}}$ function. While $\mathcal{B}$ can compute $\mathcal{I}_1$ itself, it has only access to $\mathcal{I}_2$ via the game it is playing. Concretely, it can generate bitstrings for function symbols from Σ_1 directly (using the given machine M_1), while it has to query bitstrings for function symbols from the complement

$$\overline{\Sigma_1} := \Sigma' \setminus \Sigma_1 = \Sigma_2 \cup \{f_c : c \in [\![\mathcal{T}']\!]\}$$

This procedure is reflected in the function $\mathsf{generate}_{\mathcal{B}}$ from Figure 3.5. $\mathsf{generate}_{\mathcal{B}}$ updates the states L and Λ of $\mathcal{B}$ and makes use of the random tape $\mathcal{R}$. We write $t \in \Sigma$ if the term t contains only function symbols $f \in \Sigma$.

The $\mathsf{parse}_{\mathcal{B}}$ function. Analogously, $\mathcal{B}$ needs to distinguish bitstrings from the domain of $\mathcal{I}_1$ from bitstrings that have to be parsed by the game played by $\mathcal{B}$. It uses the function $\mathsf{parse}_{\mathcal{B}}$ from Figure 3.6 to handle parsing requests.

Indistinguishability. Finally, we argue that the simulation perfectly simulates Game 1 and that it can only be distinguished from Game 0 by an adversary that breaks the collision-freeness of $\mathcal{I}_2$. More concretely, we show

```
$\mathsf{generate}_{\mathcal{B}}(t)$:
    if $t \in \Sigma_1$ then
        let $(c, L) := \mathsf{generate}'_{M_1,\mathcal{R}}(t, L)$
        return $c$
    else if $t \in \overline{\Sigma_1}$ then
        return "$\mathsf{generate}\ t$"
    else
        if $f \in \Sigma_1$ then
            for $i \in \{1, \dots, n\}$ do
                let $c_i := \mathsf{generate}_{\mathcal{B}}(t_i)$
            if $L(c) = f^l(c_1, \dots, c_n)$ for some $c$ then
                return $c$
            else
                let $r := \mathcal{R}(f^l(t_1, \dots, t_n))$
                let $c := (M_1\ f)(c_1, \dots, c_n; r)$
                if $c \in \mathrm{dom}(L)$ then
                    exit game with return value 1 (collision)
                else
                    let $L(c) := f^l(c_1, \dots, c_n)$
                    return $c$
        else (i.e., $f \in \overline{\Sigma_1}$)
            for $i \in \{1, \dots, n\}$ do
                if $t_i \in \overline{\Sigma_1}$ then
                    $\tilde{t}_i := t_i$
                else
                    let $c_i := \mathsf{generate}_{\mathcal{B}}(t_i)$
                    let $\Lambda(f_{c_i}) := t_i$
                    $\tilde{t}_i := f_{c_i}$
            return"$\mathsf{generate}\ f^l(\tilde{t}_1, \dots, \tilde{t}_n)$"
```

Figure 3.5: $\mathsf{generate}$ function used by the simulator $\mathcal{B}$. $t = f^l(t_1, \dots, t_n)$ for a label $l \in \mathsf{labelsH}$. $\mathsf{generate}'$ is the collision-aware generate function from Figure 3.3.

```
$\mathsf{parse}_\mathcal{B}(c)$:
    if $c \in [\![\mathcal{T}_1]\!]$ then
        if $c \in \mathrm{dom}(L)$ then
            let $f^l(c_1, \dots, c_n) := L(c)$
        else
            let $L_h := \{(c, f^l(\dots)) \in L : l \in \mathsf{labelsH}$
            let $(c, f^l(c_1, \dots, c_n)) := \mathsf{open}_1(c, L_h)$
            let $L(c) := f^l(c_1, \dots, c_n)$
        for $i \in \{1, \dots, n\}$ do
            let $t_i := \mathsf{parse}_\mathcal{B}(c_i)$
        return $f^l(t_1, \dots, t_n)$
    else
        let $t :=$"$\mathsf{parse}\ c$"
        let $T := \{f_{\hat{c}} : f_{\hat{c}} \in st(t)\}$
        let $\sigma := \emptyset$
        for each $f_{\hat{c}} \in T \cap \mathrm{dom}(\Lambda)$ do
            let $\sigma(f_{\hat{c}}) := \Lambda(f_{\hat{c}})$
        for each bitstring $\hat{c}$ s.t. $f_{\hat{c}} \in T \setminus \mathrm{dom}(\Lambda)$ do
            let $\hat{t} := \mathsf{parse}_\mathcal{B}(\hat{c})$
            let $\sigma(f_{\hat{c}}) := \hat{t}$
        return $t\sigma$ (replace each $f_{\hat{c}}$ with $\sigma(f_{\hat{c}})$)
```

Figure 3.6: parse function used by the simulator $\mathcal{B}$.

(i) $\mathcal{B}$ provides a perfect simulation for the output send to $\mathcal{A}$.

(ii) If the trace $\mathbb{T}_\mathcal{A}$ of $\mathcal{A}$'s queries is valid (i.e., $\mathsf{valid}'(\mathbb{T}_\mathcal{A}) = \mathsf{true}$), then the trace $\mathbb{T}_\mathcal{B}$ of $\mathcal{B}$'s queries is valid (i.e., $\mathsf{valid}_2(\mathbb{T}_\mathcal{B}) = \mathsf{true}$).

(iii) If a query "parse c" of $\mathcal{A}$ results in a non-DY term, $\mathcal{A}$ wins in the real game and in the simulation.

(iv) If a collision occurs, the simulation and Game 0 behave equivalently or the simulator $\mathcal{B}$ wins its game.

- Proof of (i): We observe that the calls to TMs M_1 and M_2 in Game 0 and in Game 1 coincide. Hence we find a bijection between the used random coins. For parsing we basically decompose the library from Game 0 into the part belonging to $\mathcal{I}_1$ and the part belonging to $\mathcal{I}_2$. Since the open functions are type safe applying them to the corresponding parts will yield the same behavior in both games.

- Proof of (ii): For the "init T, H" query, the simulator $\mathcal{B}$ cannot use $\mathsf{generate}_\mathcal{B}$ yet. However, due to requirement (2) for valid in this lemma, $\mathcal{B}$ can split the query into two disjoint parts for $\mathcal{I}_1$ and $\mathcal{I}_2$. Furthermore, we check that the additional requirements for valid capture the additional queries introduced by $\mathsf{generate}_\mathcal{B}$: Since $\mathsf{generate}_\mathcal{B}$ has to learn the bitstrings for terms from $\mathcal{M}_2$, it cannot use silent generate queries. The trace remains valid due to requirement (1). Additionally requirement (3) allows $\mathsf{generate}_\mathcal{B}$ to query the bitstrings for subterms that do not contain function symbols from Σ_1.

- Proof of (*iii*): This holds since (*i*) holds and the DY-ness check in the simulation (Game 1) is identical to the one in the real game (Game 0).

- Proof of (*iv*): First, note that all function symbols that $(M_1\ f)(c_1, \dots, c_n; r) \neq (M_2\ f')(c'_1, \dots, c'_m; r')$ for all function symbols $f \in \Sigma_1$, $g \in \Sigma_2$ and all bitstrings $c_1, \dots, c_n, r, c'_1, \dots, c'_m, r'$ of proper types. This holds since f and g cannot be of basetype (due to our requirements for symbolic models) and requirement (i) for composable implementation guarantees that the domains of non-basetype types are disjoint. Analogously, collisions with the supplementary functions cannot occur. Hence every collision is either a collision in the domain $[\![\mathcal{T}_1]\!]$ of $\mathcal{I}_1$ or in the domain $[\![\mathcal{T}_2]\!]$ of $\mathcal{I}_2$. In the first case, the simulation behaves like the real game. In the second case the simulation wins the collision freeness game for $\mathcal{I}_2$ (which may only happen with negligible probability since $\mathcal{I}_2$ is collision free).

This concludes the proof of our claim that Game 0 and Game 1 are indistinguishable.

Game 2

Analogously to the previous step, we additionally abort only for collisions of bitstrings from $[\![\mathcal{T}_2]\!]$, i.e., we replace

```
if $c \in \mathrm{dom}(L) \cap [\![\mathcal{T}_1]\!]$ then
    exit game with return value 1 (collision)
```

with

```
if $c \in \mathrm{dom}(L) \cap [\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!]$ then
    exit game with return value 1 (collision)
```

in the $\mathsf{generate}$ function. Note that $[\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!] = \emptyset$ by requirement (ii) for the composition of symbolic models and requirement (i) for the composition of implementations. Hence this game will never abort due to collisions and is equivalent to $\mathsf{DS}_{\mathcal{M}' \cup \mathcal{M}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}']\!]), \mathcal{I}' \cup \mathcal{I}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}']\!]), \mathcal{A}}(\eta)$.

The proof that Game 2 and Game 1 are indistinguishable works exactly like the proof of the indistinguishability of Game 1 and Game 0.

Since Game 2 and Game 0 are indistinguishable, $\mathcal{I}'$ is collision-free. □

3.6 Deduction soundness

In this section we recall the notion of deduction soundness of an implementation with respect to a symbolic model [48]. Informally, the definition considers an adversary that plays the following game against a challenger. The challenger maintains a mapping L between bitstrings and hybrid terms, as defined in Section 3.3. Recall that the such mappings are used to both generate bitstring interpretations for terms, and also to parse bitstrings as terms (Figures 3.1,3.2). Roughly, the adversary is allowed to request to see the interpretation of arbitrary terms, and also to see the result of the parsing function applied to arbitrary bitstrings. Throughout the execution the queries that the adversary makes need to satisfy a predicate valid (which is a parameter of the implementation). The goal of the adversary is to issue a parse request such that the result is a term, that is not deducible from the terms that he had queried in his generate requests: this illustrates the idea that the adversary, although operating with bitstrings, is restricting to only performing Dolev-Yao operations on the bitstrings that it receives.

The details of the game are in Figure 3.7. Our definition departs from the one of [48] in a few technical aspects. First, we introduce a query init which is used to "initialize" the execution by, for example, generating (and corrupting) keys. The introduction of this query allows for a clearer separation between the phases where keys are created and where they are used, and allows to simplify and clarify what are valid interactions between the adversary and the game.

Secondly, we also allow the adversary to issue sgenerate requests: these are generate requests except that the resulting bitstring is not returned to the adversary. These requests are a technical necessity that help in later simulations, and only strengthen the adversary.

Deduction soundness of an implementation $\mathcal{I}$ with respect to a symbolic model $\mathcal{M}$ for an implementation is defined by considering an adversary who plays the game sketched above against an implementation that mixes $\mathcal{I}$ with transparent functions provided by the adversary. To ensure uniform behavior on behalf of the adversary (e.g. ensure that the adversary does not provide a different set of transparent functions for each different security parameter), and also to satisfy other technical requirements like defining polynomial running time for mixed implementations, we introduce a notion of parametrized transparent functions/models.

Parametrization

A *parametrized transparent symbolic model* $\mathcal{M}_{\mathsf{tran}}(\nu)$ maps a bitstring ν (the parameter) to a transparent symbolic model. Analogously, a *parametrized transparent implementation* $\mathcal{I}_{\mathsf{tran}}(\nu)$ of $\mathcal{M}_{\mathsf{tran}}$ maps a bitstring ν (the parameters) to a transparent implementation ν where the length of ν is polynomial in the security parameter. We say that a parameter ν is *good* if $\mathcal{I}(\nu)$ is a transparent implementation of $\mathcal{M}_{\mathsf{tran}}(\nu)$ and meets the requirements of a good implementation (i.e., type-safe, randomness-safe, . . .) from Section 3.3.

Definition 43 (Deduction soundness)**.** *Let $\mathcal{M}$ be a symbolic model and $\mathcal{I}$ be an implementation of $\mathcal{M}$. We say that $\mathcal{I}$ is a* deduction sound implementation of $\mathcal{M}$ *if for all parametrized transparent symbolic models $\mathcal{M}_{\mathsf{tran}}(\nu)$ and for all parametrized transparent implementations $\mathcal{I}_{\mathsf{tran}}(\nu)$ of $\mathcal{M}_{\mathsf{tran}}$ that are composable with $\mathcal{M}$ and $\mathcal{I}$ (see requirements from Section 3.5) we have that*

$$\mathbb{P}\left[\mathsf{DS}_{\mathcal{M}\cup\mathcal{M}_{\mathsf{tran}}(\nu),\mathcal{I}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta) = 1\right]$$

is negligible for all PPT adversaries $\mathcal{A}$ sending only good parameters ν where DS *is the deduction soundness game defined in Figure 3.7. Note that $\mathcal{M} \cup \mathcal{M}_{\mathsf{tran}}(\nu)$ can be generically composed to a parametrized symbolic model $\mathcal{M}'(\nu)$ and parametrized implementation $\mathcal{I}(\nu)$ respectively.*

Collisions

The deduction soundness game from Figure 3.7 doesn't prevent collisions. I.e., a query leading to calls of the $\mathsf{generate}$ function could produce bitstrings that are already in the library and therefore overwrite a value $L(c)$ with a new one. Note that "parse c" requests can never lead to collisions due to the structure of the parse function (see Figure 3.2). Fortunately, we can use a collision-free variant of the deduction soundness game.

Lemma 59. *Let $\mathsf{DS}'_{\mathcal{M}(\nu),\mathcal{I}(\nu),\mathcal{A}}(\eta)$ be the deduction soundness game where we replace the* $\mathsf{generate}$ *function by the collision aware generate function from Figure 3.3. Then no PPT adversary $\mathcal{A}$ can distinguish $\mathsf{DS}_{\mathcal{M}(\nu),\mathcal{I}(\nu),\mathcal{A}}(\eta)$ from $\mathsf{DS}'_{\mathcal{M}(\nu),\mathcal{I}(\nu),\mathcal{A}}(\eta)$ with non-negligible probability. (Note that the transparent functions are already included in $\mathcal{M}(\nu)$ and $\mathcal{I}(\nu)$ here.)*

Proof. The only difference between DS and DS', and hence the only way to distinguish them, is to produce a collision. However, if collisions could be found with non-negligible probability, this would break the collision freeness of $\mathcal{I}(\nu)$ (we require that $\mathcal{I}(\nu)$ is a good implementation for all parameters ν in the definition of deduction soundness Definition 43). More specifically, we can use any adversary $\mathcal{A}$ that sends a parameter ν and can later distinguish $\mathsf{DS}'_{\mathcal{M}(\nu),\mathcal{I}(\nu),\mathcal{A}}(\eta)$ from $\mathsf{DS}_{\mathcal{M}(\nu),\mathcal{I}(\nu),\mathcal{A}}(\eta)$ to construct an adversary $\mathcal{B}$ that wins the collision freeness game for $\mathcal{I}(\nu)$ (according to Definition 41) whenever $\mathcal{A}$ can distinguish: All queries by $\mathcal{A}$ are forwarded by $\mathcal{B}$ to its own game. If $\mathcal{A}$ finds a collision, $\mathcal{B}$ wins. Otherwise $\mathcal{A}$ cannot distinguish. □

```
$\mathsf{DS}_{\mathcal{M}(\nu),\mathcal{I}(\nu),\mathcal{A}}(\eta)$:
    let $S := \emptyset$        (set of requested terms)
    let $L := \emptyset$        (library)
    let $\mathbb{T} := \emptyset$        (trace of queries)
    $\mathcal{R} \leftarrow \{0,1\}^*$    (random tape)

    Receive parameter $\nu$ from $\mathcal{A}$

    on request "init $T, H$" do
        add "init $T$" to $\mathbb{T}$
        if $\mathsf{valid}(\mathbb{T})$ then
            let $S := S \cup T$
            let $C := \emptyset$    (list of replies)
            for each $t \in T$ do
                let $(c, L) := \mathsf{generate}_{M,\mathcal{R}}(t, L)$
                let $C := C \cup \{c\}$
            for each $t \in H$ do
                let $(c, L) := \mathsf{generate}_{M,\mathcal{R}}(t, L)$
            send $C$ to $\mathcal{A}$
        else
            return 0 ($\mathcal{A}$ is invalid)

    on request "sgenerate $t$" do
        if $\mathsf{valid}(\mathbb{T}+$"sgenerate $t$") then
            let $(c, L) := \mathsf{generate}_{M,\mathcal{R}}(t, L)$

    on request "generate $t$" do
        add "generate $t$" to $\mathbb{T}$
        if $\mathsf{valid}(\mathbb{T})$ then
            let $S := S \cup \{t\}$
            let $(c, L) := \mathsf{generate}_{M,\mathcal{R}}(t, L)$
            send $c$ to $\mathcal{A}$
        else
            return 0 ($\mathcal{A}$ is invalid)

    on request "parse $c$" do
        let $(t, L) := \mathsf{parse}(c, L)$
        if $S \vdash_{\mathcal{D}} t$ then
            send $t$ to $\mathcal{A}$
        else
            return 1 ($\mathcal{A}$ produced non-Dolev-Yao term)
```

Figure 3.7: Game defining deduction soundness. Whenever $\mathsf{generate}_{M,\mathcal{R}}(t, L)$ is called, the requirements for t are checked (i.e., all subterms of t with adversarial labels must already be in L and t does not contain garbage symbols with honest labels) and 0 is returned if the check fails (i.e., the $\mathcal{A}$ is considered to be invalid).

3.7 Composition theorems

Our notion of deduction soundness enjoys the nice property of being easily extensible: if an implementation is deduction sound for a given symbolic model, it is possible to add other primitives, one by one, without having to prove deduction soundness, from scratch for the resulting set of primitives.

3.7.1 Public datastructures

An immediate observation with interesting implications is the following. Consider some symbolic model $\mathcal{M}$ with a deduction sound implementation $\mathcal{I}$. Now, extend $\mathcal{M}$ by a transparent symbolic model $\mathcal{M}_{\mathsf{tran}}$ and $\mathcal{I}$ by a transparent implementation $\mathcal{I}_{\mathsf{tran}}$ of $\mathcal{M}_{\mathsf{tran}}$. Then, the resulting implementation is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{tran}}$.

The intuition behind this result is simple: if $\mathcal{I}$ is sound in presence of arbitrary transparent functions with an implementation selected by the adversary, adding transparent functions with some fixed transparent implementation preserves soundness. This idea is formalized by the following theorem.

Theorem 4. *Let $\mathcal{M}$ be a symbolic model and $\mathcal{I}$ a deduction sound implementation of $\mathcal{M}$. Furthermore, let $\mathcal{M}_{\mathsf{tran}}$ be a transparent symbolic model and $\mathcal{I}_{\mathsf{tran}}$ a transparent implementation of $\mathcal{M}_{\mathsf{tran}}$. If $\mathcal{M}$ and $\mathcal{I}$ are composable with $\mathcal{M}_{\mathsf{tran}}$ and $\mathcal{I}_{\mathsf{tran}}$ (see Section 3.5), then $\mathcal{I} \cup \mathcal{I}_{\mathsf{tran}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{tran}}$.*

Proof. Let $\mathcal{A}$ be a ppt adversary that breaks the deduction soundness of $\mathcal{I} \cup \mathcal{I}_{\mathsf{tran}}$, i.e., by Definition 43 there is a transparent symbolic model $\mathcal{M}^{\mathcal{A}}_{\mathsf{tran}}$ with a transparent implementation $\mathcal{I}^{\mathcal{A}}_{\mathsf{tran}}$ such that

$$\mathbb{P}\left[\mathsf{DS}_{(\mathcal{M}\cup\mathcal{M}_{\mathsf{tran}})\cup\mathcal{M}^{\mathcal{A}}_{\mathsf{tran}},(\mathcal{I}\cup\mathcal{I}_{\mathsf{tran}})\cup\mathcal{I}^{\mathcal{A}}_{\mathsf{tran}},\mathcal{A}}(\eta) = 1\right]$$

is non-negligible. Then clearly for the transparent symbolic model $\mathcal{M}_{\mathsf{tran}} \cup \mathcal{M}^{\mathcal{A}}_{\mathsf{tran}}$ and the transparent implementation $\mathcal{I}_{\mathsf{tran}} \cup \mathcal{I}^{\mathcal{A}}_{\mathsf{tran}}$ this adversary also breaks the deduction soundness of $\mathcal{I}$, i.e.,

$$\mathbb{P}\left[\mathsf{DS}_{\mathcal{M}\cup(\mathcal{M}_{\mathsf{tran}}\cup\mathcal{M}^{\mathcal{A}}_{\mathsf{tran}}),\mathcal{I}\cup(\mathcal{I}_{\mathsf{tran}}\cup\mathcal{I}^{\mathcal{A}}_{\mathsf{tran}}),\mathcal{A}}(\eta) = 1\right]$$

is non-negligible. Since $\mathcal{I}$ is a deduction sound implementation by requirement this concludes our proof. □

3.7.2 Public key encryption

In this section we define a symbolic model $\mathcal{M}_{\mathsf{PKE}}$ for public key encryption and a corresponding implementation $\mathcal{I}_{\mathsf{PKE}}$ based on a public key encryption scheme (PKE.KeyGen, PKE.Enc, PKE.Dec). We show that composition of $\mathcal{M}_{\mathsf{PKE}}$ and $\mathcal{I}_{\mathsf{PKE}}$ with any symbolic model $\mathcal{M}$ comprising a deduction sound implementation $\mathcal{I}$ preserves this property for the resulting implementation, i.e., $\mathcal{I} \cup \mathcal{I}_{\mathsf{PKE}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{PKE}}$. This result was already stated in [48]. However, since the definition of deduction soundness as well as other parts of the framework (e.g., parse function) were updated, we present an updated proof here.

3.7.2.1 Computational preliminaries

Definition 44 (Public-key encryption scheme)**.** *A* public-key encryption scheme *(PKE scheme) is a triple of algorithms* (PKE.KeyGen, PKE.Enc, PKE.Dec)*.*

The probabilistic key generation algorithm PKE.KeyGen *takes an encoding of the security parameter and some randomness as inputs and generates a pair* (ek, dk) *containing the* encryption key *ek and the* decryption key *dk.*

The probabilistic encryption algorithm PKE.Enc *takes three arguments: an encryption key ek, the message* $m \in \{0,1\}^*$*, and some randomness* $r \in \{0,1\}^\eta$*. It computes a* ciphertext $c := \mathsf{PKE.Enc}(ek, m; r)$*.*[3]

The decryption algorithm PKE.Dec *takes a decryption key and a ciphertext as inputs and returns a value from* $\{0,1\}^* \cup \{\bot\}$*. We require* perfect correctness*, i.e.,*

$$\mathsf{PKE.Dec}(dk, \mathsf{PKE.Enc}(ek, m; r)) = m$$

for all $r \leftarrow \{0,1\}, m \in \{0,1\}^*$ *and* $(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\eta)$*.*

Definition 45 (IND-CCA security of PKE schemes)**.** *A PKE scheme* (PKE.KeyGen, PKE.Enc, PKE.Dec) *is* IND-CCA secure *if for all PPT adversaries* $\mathcal{A}$ *the probability*

$$\mathbb{P}\left[\mathit{IND\text{-}CCA\text{-}PKE}_{\mathcal{A}}^{(\mathsf{PKE.KeyGen},\mathsf{PKE.Enc},\mathsf{PKE.Dec})}(\eta) = 1\right] - \frac{1}{2}$$

is negligible for the IND-CCA game from Figure 3.8 (this game resembles a multi-user version of standard IND-CCA security).

3.7.2.2 Symbolic model

We first define the symbolic model $(\mathcal{T}_{\mathsf{PKE}}, \preccurlyeq_{\mathsf{PKE}}, \Sigma_{\mathsf{PKE}}, \mathcal{D}_{\mathsf{PKE}})$ for public key encryption. The signature Σ_{PKE} features the following function symbols

$$\begin{aligned} dk_x &: \tau_{\mathsf{PKE}}^{\mathsf{dk}_x} \\ ek_x &: \tau_{\mathsf{PKE}}^{\mathsf{dk}_x} \to \tau_{\mathsf{PKE}}^{\mathsf{ek}_x} \\ enc_x &: \tau_{\mathsf{PKE}}^{\mathsf{ek}_x} \times \top \to \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}} \end{aligned}$$

for $x \in \{h, c\}$. The randomized function dk_h of arity $\tau_{\mathsf{PKE}}^{\mathsf{dk}_h}$ returns an honest decryption key. The deterministic function ek_h of arity $\tau_{\mathsf{PKE}}^{\mathsf{dk}_h} \to \tau_{\mathsf{PKE}}^{\mathsf{ek}_h}$ derives an honest encryption key from an honest decryption key. Analogously for corrupted decryption keys (dk_h) and corrupted encryption keys (ek_c). The randomized function enc_h for honest encryptions has arity $\tau_{\mathsf{PKE}}^{\mathsf{ek}_h} \times \top \to \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}}$ (enc_c analogously). As above we sometimes write ek_x, dk_x, enc_x for $x \in \{h, c\}$ to reference the honest and the corrupted variants of the functions comprehensively. By abuse of notation, we will often write $ek_x^l()$ instead of $ek_x(dk_x^l())$ where $l \in \mathsf{labelsH}$. To complete the formal definition we set

$$\mathcal{T}_{\mathsf{PKE}} := \{\top, \tau_{\mathsf{PKE}}^{\mathsf{dk}_x}, \tau_{\mathsf{PKE}}^{\mathsf{ek}_x}, \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}}\}$$

All introduced types are direct subtypes of the base type $\top$ (this defines $\preccurlyeq_{\mathsf{PKE}}$). The deduction system captures the security of public key encryption

$$\mathcal{D}_{\mathsf{PKE}} := \left\{ \begin{array}{cc} \multicolumn{2}{c}{\frac{ek_x^l() \quad m}{enc_x^{la}(ek_x^l(),m)},} \\ \\ \frac{enc_h^{la}(ek_h^l(),m)}{m}, & \frac{enc_c^{\hat{l}}(ek_c^l(),m)}{m} \end{array} \right\}$$

[3] Since the message m is of basetype in symbolic model given below, we require a scheme with message space $\{0,1\}^*$.

```
$\mathsf{IND\text{-}CCA\text{-}PKE}_{\mathcal{A}}^{(\mathsf{PKE.KeyGen},\mathsf{PKE.Enc},\mathsf{PKE.Dec})}(\eta)$:
    $b \leftarrow \{0,1\}$
    $\mathsf{oracles} := \emptyset$

    on request “$\mathsf{new\ oracle}$” do
        let $r \leftarrow \{0,1\}^{\eta}$
        let $(ek, dk) := \mathsf{PKE.KeyGen}(1^{\eta}, r)$
        add $ek$ to $\mathsf{oracles}$
        let $\mathsf{ciphers}_{ek} := \emptyset$
        send $ek$ to $\mathcal{A}$

    on request “$\mathcal{O}^{enc}_{ek}(m)$” do
        if $ek \notin \mathsf{oracles}$ then
            send $\perp$ to $\mathcal{A}$
        else
            let $r \leftarrow \{0,1\}^{\eta}$
            if $b = 0$ then
                let $c := \mathsf{PKE.Enc}(ek, 0^{|m|}, r)$
                add $(c, m)$ to $\mathsf{ciphers}_{ek}$
        else
            send $\mathsf{PKE.Enc}(ek, m, r)$ to $\mathcal{A}$

    on request “$\mathcal{O}^{dec}_{ek}(c)$” do
        if $ek \notin \mathsf{oracles}$ then
            send $\perp$ to $\mathcal{A}$
        else
            if $b = 0$ and $(c, m) \in \mathsf{ciphers}_{ek}$ for some $m$ then
                send $m$ to $\mathcal{A}$
            else
                let $dk$ be the decryption key for $ek$
                send $\mathsf{PKE.Dec}(dk, c)$ to $\mathcal{A}$

    on request “$\mathsf{guess}\ b'$” do
        if $b = b'$ then return 1 else return 0
```

Figure 3.8: The multi-user IND-CCA game for a public key encryption scheme (PKE.KeyGen, PKE.Enc, PKE.Dec).

These rules are valid for arbitrary labels $l, \hat{l} \in \mathsf{labels}$ and adversarial labels $l_a \in \mathsf{labelsA}$. Read from top left to bottom right the following intuitions back up the rules:

- The adversary can use any honestly generated key to encrypt some term m.
- The adversary knows the message contained in any adversarial encryption.
- The adversary knows the message contained in any encryption under a corrupted key.

Since we only allow for static corruption we do not need a rule $\frac{enc^{\hat{l}}_h(ek^l_h(),m)}{ek^l_h()}$ although we are going to attach the encryption key to the ciphertext in the implementation (all encryption keys are going to be part of the response to the "$\mathsf{init}\ T, H$" query by the adversary).

3.7.2.3 Implementation

We now give a concrete implementation $\mathcal{I}_{\mathsf{PKE}}$ for public key encryption. The implementation uses some IND-CCA secure public key encryption scheme ($\mathsf{PKE.KeyGen}$, $\mathsf{PKE.Enc}$, $\mathsf{PKE.Dec}$). To prevent collisions of ciphertexts, we require that we have $\mathsf{PKE.Enc}(ek, m, r) = \mathsf{PKE.Enc}(ek, m', r')$ only with negligible probability for bitstrings m, m', r given by the adversary and r' uniformly chosen honest randomness. Many PKE schemes meet this requirement directly, e.g., all committing schemes. Furthermore, it is always possible to extend the output of $\mathsf{PKE.Enc}$ with a nonce to prevent these collisions. The computable interpretations of, dk_x, ek_x, enc_x (for $x \in \{h, c\}$) are as follows:

- $(M_{\mathsf{PKE}}\ dk_x)(r)$: Let $(ek, dk) := \mathsf{PKE.KeyGen}(1^\eta; r)$. Return $\langle ek, dk, \tau^{\mathsf{dk}_x}_{\mathsf{PKE}} \rangle$.
- $(M_{\mathsf{PKE}}\ ek_x)(\hat{dk})$: Parse $\hat{dk}$ as $\langle ek, dk, \tau^{\mathsf{dk}_x}_{\mathsf{PKE}} \rangle$. Return $\langle ek, \tau^{\mathsf{ek}_x}_{\mathsf{PKE}} \rangle$.
- $(M_{\mathsf{PKE}}\ enc_x)(\hat{ek}, m; r)$: Parse $\hat{ek}$ as $\langle ek, \tau^{\mathsf{ek}_x}_{\mathsf{PKE}} \rangle$. Let $c := \mathsf{PKE.Enc}(ek, m; r)$ and return $\langle c, ek, \tau^{\mathsf{ciphertext}}_{\mathsf{PKE}} \rangle$.

The $\mathsf{valid}_{\mathsf{PKE}}$ predicate

The predicate $\mathsf{valid}_{\mathsf{PKE}}$ guarantees, that all keys that may be used by the adversary later are generated during initialization (i.e., with the init query). We only allow static corruption of keys, i.e., the adversary has to decide which keys are honest and which are corrupted at this stage. Keys may only be used for encryption and decryption. This implicitly prevents key cycles. More formally, based on the current trace $\mathbb{T}$ of all parse and generate requests of the adversary, the predicate $\mathsf{valid}_{\mathsf{PKE}}$ returns true only if the following conditions hold:

1. The trace starts with a query "$\mathsf{init}\ T, H$" (T resp. H may be the empty list). There are no further init queries.

2. The adversary may only generate keys in the init query. Concretely, this is guaranteed by the following rules:

   a) For the query "$\mathsf{init}\ T, H$", the function symbols ek_x and dk_x may only occur in a term $t \in T$ (i.e., not as subterms of other terms) of one of the two following types (for $l \in \mathsf{labelsH}$):

```
$\mathsf{open}_{\mathsf{PKE}}(c, L)$
    if $c \in [\![\mathcal{T}_{\mathsf{PKE}}]\!] \cap \mathrm{dom}(L)$ then
        return $(c, L(c))$
    else if $c = \langle dk, \tau_{\mathsf{PKE}}^{\mathsf{dk}_x} \rangle$ then
        return $(c, g^{l(c)}_{\tau_{\mathsf{PKE}}^{\mathsf{dk}_x}})$
    else if $c = \langle ek, \tau_{\mathsf{PKE}}^{\mathsf{ek}_x} \rangle$ then
        if $\hat{dk} \in \mathrm{dom}(L)$ s.t. $\hat{dk} = \langle ek, dk, \tau_{\mathsf{PKE}}^{\mathsf{dk}_x} \rangle$ then
            return $(c, ek_x(\hat{dk}))$
        else
            return $(c, g^{l(c)}_{\tau_{\mathsf{PKE}}^{\mathsf{ek}_x}})$
    else if c $= \langle c', ek, \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}} \rangle$ and $(\langle ek, \tau_{\mathsf{PKE}}^{\mathsf{ek}_x} \rangle, ek_x^l(\hat{dk})) \in L$ then
        parse $\hat{dk}$ as $\langle ek, dk, \tau_{\mathsf{PKE}}^{\mathsf{dk}_x} \rangle$
        let $m := \mathsf{PKE.Dec}(dk, c')$
        if $m = \bot$ then
            return $(c, g^{l(c)}_{\tau_{\mathsf{PKE}}^{\mathsf{ciphertext}}})$
        else
            return $(c, enc_x^{l(c)}(ek, m))$
    else
        return $(c, g_{\top}^{l(c)})$
```

Figure 3.9: Open function for public key encryption.

- $t = ek_h(dk_h^l())$ (to generate an honest encryption key)
- $t = dk_c^l()$ (to generate a corrupted encryption key)

Any label l for $dk_x^l()$ must be unique in T.

b) Any occurrence of $ek_x^l()$ or $dk_x^l()$ in a generate query must have occurred in the init query. $dk_x^l()$ may only occur as argument for ek_x. $ek_x^l()$ may only occur as first argument for enc_x.

3. The adversary must not use the function symbols for encryption (enc_h, enc_c) in the init query.

Checking the implementation

We first observe that $\mathcal{I}_{\mathsf{PKE}}$ is collision-free (Definition 41): Basically, collisions for keys can only occur with negligible probability since they break the security of the scheme (which is IND-CCA secure). We prevent collisions of ciphertexts by the requirements stated above. Furthermore, it is easy to see that $\mathsf{open}_{\mathsf{PKE}}$ meets the requirements of Definition 42 and that $\mathsf{valid}_{\mathsf{PKE}}$ meets the requirements for valid functions.

3.7.2.4 PKE composability

Theorem 5. *Let $\mathcal{M}$ be a symbolic model and $\mathcal{I}$ a deduction sound implementation of $\mathcal{M}$. If $(\mathcal{M}_{\mathsf{PKE}}, \mathcal{I}_{\mathsf{PKE}})$ and $(\mathcal{M},\mathcal{I})$ are compatible (see requirements in Section 3.5) and the PKE scheme* (PKE.KeyGen, PKE.Enc, PKE.Dec) *is IND-CCA and INT-CTXT secure, then $\mathcal{I} \cup \mathcal{I}_{\mathsf{PKE}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{PKE}}$.*

Proof. Let $\mathcal{A}$ be a PPT adversary and $\mathcal{M}_{\mathsf{tran}}(\nu)$ a parametrized transparent symbolic model with a corresponding parametrized implementation $\mathcal{I}_{\mathsf{tran}}(\nu)$ such that $\mathcal{M}_{\mathsf{tran}}(\nu)$ and $\mathcal{I}_{\mathsf{tran}}(\nu)$ are composable with $\mathcal{M}\cup\mathcal{M}_{\mathsf{PKE}}$ and $\mathcal{I}\cup\mathcal{I}_{\mathsf{PKE}}$ (see requirements in Section 3.5) for ν sent by $\mathcal{A}$. We have to show that $\mathcal{A}$ can win the deduction soundness game $\mathsf{DS}_{(\mathcal{M}\cup\mathcal{M}_{\mathsf{PKE}})\cup\mathcal{M}_{\mathsf{tran}}(\nu),(\mathcal{I}\cup\mathcal{I}_{\mathsf{PKE}})\cup\mathcal{I}_{\mathsf{tran}}(\nu)}(\eta)$ only with negligible probability.

We first explain the basic idea behind this proof. To win the deduction soundness game, $\mathcal{A}$ has to provide a bitstring corresponding to a term t that it does not "know" symbolically, i.e., $\mathcal{A}$ cannot deduce t from the terms it generated previously. Intuitively, by adding public key encryption, exactly one additional opportunity to learn something about such a term is created: $\mathcal{A}$ can retrieve honestly generated encryptions of t under honest encryption keys. In all other encryption scenarios, $\mathcal{A}$ knows the message by the rules of $\mathcal{D}_{\mathsf{PKE}}$. The strategy of this proof consists of two steps: First, we replace honestly generated encryptions of terms by encryptions of 0-bitstrings (using the IND-CCA security of the encryption scheme). Hence the adversary cannot learn anything about the corresponding clear texts (except their length) which eliminates the additionally opportunity for $\mathcal{A}$. Second, we show that encryption can be simulated by transparent functions. Thus, any other way for $\mathcal{A}$ to come up with non-DY terms leads to a non-DY request in the deduction soundness game for $\mathcal{M}$ and $\mathcal{I}$. Since $\mathcal{I}$ is deduction sound by assumption, this concludes our proof.

Concretely, we proof this with a sequence of games.

Game 0

In Game 0 $\mathcal{A}$ plays the original deduction soundness game $\mathsf{DS}_{(\mathcal{M}\cup\mathcal{M}_{\mathsf{PKE}})\cup\mathcal{M}_{\mathsf{tran}}(\nu),(\mathcal{I}\cup\mathcal{I}_{\mathsf{PKE}})\cup\mathcal{I}_{\mathsf{tran}}(\nu)}(\eta)$.

Game 1

In Game 1 we replace the $\mathsf{generate}$ function by the collision-aware generate function from Figure 3.3.

Claim: Game 0 and Game 1 are indistinguishable

Since $(\mathcal{I}\cup\mathcal{I}_{\mathsf{PKE}})\cup\mathcal{I}_{\mathsf{tran}}(\nu)$ is a collision-free implementation Game 0 and Game 1 are indistinguishable by Lemma 59.

Game 2

In Game 2 we deprive the adversary of any option to learn something from ciphertexts or about honest decryption keys. We replace the honest decryption keys in the library by random bitstrings and add the rule $\frac{ek_x(dk_x^l())}{dk_x^l()}$ to the deduction system. Intuitively, $\mathcal{A}$ doesn't notice the difference since the PKE scheme is IND-CCA secure. More concretely, instead of calling $(M_{\mathsf{PKE}}\ ek_h)(r)$, we pick $r \leftarrow \{0,1\}^\eta$, compute $(ek, dk) := \mathsf{PKE.KeyGen}(1^\eta, r)$, pick $r' \leftarrow \{0,1\}^{|dk|}$, remember dk as the decryption key corresponding to ek and use $\hat{dk} := \langle ek, r', \tau^{\mathsf{dk}_x}_{\mathsf{PKE}}\rangle$ as bitstring for $dk^l_h()$. Additionally, we change the parsing function such that it now uses the remembered corresponding keys for decryption instead of those in the library. Furthermore, to replace the ciphertexts created under honest keys by encryptions of 0, we replace the line

let $c := (M\ f)(c_1,\ldots,c_n;r)$

in the $\mathsf{generate}$ function with

```
if $t = enc^{\hat{l}}_h(ek^l_h(), m)$ then
    let $c := (M\ enc_h)(c_1, 0^{|c_2|}; r)$
else
    let $c := (M\ f)(c_1, \dots, c_n; r)$
```

Note that c_1 and c_2 resemble the bitstrings corresponding to the encryption key and the message respectively. Thus, the generation of bitstrings works exactly as in Game 1 if t is not an encryption under an honest key. Otherwise we generate all subterms as usual but replace the message by a bitstring of zeros of appropriate length.

Claim: Game 1 and Game 2 are indistinguishable

Let $\mathcal{A}$ be an adversary that can distinguish between playing Game 1 and Game 2. Then we can construct an adversary $\mathcal{B}$ that will win the oracle-based IND-CCA game for PKE scheme $(\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ from Figure 3.8.

Generating bitstrings. For each query "generate $ek_h(dk^l_h())$", instead of calling the generate function, $\mathcal{B}$ does the following: It requests a new oracle from the IND-CCA game and receives an encryption key ek as well as access to a corresponding encryption oracle, denoted $\mathcal{O}^{enc}_{ek}$, (which either encrypts the given messages or 0-bitstrings of the same lengths) and to a corresponding decryption oracle, denoted $\mathcal{O}^{dec}_{ek}$. $\mathcal{B}$ now picks a random bitstring dk such that $\hat{dk} := \langle ek, dk, \tau^{\mathsf{dk}_h}_{\mathsf{PKE}} \rangle \in [\![\tau^{\mathsf{dk}_h}_{\mathsf{PKE}}]\!]$ and adds $(\hat{dk}, dk^l_h())$ and $(\langle ek, \tau^{\mathsf{ek}_h}_{\mathsf{PKE}} \rangle, ek_h(\hat{dk}))$ to L. Note that by requirement (2) for $\mathsf{valid}_{\mathsf{PKE}}$ $\mathcal{A}$ will never learn the value of dk. All other types of generate requests are handled by calling the generate function.

To use the provided oracles to generate honest encryptions, $\mathcal{B}$ furthermore uses a modified generate function. Concretely, it replaces the line

```
let $c := (M\ f)(c_1, \dots, c_n; r)$
```

with

```
if $t = enc^{\hat{l}}_h(ek^l_h(), m)$ then
    let $c := \langle \mathcal{O}^{enc}_{c_1}(c_2), c_1, \tau^{\mathsf{ciphertext}}_{\mathsf{PKE}} \rangle$
else
    let $c := (M\ f)(c_1, \dots, c_n; r)$
```

Again, c_1 and c_2 resemble the bitstrings corresponding to the encryption key and the message respectively. Note that this function produces encryptions like the generate function in Game 1 if the oracle encrypts the given message and like the generate function in Game 2 otherwise, i.e., if the oracle encrypts a 0-bitstring.

Parsing bitstrings. $\mathcal{B}$ also has to modify the parsing function to deal with adversarial ciphertexts created with honest keys. More specifically, it removes the lines

```
parse $\hat{dk}$ as $\langle ek, dk, \tau^{\mathsf{dk}_x}_{\mathsf{PKE}} \rangle$
let $m := \mathsf{PKE.Dec}(dk, c')$
```

in the $\mathsf{open}_{\mathsf{PKE}}$ function from Figure 3.9 and adds

```
if $(\langle ek, \tau^{\mathsf{ek}_h}_{\mathsf{PKE}} \rangle, ek^l_h()) \in L$ (honest key) then
    let $m := \mathcal{O}^{dec}_{ek}(c')$
else
parse $\hat{dk}$ as $\langle ek, dk, \tau^{\mathsf{dk}_c}_{\mathsf{PKE}} \rangle$
    let $m := \mathsf{PKE.Dec}(dk, c')$
```

For an IND-CCA secure public key encryption scheme this function decrypts like the original $\mathsf{open}_{\mathsf{PKE}}$ function. $\mathcal{B}$ just uses the decryption oracle instead of the decryption key during the simulation. Additionally, if $\mathsf{open}_{\mathsf{PKE}}$ is called for a bitstring $c \in [\![\tau_{\mathsf{PKE}}^{\mathsf{dk}_h}]\!]$, $\mathcal{B}$ parses c as $\langle ek, dk, \tau_{\mathsf{PKE}}^{\mathsf{dk}_h}\rangle$. $\mathcal{B}$ then picks a random message $m \leftarrow \{0,1\}^\eta$ and computes $\mathsf{PKE.Dec}(dk, \mathcal{O}_{ek}^{enc}(m)) = x$. If $x \neq \bot$ and $x = m$, $\mathcal{B}$ sends "guess 1" to the IND-CCA game and wins with overwhelming probability.

Analysis. If the oracle produces real encryptions, $\mathcal{B}$ simulates Game 1 for $\mathcal{A}$. The only difference are the random values for honest decryption keys in the library. Those values are only used when parsing bitstrings. The difference can be detected by $\mathcal{A}$ if it guesses one of the random bitstrings (which can only happen with negligible probability) or if it parses a bitstring belonging to a honest decryption key in Game 1. However, in the latter case, $\mathcal{B}$ wins the IND-CCA game as described above. Hence the simulation is indistinguishable for $\mathcal{A}$ if the oracles produce real encryptions.

If the oracles produces encryptions of zero, $\mathcal{B}$ perfectly simulates Game 2 for $\mathcal{A}$. Hence, every correct guess of $\mathcal{A}$ on which game he is playing leads to a correct guess of $\mathcal{B}$ in the IND-CCA game. Therefore, $\mathcal{A}$ cannot distinguish Game 1 and Game 2.

Game 3

In Game 3 $\mathcal{A}$ interacts with an adversary $\mathcal{B}$ that plays the deduction soundness game for $\mathcal{M}$ and $\mathcal{I}$ and intuitively simulates Game 2 for $\mathcal{A}$. Basically, $\mathcal{B}$ uses transparent functions to add public key encryption to $\mathcal{M}$. We construct $\mathcal{B}$ as follows:

Transparent symbolic model for public key encryption. We first describe the parametrized transparent symbolic model $\mathcal{M}_{\mathsf{PKE}}^{\mathsf{tran}}(\nu)$ and the corresponding parametrized implementation $\mathcal{I}_{\mathsf{PKE}}^{\mathsf{tran}}(\nu)$ $\mathcal{B}$ will use to simulate $\mathcal{I}_{\mathsf{PKE}}$. We use the data types and subtype relation from $\mathcal{M}_{\mathsf{PKE}}$. ν is expected to be an encoding of a list of label-triple pairs $(l, (ek, dk, dk'))$ ($l \in \mathsf{labels}$) where the triple consist of a keypair ek, dk and an additional value dk' (used to represent honest decryption keys in the library). The signature $\Sigma_{\mathsf{PKE}}^{\mathsf{tran}}$ is the following:

- deterministic $f_{dk_x^l()}$ with $\mathrm{ar}(f_{dk_x^l()}) = \tau_{\mathsf{PKE}}^{\mathsf{dk}_x}$ for all labels $l \in \nu$
- deterministic $f_{ek_x^l()}$ with $\mathrm{ar}(f_{ek_x^l()}) = \tau_{\mathsf{PKE}}^{\mathsf{ek}_x}$ for all labels $l \in \nu$
- randomized $f_{enc_h(ek_h^l(),0^\ell)}$ with $\mathrm{ar}(f_{enc_h(ek_h^l(),0^\ell)}) = \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}}$ for all $\ell \in \mathbb{N}$, $l \in \nu$
- randomized $f_{enc_h(ek_h^l(),\cdot)}$ with $\mathrm{ar}(f_{enc_h(ek_h^l(),\cdot)}) = \top \rightarrow \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}}$ for all $l \in \nu$
- randomized $f_{enc_c((ek_c^l(),\cdot}$ with $\mathrm{ar}(f_{enc_c((ek_c^l(),\cdot}) = \top \rightarrow \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}}$ for all $l \in \nu$

We specify a parametrized implementation $\mathcal{I}_{\mathsf{PKE}}^{\mathsf{tran}}(\nu)$ for $\mathcal{M}_{\mathsf{PKE}}^{\mathsf{tran}}$ and for each $(l, (ek, dk, dk')) \in \nu$ as follows:

- $(M_{\mathsf{PKE}}^{\mathsf{tran}}\ f_{dk_x^l()})()$ returns $\langle ek, dk', \tau_{\mathsf{PKE}}^{\mathsf{dk}_x}\rangle$.
- $(M_{\mathsf{PKE}}^{\mathsf{tran}}\ f_{ek_x^l()})()$ returns $\langle ek, \tau_{\mathsf{PKE}}^{\mathsf{ek}_x}\rangle$.
- $(M_{\mathsf{PKE}}^{\mathsf{tran}}\ f_{enc_h(ek_h^l(),0^\ell)})(r)$ returns $(M_{\mathsf{PKE}}\ enc_h)(\langle ek, \tau_{\mathsf{PKE}}^{\mathsf{ek}_x}\rangle, 0^\ell; r)$.
- $(M_{\mathsf{PKE}}^{\mathsf{tran}}\ f_{enc_h(ek_h^l(),\cdot)})(m; r)$ returns $(M_{\mathsf{PKE}}\ enc_h)(\langle ek, \tau_{\mathsf{PKE}}^{\mathsf{ek}_x}\rangle, m; r)$.
- $(M_{\mathsf{PKE}}^{\mathsf{tran}}\ f_{enc_c((ek_c^l(),\cdot})(m; r)$ returns $(M_{\mathsf{PKE}}\ enc_c)(\langle ek, \tau_{\mathsf{PKE}}^{\mathsf{ek}_x}\rangle, m; r)$.

```
($M^{\mathsf{tran}}_{\mathsf{PKE}}$ func)($b$):
    if $b = \langle ek, dk', \tau^{\mathsf{dk}_x}_{\mathsf{PKE}}\rangle$ for some $(l, (ek, dk, dk')) \in \nu$ then
        return $f_{dk^l_x()}$
    else if $b = \langle ek, \tau^{\mathsf{ek}_x}_{\mathsf{PKE}}\rangle$ for some $(l, (ek, dk, dk')) \in \nu$ then
        return $f_{ek^l_x()}$
    else if $b \in [\![\tau^{\mathsf{ciphertext}}_{\mathsf{PKE}}]\!]$ then
        parse $b$ as $\langle c, ek, \tau^{\mathsf{ciphertext}}_{\mathsf{PKE}}\rangle$
        if there is $(l, (ek, dk, r')) \in \nu$ for some $l, dk$ then
            let $m := \mathsf{PKE.Dec}(dk, c)$
            if $m \neq \perp$ then
                if $l$ belongs to an honest key then
                    return $f_{enc_h(ek^l_h(),\cdot)}$
                else
                    return $f_{enc_c((ek^l_c(),\cdot}$
        return $\perp$
```

For b with $(M^{\mathsf{tran}}_{\mathsf{PKE}}\ \mathsf{func})(b) = f_{enc_h(ek^l_h(),\cdot)}$ we have $(l, (ek, dk, dk')) \in \nu$ with $\mathsf{PKE.Dec}(dk, c) =: m \neq \perp$ where $b = \langle c, ek, \tau^{\mathsf{ciphertext}}_{\mathsf{PKE}}\rangle$ and define $(M^{\mathsf{tran}}_{\mathsf{PKE}}\ f_{enc_h(ek^l_h(),\cdot)}1)(b) := m$. Analogously for $(M^{\mathsf{tran}}_{\mathsf{PKE}}\ \mathsf{func})(b) = f_{enc_c((ek^l_c(),\cdot}$.

Convert terms. Adversary $\mathcal{A}$ uses the function symbols of the original symbolic model $\mathcal{M}_{\mathsf{PKE}}$. Hence $\mathcal{B}$ needs to map these symbols to the corresponding transparent functions. Towards this goal we introduce the function convert as follows (the first matching rule is applied):

- $\mathsf{convert}(f^l(t_1, \ldots, t_n)) = f^l(\mathsf{convert}(t_1), \ldots, \mathsf{convert}(t_n))$ for all $f \notin \Sigma_{\mathsf{PKE}}$.
- $\mathsf{convert}(ek_x(dk^l_x())) = f_{ek^l_x()}$
- $\mathsf{convert}(dk^l_x()) = f_{dk^l_x()}$
- $\mathsf{convert}(enc^{\hat{l}}_h(ek^l_h(), m)) = f^{\hat{l}(m)}_{enc_h(ek^l_h(),0^\ell)}()$ if $\hat{l} \in \mathsf{labelsH}$
- $\mathsf{convert}(enc^{\hat{l}}_h(ek^l_h(), m)) = f^{\hat{l}}_{enc_h(ek^l_h(),\cdot)}(\mathsf{convert}(m))$ if $\hat{l} \in \mathsf{labelsA}$
- $\mathsf{convert}(enc^{\hat{l}}_c(ek^l_c(), m)) = f^{\hat{l}}_{enc_c((ek^l_c(),\cdot}(\mathsf{convert}(m))$

For a list of terms T we define $\mathsf{convert}(T) := \{\mathsf{convert}(t) : t \in T\}$.

$\mathcal{B}$ simulates Game 2 for $\mathcal{A}$ while playing

$$\mathsf{DS}_{\mathcal{M}\cup(\mathcal{M}^{\mathsf{tran}}_{\mathsf{PKE}}(\nu)\cup\mathcal{M}_{\mathsf{tran}}(\nu')),\mathcal{I}(\cup\mathcal{I}^{\mathsf{tran}}_{\mathsf{PKE}}(\nu)\cup\mathcal{I}_{\mathsf{tran}}(\nu'))}(\eta)$$

Note that we can generically compose $\mathcal{M}^{\mathsf{tran}}_{\mathsf{PKE}}(\nu) \cup \mathcal{M}_{\mathsf{tran}}(\nu')$ to one parametrized transparent model $\mathcal{M}'_{\mathsf{tran}}(\nu||\nu')$ since ν and ν' must be good (analogously for the implementation). However, for the sake of clarity, we keep them apart to distinguish the transparent functions (and parameter) provided by $\mathcal{A}$ from the additional transparent functions introduced by $\mathcal{B}$. Next we describe how $\mathcal{B}$ deals with the queries received from $\mathcal{A}$.

init query. $\mathcal{B}$ receives a lists of terms T, H from $\mathcal{A}$. Initially, $\mathcal{B}$ sets $\nu := \emptyset$. For each occurrence of $dk^l_x() \in T$ $\mathcal{B}$ then picks a nonce $r \leftarrow \{0,1\}^\eta$ and generates a keypair $(ek, dk) := \mathsf{PKE.KeyGen}(1^\eta, r)$. If $x = h$, $\mathcal{B}$ picks $dk' \leftarrow \{0,1\}^{|dk|}$. Otherwise $\mathcal{B}$

sets $dk' = dk$. (dk' represents the decryption key in the library and should be a fresh random value for honest decryption keys.) $\mathcal{B}$ then adds $(l, (ek, dk, dk'))$ to ν. Finally, $\mathcal{B}$ sends $\nu' || \nu$ to its game and subsequently queries "$\mathsf{init}\ \mathsf{convert}(T), \mathsf{convert}(H)$". Afterwards, $\mathcal{B}$ queries "$\mathsf{sgenerate}\ dk^l_h()$" for each $dk^l_h() \in T$.

generate queries. For each request "$\mathsf{generate}\ t$" $\mathcal{B}$ adds "$\mathsf{generate}\ t$" to $\mathbb{T}$. Then $\mathcal{B}$ sends "$\mathsf{generate}\ \mathsf{convert}(t)$" to its game and relays the response to $\mathcal{A}$. For each subterm $t' \in st(t)$ that is an honest encryption, i.e., $t' = enc^{\hat{l}}_h(ek^l_h(), m)$ $\mathcal{B}$ additionally sends "$\mathsf{sgenerate}\ \mathsf{convert}(m)$" to its game. Analogously for "$\mathsf{sgenerate}\ t$".

parse queries. For each request "$\mathsf{parse}\ c$" $\mathcal{B}$ sends "$\mathsf{parse}\ c$" to its game and receives a term t. $\mathcal{B}$ sends $\mathsf{convert}^{-1}(t)$ to $\mathcal{A}$.

Claim: Game 2 and Game 3 are indistinguishable

We show that $\mathcal{B}$, while playing the game

$$\mathsf{DS}_{\mathcal{M}\cup(\mathcal{M}^{\mathsf{tran}}_{\mathsf{PKE}}(\nu)\cup\mathcal{M}_{\mathsf{tran}}(\nu')),\mathcal{I}(\cup\mathcal{I}^{\mathsf{tran}}_{\mathsf{PKE}}(\nu)\cup\mathcal{I}_{\mathsf{tran}}(\nu'))}(\eta),$$

perfectly simulates Game 2 for $\mathcal{A}$. First, we show that any valid trace produced by $\mathcal{A}$ in Game 2 leads to a valid trace by $\mathcal{B}$ (we say that these traces *correspond*). Then we show that every pair of valid corresponding traces leads to the same output for $\mathcal{A}$ by proving that a suitable invariant holds for the relation between the libraries in both settings.

Valid $\mathcal{A}$ leads to valid $\mathcal{B}$. First, we observe that any trace $\mathbb{T}_{\mathcal{A}}$ produced by $\mathcal{A}$ in Game 2 that is valid leads to a valid trace $\mathbb{T}_{\mathcal{B}}$ produced by $\mathcal{B}$: The application of $\mathsf{convert}$ to terms leads to variations the sense of requirement (i) for valid predicates. The additional $\mathsf{sgenerate}$ queries by $\mathcal{B}$ are valid by requirement (ii) for valid predicates. Furthermore, if a term t meets the requirement for the $\mathsf{generate}$ function in Game 2, $\mathsf{convert}(t)$ meets the requirements in the game $\mathcal{B}$ is playing.

Invariant. We still have to show that the output of the simulation matches the output of Game 2. First we observe that there is a bijection between the random coins used in Game 2 and the simulation. $\mathcal{B}$ uses the same amount of randomness to generate the keypairs as Game 2 does. All further uses of random coins coincide. Therefore, we can w.l.o.g. assume that the same random coins are used in Game 2 and the simulation. We show that the following invariants holds for all valid traces produced by $\mathcal{A}$ and the corresponding trace produced by $\mathcal{B}$:

1. $\mathrm{dom}(L_{ext}) = \mathrm{dom}(L_{small})$
2. $\forall c \in \mathrm{dom}(L_{small}) : \mathsf{convert}^{-1}(L_{small}[[c]]) = L_{ext}[[c]]$

where L_{ext} is the library in Game 2 and L_{small} the library in the game $\mathcal{B}$ is playing (which we call the *small game* from now on).

Initially, we have $L_{ext} = L_{small} = \emptyset$ and the invariant holds obviously.

init T, H**.** According to requirements (2) and (3) for public key encryption (i.e., $\mathsf{valid}_{\mathsf{PKE}}$) we can distinguish the following three types of terms $t \in T \cup H$.

- $t = ek_h(dk^l_h())$, i.e., $\mathsf{convert}(t) = f_{ek^l_h()}$.
- $t = dk^l_c()$, i.e., $\mathsf{convert}(t) = f_{dk^l_c()}$.
- t doesn't contain function symbols from Σ_{PKE} and $\mathsf{convert}(t) = t$:

We observe that each initial term t that is a key generation in the extended game, $\mathcal{B}$ adds the corresponding converted term $\mathsf{convert}(t)$ to its init request. This corresponds to the key generation done in the extended game while the keys in the small game are hard coded in the transparent functions. After this step we have $\mathrm{dom}(L_{ext} \setminus \{(c, dk^l_h()) : c \in [\![\tau^{\mathsf{dk}_h}_{\mathsf{PKE}}]\!], l \in \mathsf{labelsH}\}) = \mathrm{dom}(L_{small})$ since the generated keys coincide but we do not add the honest decryption keys to the library in the simulation. This is gap is closed by the additional silent generate queries by $\mathcal{B}$. Then the invariants hold.

generate ***t*****.** Assume that our invariants (1) and (2) hold for libraries L_{ext} and L_{small}. Then, they still hold after a valid query "$\mathsf{generate}\ t$" by $\mathcal{A}$ has been processed. In the extended game we have $(c_{ext}, L'_{ext}) := \mathsf{generate}_{M_{ext},\mathcal{R}}(t, L_{ext})$. In the simulation, $\mathcal{B}$ sends "$\mathsf{generate}\ \mathsf{convert}(t)$" to the small game and we have $(c_{small}, L'_{small}) := \mathsf{generate}_{M_{small},\mathcal{R}}(\mathsf{convert}(t), L_{small})$ (and maybe additional calls to $\mathsf{generate}_{M_{small},\mathcal{R}}$ if t contains honestly generated encryptions using honest keys). We observe the following (where M_{ext} and M_{small} are the Turing machines in the extended and in the small game respectively):

- By requirement (2) for $\mathsf{valid}_{\mathsf{PKE}}$, M_{ext} is never called for ek_x, dk_x. Analogously for M_{small} and the transparent translations of keys.

- For an honest encryption subterm $t = enc^{\hat{l}}_h(ek^l_h(), m)$, we have a call $(c_{ext}, L'_{ext}) := \mathsf{generate}_{M_{ext},\mathcal{R}}(t, L_{ext})$ in the extended game and calls $(c_{small}, L'_{small}) := \mathsf{generate}_{M_{small},\mathcal{R}}(f^{\hat{l}}_{enc_h(ek^l_h(),0^\ell)}(m)(ek^l_h()), L_{small})$ and $(c'_{small}, L''_{small}) := \mathsf{generate}_{M_{small},\mathcal{R}}(m, L'_{small})$ in the small game (since $\mathcal{B}$ sends an additional query "$\mathsf{sgenerate}\ m$" to the small game). It is easy to see that the newly generated bitstrings coincide.

- For all other terms $\mathsf{convert}(t)$ only removes keys from t which are already part of the library. The rest of the term remains intact and hence the newly generated bitstrings coincide.

Our invariant hold for L'_{ext} and L'_{small}. Note that this implies $c_{ext} = c_{small}$ which is the response sent to $\mathcal{A}$ in both settings. Analogously for queries "$\mathsf{sgenerate}\ t$".

parse ***c*****.** Assume that our invariants (1) and (2) hold for libraries L_{ext} and L_{small}. Then, they still hold after a valid query "$\mathsf{parse}\ c$" by $\mathcal{A}$ has been processed. If $c \in \mathrm{dom}(L_{ext})$, nothing changes and due to invariant (2) the response to $\mathcal{A}$ is the same in Game 2 and Game 3. Otherwise, since the implementations in both games are type-safe (Definition 42) we can focus our analysis on opening bitstrings from $[\![\mathcal{T}_{\mathsf{PKE}}]\!]$. For those it is easy to check that the open function for opens them structurally the same way (modulo conversion) the function $\mathsf{open}_{\mathsf{PKE}}$ does.

Claim: If $\mathcal{A}$ wins, then $\mathcal{B}$ wins Game 3

Due to the invariants (1) and (2) from above, we know that whenever a bitstring c sent by $\mathcal{A}$ is parsed as a term t in Game 3, it is parsed as $\mathsf{convert}(t)$ in the small game. By checking the deduction systems of both games we see that if t is non-DY in Game 3, then $\mathsf{convert}(t)$ is non-DY in the small game. Since $\mathcal{I}$ is a deduction sound implementation of $\mathcal{M}$ $\mathcal{A}$ can win Game 3 only with negligible probability which concludes our proof. □

3.7.3 Signatures

In this section we show that any deduction sound implementation can be extended by a signature scheme. More precisely, composition works if we require a strongly EUF-CMA-secure signature scheme and enforce static corruption. The result is again a deduction sound implementation.

3.7.3.1 Computational preliminaries

Definition 46 (signature scheme)**.** *A* signature scheme *is a triple of algorithms* $(\mathsf{SIG.KeyGen}, \mathsf{SIG.Sig}, \mathsf{SIG.Vfy})$*.*

The probabilistic key generation algorithm $\mathsf{SIG.KeyGen}$ *takes an encoding of the security parameter and some randomness as inputs and generates a pair* $(\mathit{vk}, \mathit{sk})$ *containing the* verification key vk *and the* signing key sk*.*

The probabilistic signing algorithm $\mathsf{SIG.Sig}$ *takes three arguments: a signing key* sk*, the message* $m \in \{0,1\}^*$*, and some randomness* $r \in \{0,1\}^\eta$*. It computes a* signature $\sigma := \mathsf{SIG.Sig}(\mathit{sk}, m; r)$*.*

The verification algorithm $\mathsf{SIG.Vfy}$ *takes a verification key* vk*, a signature* σ*, and a message* m *as inputs and returns a value from* $\{0,1\}$*. We require* perfect correctness*:*

$$\mathsf{SIG.Vfy}(\mathit{vk}, \mathsf{SIG.Sig}(\mathit{sk}, m; r), m) = 1$$

for all $r \leftarrow \{0,1\}, m \in \{0,1\}^*$ *and* $(\mathit{vk}, \mathit{sk}) \leftarrow \mathsf{SIG.KeyGen}(1^\eta)$*.*

Definition 47 (strong EUF-CMA security of signature schemes)**.** *A signature scheme* $(\mathsf{SIG.KeyGen}, \mathsf{SIG.Sig}, \mathsf{SIG.Vfy})$ *is* strongly EUF-CMA secure *if for all PPT adversaries* $\mathcal{A}$ *the probability*

$$\mathbb{P}\left[\mathit{EUF\text{-}CMA\text{-}SIG}_{\mathcal{A}}^{(\mathsf{SIG.KeyGen},\mathsf{SIG.Sig},\mathsf{SIG.Vfy})}(\eta) = 1\right]$$

is negligible for the EUF-CMA game from Figure 3.10. Note that, analogously to Figure 3.8, we can also define an equivalent multi-user version of strong EUF-CMA security.

```
$\mathsf{EUF\text{-}CMA\text{-}SIG}_{\mathcal{A}}^{(\mathsf{SIG.KeyGen},\mathsf{SIG.Sig},\mathsf{SIG.Vfy})}(\eta)$:
    let $(\mathit{vk}, \mathit{sk}) \leftarrow \mathsf{SIG.KeyGen}(1^\eta)$
    let $\mathsf{sigs} = \emptyset$
    send $\mathit{vk}$ to $\mathcal{A}$

    on request "$\mathsf{sign}\ m$" do
        let $\sigma \leftarrow \mathsf{SIG.Sig}(\mathit{sk}, m)$
        add $(\sigma, m)$ to $\mathsf{sigs}$
        send $\sigma$ to $\mathcal{A}$

    on request "$\mathsf{forge}\ \sigma^*, m^*$" do
        if $\mathsf{SIG.Vfy}(\mathit{vk}, \sigma^*, m^*) = 1$ and $(\sigma^*, m^*) \notin \mathsf{sigs}$ then return 1 else return 0
```

Figure 3.10: The strong EUF-CMA game for a signature scheme $(\mathsf{SIG.KeyGen}, \mathsf{SIG.Sig}, \mathsf{SIG.Vfy})$.

3.7.3.2 Symbolic model

We first define the symbolic model $(\mathcal{T}_{\mathsf{SIG}}, \preccurlyeq_{\mathsf{SIG}}, \Sigma_{\mathsf{SIG}}, \mathcal{D}_{\mathsf{SIG}})$ for signatures. The signature Σ_{SIG} features the following function symbols:

$$\begin{aligned} &sk : \tau^{\mathsf{sk}}_{\mathsf{SIG}} \\ &vk : \tau^{\mathsf{sk}}_{\mathsf{SIG}} \to \tau^{\mathsf{vk}}_{\mathsf{SIG}} \\ &sig : \tau^{\mathsf{sk}}_{\mathsf{SIG}} \times \top \to \tau^{\mathsf{sig}}_{\mathsf{SIG}} \end{aligned}$$

The randomized function sk of arity $\tau^{\mathsf{sk}}_{\mathsf{SIG}}$ returns a signing key. The deterministic function vk of arity $\tau^{\mathsf{sk}}_{\mathsf{SIG}} \to \tau^{\mathsf{vk}}_{\mathsf{SIG}}$ derives a verification key from a signing key. The randomized signing function sig has arity $\tau^{\mathsf{sk}}_{\mathsf{SIG}} \times \top \to \tau^{\mathsf{sig}}_{\mathsf{SIG}}$ and, given a signing key and a message of type $\top$, represents a signature of that message. To complete the formal definition we set the types

$$\mathcal{T}_{\mathsf{SIG}} := \{\top, \tau^{\mathsf{sk}}_{\mathsf{SIG}}, \tau^{\mathsf{vk}}_{\mathsf{SIG}}, \tau^{\mathsf{sig}}_{\mathsf{SIG}}\}$$

All introduced types are direct subtypes of the base type $\top$ (this defines $\preccurlyeq_{\mathsf{SIG}}$). The deduction system captures the security of signatures

$$\mathcal{D}_{\mathsf{SIG}} := \left\{ \begin{array}{cc} \multicolumn{2}{c}{\frac{sk^l()}{vk(sk^l())},} \\ \\ \frac{sig^{\hat{l}}(sk^l(),m)}{m}, & \frac{sk^l() \quad m}{sig^{l_a}(sk^l(),m)} \end{array} \right\}$$

These rules are valid for arbitrary labels $l, \hat{l} \in \mathsf{labels}$ and adversarial labels $l_a \in \mathsf{labelsA}$. Read from top left to bottom right the following intuitions back up the rules:

- The adversary can derive verification keys from signing keys.
- Signatures reveal the message that was signed.
- The adversary can use known signing keys to deduce signatures under those keys.

Although the verification key is going to be part of the computational implementation of a signatures, we don't need a rule $\frac{sig^{\hat{l}}(sk^l(),m)}{vk(sk^l)}$ since we enforce static corruption where adversary knows all verification keys.

3.7.3.3 Implementation

We now give a concrete implementation $\mathcal{I}_{\mathsf{SIG}}$ for signatures given a signature scheme $(\mathsf{SIG.KeyGen}, \mathsf{SIG.Sig}, \mathsf{SIG.Vfy})$. The computable interpretations of sk, vk, sig are as follows:

- $(M_{\mathsf{SIG}}\ sk)(r)$: Let $(vk, sk) := \mathsf{SIG.KeyGen}(1^\eta; r)$. Return $\langle vk, sk, \tau^{\mathsf{sk}}_{\mathsf{SIG}}\rangle$.
- $(M_{\mathsf{SIG}}\ vk)(\hat{sk}$: Parse $\hat{sk}$ as $\langle vk, sk, \tau^{\mathsf{sk}}_{\mathsf{SIG}}\rangle$. Return $\langle vk, \tau^{\mathsf{vk}}_{\mathsf{SIG}}\rangle$.
- $(M_{\mathsf{SIG}}\ sig)(\hat{sk}, m; r)$: Parse $\hat{sk}$ as $\langle vk, sk, \tau^{\mathsf{sk}}_{\mathsf{SIG}}\rangle$. Let $\sigma := \mathsf{SIG.Sig}(sk, m; r)$ and return $\langle \sigma, m, vk, \tau^{\mathsf{sig}}_{\mathsf{SIG}}\rangle$.

```
$\mathsf{open}_{\mathsf{SIG}}(c, L)$
    if $c \in [\![\mathcal{T}_{\mathsf{SIG}}]\!] \cap \mathrm{dom}(L)$ then
        return $(c, L(c))$
    else if $c = \langle sk, \tau_{\mathsf{SIG}}^{\mathsf{sk}} \rangle$ then
        return $(c, g_{\tau_{\mathsf{SIG}}^{\mathsf{sk}}}^{l(c)})$
    else if $c = \langle vk, \tau_{\mathsf{SIG}}^{\mathsf{vk}} \rangle$ then
        if $\hat{sk} \in \mathrm{dom}(L)$ s.t. $\hat{sk} = \langle vk, sk, \tau_{\mathsf{SIG}}^{\mathsf{sk}} \rangle$ then
            return $(c, vk(\hat{sk}))$
        else
            return $(c, g_{\tau_{\mathsf{SIG}}^{\mathsf{vk}}}^{l(c)})$
    else if $c = \langle \sigma, m, vk, \tau_{\mathsf{SIG}}^{\mathsf{sig}} \rangle$ then
        if $(\langle vk, \tau_{\mathsf{SIG}}^{\mathsf{vk}} \rangle, vk(\hat{sk})) \in L$
            and $\mathsf{SIG.Vfy}(vk, \sigma, m) = \mathsf{true}$ then
            return $(c, sig^{l(c)}(\hat{sk}, m))$
        else
            return $(c, g_{\tau_{\mathsf{SIG}}^{\mathsf{sig}}}^{l(c)})$
    else
        return $(c, g_{\top}^{l(c)})$
```

Figure 3.11: Open function for signatures.

The $\mathsf{valid}_{\mathsf{SIG}}$ predicate

Intuitively, we require static corruption of signing keys and that verification and signing keys are only used for signing and verification. Formally, based on the current trace $\mathbb{T}$ of all parse and generate requests of the adversary, the predicate $\mathsf{valid}_{\mathsf{SIG}}$ returns true only if the following conditions hold:

1. The trace starts with a query "init T, H" (T, H may be the empty list respectively). There are no further init queries.

2. The adversary may only generate keys in the init query. Concretely, this is guaranteed by the following rules:

   a) For the query "init T, H", the function symbols sk and vk may only occur in a term $t \in T$ (i.e., not as subterms of other terms) of one of the two following types (for $l \in \mathsf{labelsH}$):

      - $t = vk(sk^l())$ (to generate an honest signing key)
      - $t = sk^l()$ (to generate a corrupted signing key)

      Any label l for $sk^l()$ must be unique in T.

   b) Any occurrence of $vk(sk^l())$ or $sk^l()$ in a $\mathsf{generate}$ query must have occurred in the init query.

3. The adversary must not use the function symbol sig in the init query.

4. The term $sk^l()$ may only occur as the first argument of sig.

Checking the implementation

We first observe that $\mathcal{I}_{\mathsf{SKE}}$ is collision-free (Definition 41): Basically, collisions for keys can only occur with negligible probability since they break the security of the scheme (which is strong EUF-CMA secure). Collisions of signatures can only occur with negligible probability as well due to the EUF-CMA security. Furthermore, it is easy to see that $\mathsf{open}_{\mathsf{SIG}}$ meets the requirements of Definition 42 and that $\mathsf{valid}_{\mathsf{SIG}}$ meets the requirements for valid functions.

3.7.3.4 Signature composability

Theorem 6. *Let $\mathcal{M}$ be a symbolic model and $\mathcal{I}$ be deduction sound implementation of $\mathcal{M}$. If ($\mathcal{M}_{\mathsf{SIG}}$,$\mathcal{I}_{\mathsf{SIG}}$) and ($\mathcal{M}$,$\mathcal{I}$) are compatible (see requirements in Section 3.5), then $\mathcal{I} \cup \mathcal{I}_{\mathsf{SIG}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{SIG}}$ for any $\mathcal{I}_{\mathsf{SIG}}$ constructed from a strongly EUF-CMA-secure signature scheme.*

Proof. First, we briefly describe the intuition behind the proof: Let $\mathcal{A}$ be an adversary playing the deduction soundness game. Assume that $\mathcal{A}$ queries "$\mathsf{parse}\ c$" and c is parsed as a non-DY term t that contains a signature $sig := sig^{\hat{l}}(sk^{l}(), m)$ and $S \nvdash sig$ (where S is the list of terms generated for $\mathcal{A}$ in the deduction soundness game). We distinguish two possible ways the adversary could potentially have learned sig:

If sig was previously generated for $\mathcal{A}$ (i.e., $sig \in st(S)$ and $\hat{l} \in \mathsf{labelsH}$), we say that $\mathcal{A}$ *reconstructed sig*. Since signatures and transparent functions do not introduce function symbols that allow for signatures as input such that the signature is not derivable from the constructed term, $\mathcal{A}$ must have broken the deduction soundness of $\mathcal{I}$ in this case. Hence, using $\mathcal{A}$, we can construct an successful adversary $\mathcal{B}$ on the deduction soundness of $\mathcal{I}$. $\mathcal{B}$ simulates signatures using transparent functions.

If sig was not previously generated for $\mathcal{A}$ (i.e., $\hat{l} \in \mathsf{labelsA}$), we say that $\mathcal{A}$ *forged sig*. In this case $\mathcal{A}$ can be used to break the strong EUF-CMA security of the signature scheme.

Since reconstructions and forgeries can only occur with negligible probability the composed implementation $\mathcal{I} \cup \mathcal{I}_{\mathsf{SIG}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{SIG}}$.

Game 0

In Game 0 $\mathcal{A}$ plays the original deduction soundness game $\mathsf{DS}_{(\mathcal{M}\cup\mathcal{M}_{\mathsf{SIG}})\cup\mathcal{M}_{\mathsf{tran}}(\nu),(\mathcal{I}\cup\mathcal{I}_{\mathsf{SIG}})\cup\mathcal{I}_{\mathsf{tran}}(\nu)}(\eta)$.

Game 1

In Game 1 we replace the $\mathsf{generate}$ function by the collision-aware generate function from Figure 3.3. Since $(\mathcal{I} \cup \mathcal{I}_{\mathsf{SIG}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu)$ is a collision-free implementation Game 0 and Game 1 are indistinguishable by Lemma 59.

Game 2

Game 2 is Game 1 with a changed winning condition for the adversary. First we introduce the set of *reconstruction rules* as follows: $\mathcal{D}_r := \{\frac{t}{sig^{\hat{h}}(sk^{h}(),m)} : t$ is a term from $(\mathcal{M} \cup \mathcal{M}_{\mathsf{SIG}}) \cup \mathcal{M}_{\mathsf{tran}}(\nu)$, $\hat{h}, h \in \mathsf{labelsH}$ and $sig^{\hat{h}}(sk^{h}(), m)$ is a subterm of $t\}$.

Let $\vdash_1$ denote the deduction relation of the previous game based on the rules from $\mathcal{D} \cup \mathcal{D}_{\mathsf{SIG}} \cup \mathcal{D}_{\mathsf{tran}}$. In this game we use the deduction relation $\vdash_2$ based on the rules from $\mathcal{D} \cup \mathcal{D}_{\mathsf{SIG}} \cup \mathcal{D}_{\mathsf{tran}} \cup \mathcal{D}_r$, i.e., the adversary may now use any honestly generated signature to deduce new terms. In other words, the adversary cannot win (produce a non-DY term) any longer by using a signature that has been generated for it.

Claim: Game 1 and Game 2 are indistinguishable

The only difference between Game 1 and Game 2 is the handling of parse requests. Since $\vdash_2$ potentially allows to deduce more terms from a given set of terms S than $\vdash_1$, the adversary might produce a non-DY term in Game 1 that is DY in Game 2. We now show that the adversary breaks the deduction soundness of $\mathcal{I}$ in that case. This part of the proof is very similar to the proof of indistinguishability between Games 2 and 3 in Theorem 5.

The simulator. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ on the deduction soundness of $\mathcal{I}$. Towards this goal $\mathcal{B}$, in addition to the transparent symbolic model $\mathcal{M}_{\mathsf{tran}}(\nu')$ used by $\mathcal{A}$, uses a transparent symbolic model to simulate signatures in the deduction soundness game for $\mathcal{I}$.

Transparent symbolic model for signatures. We first describe the parametrized transparent symbolic model $\mathcal{M}_{\mathsf{SIG}}^{\mathsf{tran}}(\nu)$ and the corresponding parametrized implementation $\mathcal{I}_{\mathsf{SIG}}^{\mathsf{tran}}(\nu)$ $\mathcal{B}$ will use to simulate $\mathcal{I}_{\mathsf{SIG}}$. We use the data types and subtype relation from $\mathcal{M}_{\mathsf{SIG}}$. ν is expected to be an encoding of a list of label-triple pairs $(l, (ek, dk, sk'))$ ($l \in \mathsf{labels}$) where the triple consist of a keypair vk, sk and an additional value sk' (used to represent honest signing keys in the library). The signature $\Sigma_{\mathsf{SIG}}^{\mathsf{tran}}$ is the following:

- deterministic f_{sk}^{h} with $\mathrm{ar}(f_{sk}^{h}) = \tau_{\mathsf{SIG}}^{\mathsf{sk}}$ for all labels $l \in \nu$
- deterministic f_{vk}^{h} with $\mathrm{ar}(f_{vk}^{h}) = \tau_{\mathsf{SIG}}^{\mathsf{vk}}$ for all labels $l \in \nu$
- randomized $f_{sig(sk^{h}(),\cdot)}$ with $\mathrm{ar}(f_{sig(sk^{h}(),\cdot)}) = \top \to \tau_{\mathsf{SIG}}^{\mathsf{sig}}$ for all labels $l \in \nu$

We specify a parametrized implementation $\mathcal{I}_{\mathsf{SIG}}^{\mathsf{tran}}(\nu)$ for $\mathcal{M}_{\mathsf{SIG}}^{\mathsf{tran}}$ as follows for each $(l, (vk, sk, sk')) \in \nu$:

- $(M_{\mathsf{SIG}}^{\mathsf{tran}}\ f_{sk}^{h})()$ returns $\langle vk, sk', \tau_{\mathsf{SIG}}^{\mathsf{sk}}\rangle$
- $(M_{\mathsf{SIG}}^{\mathsf{tran}}\ f_{vk}^{h})()$ returns $\langle vk, \tau_{\mathsf{SIG}}^{\mathsf{vk}}\rangle$
- $(M_{\mathsf{SIG}}^{\mathsf{tran}}\ f_{sig(sk^{h}(),\cdot)})(m; r)$ returns $(M_{\mathsf{SIG}}\ sig)(\langle sk, \tau_{\mathsf{SIG}}^{\mathsf{sk}}\rangle, m; r)$

Furthermore, we have to define the transparent modes of operation for $M_{\mathsf{SIG}}^{\mathsf{tran}}$. $(M_{\mathsf{SIG}}^{\mathsf{tran}}\mathsf{proj}\ f_{sig(sk^{h}(),\cdot)}\ 1)(sig)$ parses sig as $\langle \sigma, m, vk, \tau_{\mathsf{SIG}}^{\mathsf{sig}}\rangle$ and returns m if parsing succeeds and $\perp$ otherwise. $(M_{\mathsf{SIG}}^{\mathsf{tran}}\ \mathsf{func})(b)$ returns f_{sk}^{h} if for some $(l, (vk, sk, sk')) \in \nu$ b can be parsed as $\langle vk, sk', \tau_{\mathsf{SIG}}^{\mathsf{sk}}\rangle$ and f_{vk}^{h} if b can be parsed as $\langle vk, \tau_{\mathsf{SIG}}^{\mathsf{vk}}\rangle$. If $b \in [\![\tau_{\mathsf{SIG}}^{\mathsf{sig}}]\!]$, $(M_{\mathsf{SIG}}^{\mathsf{tran}}\ \mathsf{func})$ tries to parse b as $\langle \sigma, m, vk, \tau_{\mathsf{SIG}}^{\mathsf{sig}}\rangle$. If parsing succeeds, $\mathsf{SIG.Vfy}(vk, \sigma, m) = \mathsf{true}$ and there is a $(l, (vk, sk, sk')) \in \nu$, then $(M_{\mathsf{SIG}}^{\mathsf{tran}}\ \mathsf{func})$ returns $f_{sig(sk^{h}(),\cdot)}$, $\perp$ otherwise.

Convert terms. Adversary $\mathcal{A}$ uses the function symbols of the original symbolic model $\mathcal{M}_{\mathsf{SIG}}$. Hence $\mathcal{B}$ needs to map these symbols to the corresponding transparent functions. Towards this goal we introduce the function convert as follows (the first matching rule is applied):

- $\mathsf{convert}(f^{l}(t_1, \ldots, t_n)) = f^{l}(\mathsf{convert}(t_1), \ldots, \mathsf{convert}(t_n))$ for all $f \notin \Sigma_{\mathsf{SIG}}$
- $\mathsf{convert}(sk^{h}()) = f_{sk}^{h}()$
- $\mathsf{convert}(vk(sk^{h}())) = f_{vk}^{h}()$
- $\mathsf{convert}(sig^{\hat{h}}(sk^{h}(), m) = f_{sig(sk^{h}(),\cdot)}^{\hat{h}}(\mathsf{convert}(m))$

For a list of terms T we define $\mathsf{convert}(T) := \{\mathsf{convert}(t) : t \in T\}$.

$\mathcal{B}$ simulates Game 1 for $\mathcal{A}$ while playing

$$\mathsf{DS}_{\mathcal{M}\cup(\mathcal{M}^{\mathsf{tran}}_{\mathsf{SIG}}(\nu)\cup\mathcal{M}_{\mathsf{tran}}(\nu')),\mathcal{I}(\cup\mathcal{I}^{\mathsf{tran}}_{\mathsf{SIG}}(\nu)\cup\mathcal{I}_{\mathsf{tran}}(\nu'))}(\eta)$$

Note that we can generically compose $\mathcal{M}^{\mathsf{tran}}_{\mathsf{SIG}}(\nu) \cup \mathcal{M}_{\mathsf{tran}}(\nu')$ to one parametrized transparent model $\mathcal{M}'_{\mathsf{tran}}(\nu||\nu')$ since ν and ν' must be good (analogously for the implementation). However, for the sake of clarity, we keep them apart to distinguish the transparent functions (and parameter) provided by $\mathcal{A}$ from the additional transparent functions introduced by $\mathcal{B}$. Next we describe how $\mathcal{B}$ deals with the queries received from $\mathcal{A}$.

init query. $\mathcal{B}$ receives a lists of terms T, H from $\mathcal{A}$. Initially, $\mathcal{B}$ sets $\nu := \emptyset$. For each occurrence of $sk^l() \in T$ $\mathcal{B}$ then picks a nonce $r \leftarrow \{0,1\}^\eta$ and generates a keypair $(vk, sk) := \mathsf{SIG.KeyGen}(1^\eta, r)$. $\mathcal{B}$ sets $sk' = sk$. (dk' represents the signing key in the library and will be a fresh random value for honest signing keys in a later simulation.) $\mathcal{B}$ then adds $(l, (vk, sk, sk'))$ to ν. Finally, $\mathcal{B}$ sends $\nu'||\nu$ to its game and subsequently queries "$\mathsf{init}\ \mathsf{convert}(T), \mathsf{convert}(H)$". Afterwards, $\mathcal{B}$ queries "$\mathsf{sgenerate}$ $sk^l()$" for each $sk^l() \in T$.

The other queries are handled exactly as in the simulation using transparent functions in Theorem 5. Likewise, the proof that the simulation - apart from the winning condition - perfectly simulates Game 1 (and thus Game 2) is analogous to the corresponding proof in Theorem 5.

The changed winning condition. Let us now assume that $\mathcal{A}$ sends a "parse c" such that c is parsed as a non-DY term t in Game 1 while t is DY in Game 2. Concretely, we have $S \nvdash_1 t$ and $S \vdash_2 t$ where S is the set of the terms previously generated for $\mathcal{A}$. We have to show that $\mathsf{convert}(S) \nvdash_{sim} \mathsf{convert}(t)$ where $\mathsf{convert}(S) := \{\mathsf{convert}(t) : t \in S\}$, i.e., that $\mathcal{A}$ breaks the small library in this case. Since the simulation is perfect, c is actually parsed as $\mathsf{convert}(t)$ in the simulation.

From $\vdash_2$ to $\vdash_1$. By Lemma 61 there is an S' with $S \vdash_2 S'$ such that $S' \nvdash_1 t$ and $S' \cup \{sig^{\hat{h}}(sk^h, m)\} \vdash_1 t$ where $sig^{\hat{h}}(sk^h, m) \in st(S')$ (i.e., derivable by a rule from $\mathcal{D}_r$).

From $\vdash_{sim}$ to $\vdash_1$. In this paragraph we show that if $\mathsf{convert}(t)$ was deducible in the simulation, t would be deducible in Game 1. Therefor we assume $\mathsf{convert}(S') \vdash_{sim}$ $\mathsf{convert}(t)$ towards contradiction. Concretely, we show that it implies $S' \vdash_1 t$ contradicting $S' \nvdash_1 t$ which we have due to the previous paragraph.

Let π be a proof for $\mathsf{convert}(S') \vdash_{sim} \mathsf{convert}(t)$. Then there is a proof π' for $\mathsf{convert}(S') \vdash_{sim} \mathsf{convert}(t)$ such that for every $\alpha_i = \frac{m}{f^l_{sig(sk^h(),\cdot)}(m)}$ we have $f^l_{sig(sk^h(),\cdot)}(m) \in st(\mathsf{convert}(t))$.

Assume we had an $\alpha_i = \frac{m}{f^l_{sig(sk^h(),\cdot)}(m)}$ in π such that $f^l_{sig(sk^h(),\cdot)}(m) \notin st(\mathsf{convert}(t))$. Furthermore, and w.l.o.g., we assume that $f^l_{sig(sk^h(),\cdot)}(m) \notin st(\mathsf{convert}(S'))$ (a justification for this follows in the next paragraph). We define the substitution θ on terms as $\theta(f^l(t_1, \ldots, t_n)) = f^l(\theta(t_1), \ldots, \theta(t_n))$ for all function symbols $f! = f_{sig(sk^h(),\cdot)}$ and $\theta(f^l_{sig(sk^h(),\cdot)}(m)) = m$ (we could pick any term from S_{i-1} here). $\mathsf{convert}(S') =: S_0 \overset{\alpha_1}{\rightarrow} \cdots \overset{\alpha_{i-1}}{\rightarrow} S_{i-1} \overset{\theta\alpha_{i+1}}{\rightarrow} \theta(S_{i+1}) \cdots \overset{\theta\alpha_n}{\rightarrow} \theta(S_n)$ is a new proof $\widetilde{\pi}$ for $\mathsf{convert}(S') \vdash_{sim} \mathsf{convert}(t)$ since

- $S_n \ni \mathsf{convert}(t) \rightarrow \theta(S_n) \ni \theta(\mathsf{convert}(t)) = \mathsf{convert}(t)$ since $f^l_{sig(sk^h(),\cdot)}(m) \notin st(\mathsf{convert}(t))$

- $\theta(S_j) = S_j$ and $\theta(\alpha_j) = \alpha_j$ for $j \in \{1, \ldots, i-1\}$ since $f^l_{sig(sk^h(),\cdot)}(m) \notin st(\mathsf{convert}(S'))$
- $\theta(\alpha_j)$ is still an instantiation of the same rule as α_j for $j \in \{i+1, \ldots, n\}$ since the only available rule that uses the structure of $f^l_{sig(sk^h(),\cdot)}(m)$ is $\frac{f^l_{sig(sk^h(),\cdot)}(x)}{x}$. However, none of the α_j can be an instantiation of this rule since $m \in S_{j-1}$ for $j \in \{i+1, \ldots, n\}$ (and we are only considering proofs where already known terms must not be derived again).

Why can we assume $f^l_{sig(sk^h(),\cdot)}(m) \notin st(\mathsf{convert}(S'))$ w.l.o.g.? $f^l_{sig(sk^h(),\cdot)}(m) \in st(\mathsf{convert}(S'))$ implies $sig^l(sk^h(), m) \in st(S')$. Furthermore, the setting in Game 1 satisfies the requirements for Lemma 60 and $t' = sig^l(sk^h(), m)$: Instantiations of rules from $\mathcal{D} \cup \mathcal{D}_{\mathsf{tran}}$ use signature terms in a black-box way and hence satisfy property (i). Instantiations of rules from $\mathcal{D}_{\mathsf{SIG}}$ satisfy either (i) or (ii). Thus, by Lemma 60 we have $S' \vdash_1 sig^l(sk^h(), m)$. Converting this proof leads to a proof $\mathsf{convert}(S') \vdash_{sim} f^l_{sig(sk^h(),\cdot)}(m)$ that does not use a rule of the type $\frac{x}{f^l_{sig(sk^h(),\cdot)}(x)}$. Hence we can always find a proof where instantiations $\frac{m}{f^l_{sig(sk^h(),\cdot)}(m)}$ are only used for $f^l_{sig(sk^h(),\cdot)}(m) \notin st(\mathsf{convert}(S'))$.

In conclusion, if we find a proof π for $\mathsf{convert}(S') \vdash_{sim} \mathsf{convert}(t)$, then we find a proof π' that does not contain any α_i of type $\frac{m}{f^l_{sig(sk^h(),\cdot)}(m)}$. We then apply $\mathsf{convert}^{-1}$ to this proof and get a proof for $S' \vdash_1 t$ which is a contradiction to our requirements for S'. Hence we cannot have $\mathsf{convert}(S') \vdash_{sim} \mathsf{convert}(t)$ and thus "$\mathsf{parse}\ c$" lets the simulator win the deduction soundness game for $\mathcal{I}$ where signatures are replaced by transparent functions. This proves our claim that Game 1 and Game 2 are indistinguishable.

Game 3

We define the set $\mathcal{D}_f := \{\frac{vk(x)}{x}\}$. Let $\vdash_2$ denote the deduction relation of the previous game based on the rules from $\mathcal{D} \cup \mathcal{D}_{\mathsf{SIG}} \cup \mathcal{D}_{\mathsf{tran}} \cup \mathcal{D}_r$. In this game we use the deduction relation $\vdash_3$ based on the rules from $\mathcal{D} \cup \mathcal{D}_{\mathsf{SIG}} \cup \mathcal{D}_{\mathsf{tran}} \cup \mathcal{D}_r \cup \mathcal{D}_f$, i.e., the adversary is allowed to derive the signing key for any verification key in use. This also means that the adversary can no longer win by producing a forgery. Furthermore we replace the honest signing keys in the library by random bistrings analogously to Game 2 in Theorem 5.

Claim: Game 2 and Game 3 are indistinguishable.

Let $\mathcal{A}$ be an adversary for Game 2. To prove our claim we show how an adversary $\mathcal{B}$ for the strong EUF-CMA security game can be constructed using $\mathcal{A}$. Whenever $\mathcal{A}$ wins Game 2 by having a bitstring c parsed as a term t that is non-DY in Game 2 but is DY in Game 3, $\mathcal{B}$ will win its game. Since our signature scheme is strongly EUF-CMA secure, such a "distinguishing" c can only be produced by an adversary with negligible probability. We first describe the adversary $\mathcal{B}$ in detail.

init request. By requirement (1) $\mathcal{A}$ starts with a request "$\mathsf{init}\ T, H$". Furthermore, by requirement (2), we can distinguish three types of terms t in T. They are handled by the simulator as follows:

- $t = vk(sk^l())$ ($\mathcal{A}$ requests an honest signing key): $\mathcal{B}$ request a verification key vk with a corresponding signing oracle from the strong EUF-CMA game.

Then, it picks a random value sk' and sets $\hat{sk} := \langle vk, sk', \tau^{\mathsf{sk}}_{\mathsf{SIG}} \rangle$. We refer to the signing oracle corresponding to vk with $\mathcal{O}^{sig}_{\hat{sk}}(\cdot)$. $\mathcal{B}$ sets $L := L \cup \{(\hat{sk}, sk^l()), (\langle vk, \tau^{\mathsf{vk}}_{\mathsf{SIG}} \rangle, vk(\hat{sk}))\}$ and adds $\hat{vk}$ to the list of bistrings that will be returned to $\mathcal{A}$.

- otherwise we have $t = sk^h()$ ($\mathcal{A}$ requests a corrupted signing key) or t does not contain function symbols from Σ_{SIG} (note that t must not contain signatures by 3). In this case $\mathcal{B}$ uses the normal $\mathsf{generate}$ function and computes $(c, L) := \mathsf{generate}(t, L)$. It then adds c to the list of bitstrings that will be returned to $\mathcal{A}$.

After the init request, $\mathcal{B}$ changes the $\mathsf{generate}$ function to use the oracles for signatures under honest signing keys. Concretely, it replaces the line

$$\text{let } c := (M\ f)(c_1, \ldots, c_n; r)$$

with

```
if $t = sig^{\hat{h}}(sk^h(), m)$ then
    let $c := \langle \mathcal{O}^{sig}_{c_1}(c_2), c_2, vk, \tau^{\mathsf{sig}}_{\mathsf{SIG}} \rangle$
else
    let $c := (M\ f)(c_1, \ldots, c_n; r)$
```

Note that the bitstrings c_1 and c_2 correspond to the signing key and the message to be signed respectively. Using the updated $\mathsf{generate}$ function, $\mathcal{B}$ simulates the rest of Game 2 according to the normal deduction soundness game from Figure 3.7. The simulation is indistinguishable:

- There is a bijection between the randomness used in an execution of Game 2 and the randomness used in the simulation: The randomness used for key generation and for generating signatures under honest keys is used by the strong EUF-CMA game in the simulation (and this is the only difference between Game 2 and the simulation as far as the use of randomness is concerned).

- The fact that the library contains randomized honest signing keys cannot be detected by the adversary for the same reason the randomized honest encryption keys cannot be detected by the adversary in Game 2 in Theorem 5: According to $\mathsf{valid}_{\mathsf{SIG}}$ (requirement (4)) signing keys may only occur as the first argument to sig. If the adversary parses a bitstring that can be used as a signing key, $\mathcal{B}$ wins the EUF-CMA game.

Extracting a forgery. Let "parse c" be a request sent by $\mathcal{A}$ such that $t := L[[c]]$ and $S \nvdash_2 t$ but $S \vdash_3 t$.

We claim that t contains either an honest signing key $sk^h()$ or a forgery under an honest signing key $sig^l(sk^h(), m) \notin st(S)$. To prove our claim we assume towards contradiction that t contains neither and let π be a proof for $S \vdash_3 t$. Then, analogously to above, we can first remove all instantiations of the rule $\frac{sk^h() \quad x}{sig^l(sk^h(), x)}$ for honest signing keys $sk^h()$ from π yielding a proof π'. Next we remove all instantiations of the rule $\frac{vk(sk^h())}{sk^h()}$ from π' following the same principle and get a proof π''. However, π'' is a proof for $S \vdash_2 t$ which contradicts our initial assumption. Hence t contains either an honest signing key or a forgery.

Since $\mathcal{B}$ could parse the bitstring c, the library L contains a bitstring corresponding to every subterm of t. If t contains the term for an honest signing key, $\mathcal{A}$ must have guessed the randomized bitstring sk' for this key in the library which was never used to compute any bitstring sent to $\mathcal{A}$. This can only happen with negligible probability. Hence, c contains a forgery with overwhelming probability and $\mathcal{B}$ can use this forgery to win the strong EUF-CMA game it is playing (note that we need strong EUF-CMA security here since the forgery could be a re-randomization of a signature that was generated for the adversary and we wouldn't break EUF-CMA security in this case).

$\mathcal{A}$ cannot win Game 3 with non-negligible probability

To conclude the proof we observe that an adversary $\mathcal{A}$ that wins Game 3 with non-negligible probability also wins the deduction soundness game for $\mathcal{I}$ with non-negligible probability: Analogously to the proof for the indistinguishability of Game 0 and Game 2 we can construct an adversary $\mathcal{B}$ that attacks the deduction soundness of $\mathcal{I}$ and simulates signatures using transparent functions. The simulation is perfect and if $\mathcal{A}$ wins, $\mathcal{B}$ wins since the deduction rules in Game 3 are effectively a superset of the deduction rules in the simulation.

Hence $\mathcal{A}$ cannot win Game 3 with non-negligible probability and $\mathcal{I} \cup \mathcal{I}_{\mathsf{SIG}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{SIG}}$. □

Lemma 60. *Let $\mathcal{M} = (\mathcal{T}, \preccurlyeq, \Sigma, \mathcal{D})$ be a symbolic model and $\mathcal{I}$ an implementation of $\mathcal{M}$. For the set of terms generated for the adversary S during the deduction soundness game $\mathsf{DS}_{\mathcal{M},\mathcal{I},\mathcal{A}}(\eta)$ holds, that for any term $t' \in st(S)$ with adversarial label we have $S \vdash t'$ if for all instantiations $\alpha = \frac{t_1 \quad \cdots \quad t_n}{t}$ of rules from $\mathcal{D}$ with $t' \in st(t)$ meet at least one of the following properties:*

(i) $t' \in st(t_i)$ for some $i \in \{1, \ldots, n\}$

(ii) for any $\widetilde{S} \overset{\alpha}{\rightarrow} \widetilde{S}'$ we have $\widetilde{S}' \vdash t'$

Proof. Since $\mathsf{generate}$ does not introduce adversarial labels, every subterm with adversarial label of any term in S must have been introduced by a previous parse request. Let "parse c" be the first parse request that returns a term t such that t' is a subterm of t. Since the adversary didn't win with that request, t must be a DY term with respect to $S' \subseteq S$ where S' denote the terms generated for the adversary until that parse request. Concretely, we have $S' \vdash t$ and thus a proof $S' =: S_0 \overset{\alpha_1}{\rightarrow} S_1 \overset{\alpha_2}{\rightarrow} \cdots \overset{\alpha_n}{\rightarrow} S_n$ such that $t \in S_n$. Let $i \in \{1, \ldots, n\}$ be the smallest index such that $t' \in st(S_i)$ ($i \neq 0$ since $t' \in st(S')$). α_i cannot meet property (i) since $t' \notin st(S_{i-1})$ (i is minimal). Hence we have $S_i \vdash t'$ by (ii) and $S' \vdash t'$ since $S'\ deducS_i$ and $S \vdash t'$ since $S' \subseteq S$. □

Lemma 61. *Let $\mathcal{M} = (\mathcal{T}, \preccurlyeq, \Sigma, \mathcal{D})$ be a symbolic model and let $\mathcal{D}' \supseteq \mathcal{D}'$ be a set of deduction rules for $\mathcal{M}$. $\vdash$ and $\vdash'$ denote the deduction relations corresponding to $\mathcal{D}$ and $\mathcal{D}'$ respectively. Let S be a set of terms and t be a term such that $S \nvdash t$ and $S \vdash' t$. Then there is a set of terms S' and a term t' such that*

- *$S \vdash' S'$*
- *$S' \nvdash t$*
- *$S' \cup \{t'\} \vdash t$*

- $S' \vdash' t'$

Proof. $S \vdash' t$ implies that there is a deduction proof $S =: S_0 \xrightarrow{\alpha_1} S_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} S_n$. For $\alpha_i = \frac{u_1 \quad \cdots \quad u_m}{u}$ and $S_{i-1} \xrightarrow{\alpha_i} S_i$ we require $u_i \in S_{i-1}$ and w.l.o.g. $u \notin S_{i-1}$.

Let $j \in \{1, \ldots, n\}$ be the biggest index such that α_j is an instantiation of a rule from $\mathcal{D}' \setminus \mathcal{D}$ and $S_j \nvdash t$. (There must be such a rule since we would have $S \vdash t$ otherwise.) Then we set $S' := S_{j-1}$ and t' to be the one element in $S_j \setminus S_{j-1}$. We have

- $S \vdash' S'$ since $S \xrightarrow{\alpha_1} S_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{j-1}} S_{j-1} = S'$
- $S' \nvdash t$ by requirements for j
- $S' \cup \{t'\} \vdash t$: If there is an instantiation $\alpha_{j'}$ of a rule from $\mathcal{D}' \setminus \mathcal{D}$ with $j' > j$, we have $S_{j'-1} \vdash t$ by requirements for j. For the smallest such index j' we observe $S_j \vdash S_{j'-1}$ and hence $S_j = S' \cup \{t'\} \vdash t$.
- $S' \vdash' t'$ obviously by application of α_j

This concludes our proof. □

3.7.4 Secret key encryption

In this section we define a symbolic model $\mathcal{M}_{\mathsf{SKE}}$ for secret key encryption and a corresponding implementation $\mathcal{I}_{\mathsf{SKE}}$ based on a secret key encryption scheme $(\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$. We show that composition of $\mathcal{M}_{\mathsf{SKE}}$ and $\mathcal{I}_{\mathsf{SKE}}$ with any symbolic model $\mathcal{M}$ comprising a deduction sound implementation $\mathcal{I}$ preserves this property for the resulting implementation, i.e., $\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{SKE}}$ if we use a secret key encryption scheme that is IND-CCA and INT-CTXT secure.

3.7.4.1 Computational preliminaries

Definition 48 (secret key encryption scheme). *A secret key encryption scheme (*SKE *scheme) is a triple of algorithms* (SKE.KeyGen, SKE.Enc, SKE.Dec)*.*

The probabilistic key generation algorithm SKE.KeyGen *takes an encoding of the security parameter and some randomness as inputs and generates a* secret key *k.*

The probabilistic encryption algorithm SKE.Enc *takes three arguments: a secret key k, the message $m \in \{0,1\}^*$, and some randomness $r \in \{0,1\}^\eta$. It computes a* ciphertext $c := \mathsf{SKE.Enc}(k, m; r)$.[4]

The decryption algorithm SKE.Dec *takes a secret key and a ciphertext as inputs and returns a value from $\{0,1\}^* \cup \{\bot\}$. We require* perfect correctness*, i.e.,*

$$\mathsf{SKE.Dec}(k, \mathsf{SKE.Enc}(k, m; r)) = m$$

for all $r \leftarrow \{0,1\}, m \in \{0,1\}^*$ and $k \leftarrow$ $\mathsf{SKE.KeyGen}(1^\eta)$*.*

Definition 49 (IND-CCA security of SKE schemes). *An SKE scheme* (SKE.KeyGen, SKE.Enc, SKE.Dec) *is* IND-CCA secure *if for all PPT adversaries $\mathcal{A}$ the probability*

$$\mathbb{P}\left[\textit{IND-CCA-SKE}_{\mathcal{A}}^{(\mathsf{SKE.KeyGen},\mathsf{SKE.Enc},\mathsf{SKE.Dec})}(\eta) = 1\right] - \frac{1}{2}$$

is negligible for the IND-CCA game from Figure 3.12. Note that, analogously to Figure 3.8, we can also define an equivalent multi-user version of IND-CCA security for SKE schemes.

```
$\mathsf{IND\text{-}CCA\text{-}SKE}_{\mathcal{A}}^{(\mathsf{SKE.KeyGen},\mathsf{SKE.Enc},\mathsf{SKE.Dec})}(\eta)$:
    let $b \leftarrow \{0,1\}$
    let $k \leftarrow \mathsf{SKE.KeyGen}(1^\eta)$
    let $\mathsf{ciphers} = \emptyset$

    on request "$\mathsf{encrypt}\ m$" do
        if $b = 0$ then
            let $c \leftarrow \mathsf{SKE.Enc}(k, 0^{|m|})$
            add $(c, m)$ to $\mathsf{ciphers}$
        else
            let $c \leftarrow \mathsf{SKE.Enc}(k, m)$
        send $c$ to $\mathcal{A}$

    on request "$\mathsf{decrypt}\ c$" do
        if $b = 0$ and $(c, m) \in \mathsf{ciphers}$ for some $m$ then
            send $m$ to $\mathcal{A}$
        else
            send $\mathsf{SKE.Dec}(k, c)$ to $\mathcal{A}$

    on request "$\mathsf{guess}\ b'$" do
        if $b = b'$ then return 1 else return 0
```

Figure 3.12: The IND-CCA game for an SKE scheme (SKE.KeyGen, SKE.Enc, SKE.Dec).

Definition 50 (INT-CTXT security of SKE schemes)**.** *An SKE scheme* (SKE.KeyGen, SKE.Enc, SKE.Dec) *is* INT-CTXT secure *if for all PPT adversaries $\mathcal{A}$ the probability*

$$\mathbb{P}\left[\mathit{IND\text{-}CCA\text{-}SKE}_{\mathcal{A}}^{(\mathsf{SKE.KeyGen},\mathsf{SKE.Enc},\mathsf{SKE.Dec})}(\eta) = 1\right] - \frac{1}{2}$$

is negligible for the INT-CTXT game from Figure 3.13.

3.7.4.2 Symbolic model

We first define the symbolic model $(\mathcal{T}_{\mathsf{SKE}}, \preccurlyeq_{\mathsf{SKE}}, \Sigma_{\mathsf{SKE}}, \mathcal{D}_{\mathsf{SKE}})$ for secret key encryption. The signature Σ_{SKE} features the following function symbols

$$\begin{aligned} k_x &: \tau_{\mathsf{SKE}}^{\mathsf{k}_x} \\ E_x &: \tau_{\mathsf{SKE}}^{\mathsf{k}_x} \times \top \to \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}} \end{aligned}$$

for $x \in \{h, c\}$. The randomized functions k_h and k_c return honest or corrupted keys respectively. The randomized function E_x has arity $\tau_{\mathsf{SKE}}^{\mathsf{k}_x} \times \top \to \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}}$ and represents a ciphertext under the given key. To complete the formal definition we set

$$\mathcal{T}_{\mathsf{SKE}} := \{\top, \tau_{\mathsf{SKE}}^{\mathsf{k}_x}, \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}}\}$$

[4] Since the message m is of basetype in symbolic model given below, we require a scheme with message space $\{0,1\}^*$.

```
$\mathsf{INT\text{-}CTXT\text{-}SKE}_{\mathcal{A}}^{(\mathsf{SKE.KeyGen},\mathsf{SKE.Enc},\mathsf{SKE.Dec})}(\eta)$:
    let $k \leftarrow \mathsf{SKE.KeyGen}(1^\eta)$
    let $\mathsf{ciphers} := \emptyset$

    on request "$\mathsf{encrypt}\ m$" do
        let $c \leftarrow \mathsf{SKE.Enc}(k, m)$
        add $c$ to $\mathsf{ciphers}$
        send $c$ to $\mathcal{A}$

    on request "$\mathsf{forge}\ c$" do
        let $m := \mathsf{SKE.Dec}(k, c)$
        if $m \neq \bot$ and $c \notin \mathsf{ciphers}$ then return 1 else return 0
```

Figure 3.13: The INT-CTXT game for an SKE scheme (SKE.KeyGen, SKE.Enc, SKE.Dec).

All introduced types are direct subtypes of the base type $\top$ (this defines $\preccurlyeq_{\mathsf{SKE}}$). The deduction system captures the security of secret key encryption

$$\mathcal{D}_{\mathsf{SKE}} := \left\{ \begin{array}{cc} \multicolumn{2}{c}{\dfrac{k_x^l() \quad m}{E_x^{l_a}(k_x^l(),m)},} \\ \dfrac{E_h^{l_a}(k_h^l(),m)}{m}, & \dfrac{E_c^{\hat{l}}(k_c^l(),m)}{m} \end{array} \right\}$$

These rules are valid for arbitrary labels $l, \hat{l} \in \mathsf{labels}$ and adversarial labels $l_a \in \mathsf{labelsA}$. Read from top left to bottom right the following intuitions back up the rules:

- The adversary can use any honestly generated key to encrypt some term u.
- The adversary knows the message contained in any adversarial encryption.
- The adversary knows the message contained in any encryption under a corrupted key.

3.7.4.3 Implementation

We now give a concrete implementation $\mathcal{I}_{\mathsf{SKE}}$ for secret key encryption. Let (SKE.KeyGen, SKE.Enc, SKE.Dec) be a secret key encryption scheme. The computable interpretations of, k_x and E_x (for $x \in \{h, c\}$) are as follows:

- $(M_{\mathsf{SKE}}\ k_x)(r)$: Let $k := \mathsf{SKE.KeyGen}(1^\eta; r)$. Return $\langle k, \tau_{\mathsf{SKE}}^{\mathsf{k}_x} \rangle$
- $(M_{\mathsf{SKE}}\ E_x)(\hat{k}, m)(r)$: Parse $\hat{k}$ as $\langle k, \tau_{\mathsf{SKE}}^{\mathsf{k}_x} \rangle$. Let $c := \mathsf{SKE.Enc}(k, m; r)$ and return $\langle c, \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}} \rangle$

The $\mathsf{valid}_{\mathsf{SKE}}$ predicate

The predicate $\mathsf{valid}_{\mathsf{SKE}}$ guarantees, that all keys that may be used by the adversary later are generated during initialization (i.e., with the init query). We only allow static corruption of keys, i.e., the adversary has to decide which keys are honest and which are corrupted at this stage. Keys may only be used for encryption and

```
$\mathsf{open}_{\mathsf{SKE}}(c, L)$
    if $c \in [\![\mathcal{T}_{\mathsf{SKE}}]\!] \cap \mathrm{dom}(L)$ then
        return $(c, L(c))$
    else if $c = \langle k, \tau^{\mathsf{k}_x}_{\mathsf{SKE}} \rangle$ then
        return $(c, g^{l(c)}_{\tau^{\mathsf{k}_x}_{\mathsf{SKE}}})$
    else if $c = \langle c', \tau^{\mathsf{ciphertext}}_{\mathsf{SKE}} \rangle$ then
        for each $(\hat{k}, k^h_x()) \in L$ do
            parse $\hat{k}$ as $\langle k, \tau^{\mathsf{k}_x}_{\mathsf{SKE}} \rangle$
            let $m := \mathsf{SKE.Dec}(k, c')$
            if $m \neq \bot$ then
                return $(c, E^{l(c)}_x(\hat{k}, m))$
        return $(c, g^{l(c)}_{\tau^{\mathsf{ciphertext}}_{\mathsf{SKE}}})$
    else
        return $(c, g^{l(c)}_{\top})$
```

Figure 3.14: Open function for secret key encryption.

decryption. This implicitly prevents key cycles. More formally, based on the current trace $\mathbb{T}$ of all parse and generate requests of the adversary, the predicate $\mathsf{valid}_{\mathsf{SKE}}$ returns true only if the following conditions hold:

1. The trace starts with a query "init T, H" (T resp. H may be the empty list). There are no further init queries.

2. The adversary may only generate keys in the init query. Concretely, this is guaranteed by the following rules:

   a) For the query "init T, H", the function symbol k_c may only occur in a term $k^l_c() \in T$. Analogously, k_h may only occur in H. Any label l for $k^l_x()$ must be unique in $T \cup H$.

   b) Any occurrence of $k^l_x()$ in a $\mathsf{generate}$ query must have occurred in the init query. $k^l_x()$ may only occur as the first argument to E_x.

3. The adversary must not use the function symbols for encryption E_x in the init query.

Checking the implementation

We first observe that $\mathcal{I}_{\mathsf{SKE}}$ is collision-free (Definition 41): Basically, collisions for keys can only occur with negligible probability for an IND-CCA secure scheme since they break the security of the scheme. Collision of ciphertexts for the same secret key cannot occur due to the scheme's correctness, collisions of ciphertexts under different keys can only occur with negligible probability for an INT-CTXT secure scheme. Furthermore, it is easy to see that $\mathsf{open}_{\mathsf{SKE}}$ meets the requirements of Definition 42 and that $\mathsf{valid}_{\mathsf{SKE}}$ meets the requirements for valid functions.

3.7.4.4 SKE composability

Theorem 7. *Let $\mathcal{M}$ be a symbolic model and $\mathcal{I}$ a deduction sound implementation of $\mathcal{M}$. If ($\mathcal{M}_{\mathsf{SKE}}$,$\mathcal{I}_{\mathsf{SKE}}$) and ($\mathcal{M}$,$\mathcal{I}$) are compatible (see requirements in Section 3.5)* *and the SKE scheme* $(\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ *is IND-CCA and INT-CTXT secure, then* $\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}$ *is a deduction sound implementation of* $\mathcal{M} \cup \mathcal{M}_{\mathsf{SKE}}$.

Proof. This proof is very similar to that for public key encryption (Theorem 5). The main difference is that the adversary cannot create ciphertexts under honest keys (by $\mathcal{D}_{\mathsf{SKE}}$). Therefore we include an additional game hop to where we add rules of the type $\frac{m}{E_h(k_h^l(),m)}$ to the deduction system. If an adversary notices the difference (i.e., it was able to produce non-DY terms without these rules), we can use it to break the authentication of ciphertexts. Hence this can only happen with negligible probability.

Game 0

In Game 0 $\mathcal{A}$ plays the original deduction soundness game $\mathsf{DS}_{(\mathcal{M} \cup \mathcal{M}_{\mathsf{SKE}}) \cup \mathcal{M}_{\mathsf{tran}}(\nu),(\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu)}(\eta)$.

Game 1

In Game 1 we replace the $\mathsf{generate}$ function by the collision-aware generate function from Figure 3.3. Since $(\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu)$ is a collision-free implementation Game 0 and Game 1 are indistinguishable by Lemma 59.

Game 2

As in Game 2 from Theorem 5 we replace the ciphertexts created under honest keys by encryptions of 0 and the honest keys in the library by random bitstrings. The simulation that Game 1 and Game 2 are indistinguishable works analogously to Theorem 5.

Game 3

In Game 3 we add rules $\frac{m}{E_h^{l_a}(k_h^l(),m)}$ for all honest keys $k_h^l()$ and labels $l_a \in \mathsf{labelsA}$ to the deduction system. This establishes a deduction system similar to that of public key encryption. We show that an adversary that can distinguish Game 2 from Game 3 can be used to break the INT-CTXT security of the encryption scheme. Towards this goal we use the same technique as for the proof of indistinguishability of Games 2 and 3 in Theorem 6[5]. From $\mathcal{A}$ we construct an adversary $\mathcal{B}$ on playing the INT-CTXT game from Figure 3.13 and simulating Game 2 for $\mathcal{A}$. If a bitstring c sent by $\mathcal{A}$ is parsed as a term t such that $S \nvdash_2 t$ but $S \vdash_3 t$, we can (using the same arguments as in Theorem 6) extract a forgery from c.

Game 4

In Game 4 $\mathcal{A}$ interacts with an adversary $\mathcal{B}$ that plays the deduction soundness game for $\mathcal{M}$ and $\mathcal{I}$ and intuitively simulates Game 3 for $\mathcal{A}$. Basically, $\mathcal{B}$ uses transparent functions to add symmetric key encryption to $\mathcal{M}$.

Transparent symbolic model for symmetric key encryption. We first describe the parametrized transparent symbolic model $\mathcal{M}_{\mathsf{SKE}}^{\mathsf{tran}}(\nu)$ and the corresponding parametrized implementation $\mathcal{I}_{\mathsf{SKE}}^{\mathsf{tran}}(\nu)$ $\mathcal{B}$ will use to simulate $\mathcal{I}_{\mathsf{SKE}}$. Analogously to Theorem 5, we use the data types and subtype relation from $\mathcal{M}_{\mathsf{SKE}}$. ν is expected to be an encoding of a list of triples (l, k, k') $(l \in \mathsf{labels}, k \in \{0,1\}^*)$. The signature $\Sigma_{\mathsf{SKE}}^{\mathsf{tran}}$ is the following:

- deterministic $f_{k_x^l()}$ with $\mathrm{ar}(f_{k_x^l()}) = \tau_{\mathsf{SKE}}^{\mathsf{k}_x}$ for all labels $l \in \nu$

[5] There we excluded forged signatures as a way to produce non-DY terms for the adversary.

- randomized $f_{E_h(k_h^l(),0^\ell)}$ with $\mathrm{ar}(f_{E_h(k_h^l(),0^\ell)}) = \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}}$ for all $\ell \in \mathbb{N}$, $l \in \nu$
- randomized $f_{E_h(k_h^l(),\cdot)}$ with $\mathrm{ar}(f_{E_h(k_h^l(),\cdot)}) = \top \to \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}}$ for all $l \in \nu$
- randomized $f_{E_c(k_c^l(),\cdot)}$ with $\mathrm{ar}(f_{E_c(k_c^l(),\cdot)}) = \top \to \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}}$ for all $l \in \nu$

We specify a parametrized implementation $\mathcal{I}_{\mathsf{SKE}}^{\mathsf{tran}}(\nu)$ for $\mathcal{M}_{\mathsf{SKE}}^{\mathsf{tran}}$ as follows for $(l, k, k') \in \nu$:

- $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ f_{k_x^l()})(r)$ returns $\langle k', \tau_{\mathsf{SKE}}^{\mathsf{k}_x} \rangle$
- $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ f_{E_h(k_h^l(),0^\ell)})(r)$ returns $(M_{\mathsf{SKE}}\ E_h)(\langle k, \tau_{\mathsf{SKE}}^{\mathsf{k}_x} \rangle, 0^\ell; r)$
- $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ f_{E_h(k_h^l(),\cdot)})(m; r)$ returns $(M_{\mathsf{SKE}}\ E_h)(\langle k, \tau_{\mathsf{SKE}}^{\mathsf{k}_x} \rangle, m; r)$
- $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ f_{E_c(k_c^l(),\cdot)})(m; r)$ returns $(M_{\mathsf{SKE}}\ E_c)(\langle k, \tau_{\mathsf{SKE}}^{\mathsf{k}_x} \rangle, m; r)$

```
$(M_{\mathsf{SKE}}^{\mathsf{tran}}\ \mathsf{func})(b)$:
    if $b = \langle k', \tau_{\mathsf{SKE}}^{\mathsf{k}_x} \rangle$ for some $(l, k, k') \in \nu$ then
        return $f_{k_x^l()}$
    if $b \in \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}}$ then
        parse $b$ as $\langle c, \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}} \rangle$
        for each $(l, k, k') \in \nu$ do
            let $m := \mathsf{SKE.Dec}(k, c)$
                if $m \neq \bot$ then
                    if $l$ belongs to an honest key then
                        return $f_{E_h(k_h^l(),\cdot)}$
                else
                        return $f_{E_c(k_c^l(),\cdot)}$
        return $\bot$
```

For b with $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ \mathsf{func})(b) = f_{E_h(k_h^l(),\cdot)}$ we have $(l, k) \in \nu$ with $\mathsf{SKE.Dec}(k, c) =: m \neq \bot$ for $b = \langle c, \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}} \rangle$ and define $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ \mathsf{proj}\ f_{E_h(k_h^l(),\cdot)}\ 1)(b) := m$. Analogously for $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ \mathsf{func})(b) = f_{E_c(k_c^l(),\cdot)}$.

Convert terms. Adversary $\mathcal{A}$ uses the function symbols of the original symbolic model for encryption $\mathcal{M}_{\mathsf{SKE}}$. Hence $\mathcal{B}$ needs to map these symbols to the corresponding transparent functions introduced by $\mathcal{B}$. Towards this goal we introduce the function convert as follows:

- $\mathsf{convert}(f^l(t_1, \ldots, t_n)) = f^l(\mathsf{convert}(t_1), \ldots, \mathsf{convert}(t_n))$ for all $f \notin \Sigma_{\mathsf{SKE}}$.
- $\mathsf{convert}(k_x^l()) = f_{k_x^l()}$
- $\mathsf{convert}(E_h^{\hat{l}}(k_h^l(), m)) = f_{E_h(k_h^l(),0^\ell)}^{\hat{l}(m)}()$ if $\hat{l} \in \mathsf{labelsH}$
- $\mathsf{convert}(E_h^{\hat{l}}(k_h^l(), m)) = f_{E_h(k_h^l(),\cdot)}^{\hat{l}}(\mathsf{convert}(m))$ if $\hat{l} \in \mathsf{labelsA}$
- $\mathsf{convert}(E_c^{\hat{l}}(k_c^l(), m)) = f_{E_c(k_c^l(),\cdot)}^{\hat{l}}(\mathsf{convert}(m))$

$\mathcal{B}$ simulates the game $\mathsf{DS}_{(\mathcal{M}\cup\mathcal{M}_{\mathsf{SKE}})\cup\mathcal{M}_{\mathsf{tran}}(\nu'),(\mathcal{I}\cup\mathcal{I}_{\mathsf{SKE}})\cup\mathcal{I}_{\mathsf{tran}}(\nu')}(\eta)$ for $\mathcal{A}$ while playing $\mathsf{DS}_{\mathcal{M}\cup(\mathcal{M}^{\mathsf{tran}}_{\mathsf{SKE}}(\nu)\cup\mathcal{M}_{\mathsf{tran}}(\nu')),\mathcal{I}(\cup\mathcal{I}^{\mathsf{tran}}_{\mathsf{SKE}}(\nu)\cup\mathcal{I}_{\mathsf{tran}}(\nu'))}(\eta)$. Note that we can generically compose $\mathcal{M}^{\mathsf{tran}}_{\mathsf{SKE}}(\nu) \cup \mathcal{M}_{\mathsf{tran}}(\nu')$ to one parametrized transparent model $\mathcal{M}'_{\mathsf{tran}}(\nu||\nu')$ since ν and ν' must be good (analogously for the implementation). However, for the sake of clarity, we keep them apart to distinguish the transparent functions (and parameter) provided by $\mathcal{A}$ from the additional transparent functions introduced by $\mathcal{B}$.

The simulation. $\mathcal{B}$ receives a parameter ν' from $\mathcal{A}$. $\mathcal{B}$ initializes the $\mathbb{T} := \emptyset$ of $\mathcal{A}$'s queries it maintains. Analogously to Theorem 5 $\mathcal{B}$ extracts the keys for SKE from T, H and sets up the parameter ν for $\mathcal{I}^{\mathsf{tran}}_{\mathsf{SKE}}(\nu)$ accordingly. It deals with request exactly as the simulator in Theorem 5.

Claim: Game 3 and Game 4 are indistinguishable

This part is also completely analogous to the corresponding part in Theorem 5.

Claim: If $\mathcal{A}$ wins, then $\mathcal{B}$ wins Game 4

As well analogous to Theorem 5. □

3.7.5 MACs

In this section we show that any deduction sound implementation can be extended by a mac scheme. More precisely, we require a strongly EUF-CMA-secure MAC scheme.

3.7.5.1 Computational preliminaries

Definition 51 (MAC scheme)**.** *A* MAC scheme *is a triple of algorithms* ($\mathsf{MAC.KeyGen}$, $\mathsf{MAC.Mac}$, $\mathsf{MAC.Vfy}$)*.*

The probabilistic key generation algorithm $\mathsf{MAC.KeyGen}$ *takes an encoding of the security parameter and some randomness as inputs and generates a MAC key k.*

The probabilistic algorithm $\mathsf{MAC.Mac}$ *takes three arguments: a MAC key k, the message* $m \in \{0,1\}^*$*, and some randomness* $r \in \{0,1\}^\eta$*. It computes a MAC* $\sigma := \mathsf{MAC.Mac}(k, m; r)$*.*

The verification algorithm $\mathsf{MAC.Vfy}$ *takes a MAC key k, a MAC σ, and a message m as inputs and returns a value from* $\{0,1\}$*. We require* perfect correctness*:*

$$\mathsf{MAC.Vfy}(k, \mathsf{MAC.Mac}(k, m; r), m) = 1$$

for all $r \leftarrow \{0,1\}, m \in \{0,1\}^*$ *and* $k \leftarrow \mathsf{SIG.KeyGen}(1^\eta)$*.*

Definition 52 (strong EUF-CMA security for MAC schemes)**.** *A MAC scheme* ($\mathsf{MAC.KeyGen}$, $\mathsf{MAC.Mac}$, $\mathsf{MAC.Vfy}$) *is* strongly EUF-CMA secure *if for all PPT adversaries* $\mathcal{A}$ *the probability*

$$\mathbb{P}\left[\mathsf{\textit{EUF-CMA-MAC}}_{\mathcal{A}}^{(\mathsf{MAC.KeyGen},\mathsf{MAC.Mac},\mathsf{MAC.Vfy})}(\eta) = 1\right]$$

is negligible. Here, the game $\mathsf{\textit{EUF-CMA-MAC}}$ *is defined analogously to the game* $\mathsf{\textit{EUF-CMA-SIG}}$ *from Figure 3.10 except that no verification key is sent to the adversary and the MAC key is used as signing key.*

3.7.5.2 Symbolic model

We first define the symbolic model $(\mathcal{T}_{\mathsf{MAC}}, \preccurlyeq_{\mathsf{MAC}}, \Sigma_{\mathsf{MAC}}, \mathcal{D}_{\mathsf{MAC}})$ for macs. The signature Σ_{MAC} features the following function symbols:

$$\begin{aligned} k &: \tau^{\mathsf{k}}_{\mathsf{MAC}} \\ mac &: \tau^{\mathsf{k}}_{\mathsf{MAC}} \times \top \to \tau^{\mathsf{mac}}_{\mathsf{MAC}} \end{aligned}$$

for $x \in \{c, h\}$. The randomized function symbol k of arity $\tau^{\mathsf{k}}_{\mathsf{MAC}}$ represents keys. The randomized function symbol mac of arity $\tau^{\mathsf{k}}_{\mathsf{MAC}} \times \top \to \tau^{\mathsf{mac}}_{\mathsf{MAC}}$ represents the mac of a message. To complete the formal definition we set the types

$$\mathcal{T}_{\mathsf{MAC}} := \{\top, \tau^{\mathsf{k}}_{\mathsf{MAC}}, \tau^{\mathsf{mac}}_{\mathsf{MAC}}\}$$

All introduced types are direct subtypes of the base type $\top$ (this defines $\preccurlyeq_{\mathsf{MAC}}$). The deduction system captures the security of macs

$$\mathcal{D}_{\mathsf{MAC}} := \left\{ \frac{mac^{\hat{l}}(k^l(), m)}{m}, \qquad \frac{k^l() \quad m}{mac^{l_a}(k^l(), m)} \right\}$$

These rules are valid for arbitrary labels $l, \hat{l} \in \mathsf{labels}$ and adversarial labels $l_a \in \mathsf{labelsA}$. The following intuitions back up the rules:

- Macs reveal the message that was signed.
- The adversary can use known keys to deduce macs under those keys.

3.7.5.3 Implementation

We now give a concrete implementation $\mathcal{I}_{\mathsf{MAC}}$ for macs. Let ($\mathsf{MAC.KeyGen}$, $\mathsf{MAC.Mac}$, $\mathsf{MAC.Vfy}$) be a MAC scheme. The computable interpretations of k and mac are as follows:

- $(M_{\mathsf{MAC}}\ k)(r)$: Let $k := \mathsf{MAC.Mac}(1^\eta; r)$. Return $\langle k, \tau^{\mathsf{k}}_{\mathsf{MAC}}\rangle$.
- $(M_{\mathsf{MAC}}\ sig)(\hat{k}, m; r)$: Parse $\hat{k}$ as $\langle k, \tau^{\mathsf{k}}_{\mathsf{MAC}}\rangle$. Let $\sigma := \mathsf{MAC.Mac}(k, m; r)$ and return $\langle \sigma, m, \tau^{\mathsf{mac}}_{\mathsf{MAC}}\rangle$.

The $\mathsf{valid}_{\mathsf{MAC}}$ predicate

Based on the current trace $\mathbb{T}$ of all parse and generate requests of the adversary, the predicate $\mathsf{valid}_{\mathsf{MAC}}$ returns true only if the following conditions hold:

1. The trace starts with a query "$\mathsf{init}\ T, H$" (where T and H may be the empty list respectively). There are no further init queries.
2. The adversary may only generate keys in the init query. Concretely, this is guaranteed by the following rules:
   a) For the query "$\mathsf{init}\ T, H$", the function symbol k may only occur in a term $k^h() \in T \cup H$ (i.e., not as subterm of other terms) for $l \in \mathsf{labelsH}$. Any label h for $k^h()$ must be unique in $T \cup H$.
   b) Any occurrence of $k^h()$ in a $\mathsf{generate}$ query must have occurred in the init query.
3. The adversary must not use the function symbol mac in the init query.
4. $k^h()$ may only occur as the first argument for mac.

```
$\mathsf{open}_{\mathsf{MAC}}(c, L)$
    if $c \in [\![\mathcal{T}_{\mathsf{MAC}}]\!] \cap \mathrm{dom}(L)$ then
        return $(c, L(c))$
    else if $c = \langle k, \tau^{\mathsf{k}}_{\mathsf{MAC}} \rangle$ then
        return $(c, g^{l(c)}_{\tau^{\mathsf{k}}_{\mathsf{MAC}}})$
    else if $c = \langle \sigma, m, \tau^{\mathsf{mac}}_{\mathsf{MAC}} \rangle$ then
        for each $(\hat{k}, k^l) \in L$ do
            parse $\hat{k}$ as $\langle k, \tau^{\mathsf{k}}_{\mathsf{MAC}} \rangle$
            if $\mathsf{MAC.Vfy}(k, \sigma, m) = \mathsf{true}$ then
                return $(c, mac^{l(c)}(\hat{k}, m))$
        return $(c, g^{l(c)}_{\tau^{\mathsf{sig}}_{\mathsf{SIG}}})$
    else
        return $(c, g^{l(c)}_{\top})$
```

Figure 3.15: Open function for macs.

3.7.5.4 MAC composability

Theorem 8. *Let $\mathcal{M}$ be a symbolic model and $\mathcal{I}$ be deduction sound implementation of $\mathcal{M}$. If ($\mathcal{M}_{\mathsf{MAC}}$,$\mathcal{I}_{\mathsf{MAC}}$) and($\mathcal{M}$,$\mathcal{I}$) are compatible (see the conditions in Section 3.5), then $\mathcal{I} \cup \mathcal{I}_{\mathsf{MAC}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{MAC}}$ for any $\mathcal{I}_{\mathsf{MAC}}$ constructed from a strong EUF-CMA secure mac scheme.*

Proof. This proof is very similar to Theorem 6.

Game 0

Game 0 is the original deduction soundness game for $\mathcal{I} \cup \mathcal{I}_{\mathsf{MAC}}$.

Game 1

In Game 1 we abort in case of collisions. Game 0 and Game 1 are indistinguishable by Lemma 59 and using the fact that our implementation is collision free.

Game 2

Analogously to Game 2 from Theorem 6 we change the deduction system to prevent the adversary from winning using reconstructed macs in Game 2. Concretely, we add rules that allow to deduce every honestly generated mac that is a subterm of a term t from t. Game 1 and Game 2 are indistinguishable since any $\mathcal{A}$ that notices a difference with non-negligible probability could be used to construct a successful adversary against the deduction soundness of $\mathcal{I}$.

Game 3

In Game 3, analogously to Game 2 from Theorem 6, we change the deduction system to make arbitrary macs deducible. Furthermore we use random bitstrings to represent honestly generated mac-keys in the library. An adversary that can distinguish Game 2 and Game 3 can be used to break the strong EUF-CMA security of the mac scheme. It will either produce a forgery or one of the honest keys.

Game 4

Finally, in Game 4, we simulate Game 3 using transparent functions for macs while playing the deduction soundness game for $\mathcal{I}$. Any adversary winning this game lets

```
$\mathsf{open}_{\mathsf{HASH}}(c, L)$
    if $c \in [\![\mathcal{T}_{\mathsf{HASH}}]\!] \cap \mathrm{dom}(L)$ then
        return $(c, L(c))$
    else if $c = \langle h, \tau_{\mathsf{HASH}} \rangle$ then
        return $(c, g^{l(c)}_{\tau_{\mathsf{HASH}}})$
    else
        return $(c, g^{l(c)}_{\top})$
```

Figure 3.16: Open function for hash functions.

the simulator break the deduction soundness game of $\mathcal{I}$. By requirement this can only happen with negligible probability which concludes our proof. □

3.7.6 Hash functions

In this section we deal with the composition of deduction sound implementations of arbitrary primitives with hash functions. We consider hash functions implemented as random oracles [16]: in this setting calls to the hash function are implemented by calls to a random function which can only be accessed in a black-box way. We model this idea directly in our framework. In the symbolic model model we consider a symbolic function that is randomized and which is implemented by a randomized function. We recover the intuition that hash functions are deterministic by restricting the calls that an adversary can make: for each term t, the adversary can only call the hash function with the honest label $l(t)$.

3.7.6.1 Symbolic model

The symbolic model for hash functions is rather standard. It is given by the tuple $(\mathcal{T}_{\mathsf{HASH}}, \preccurlyeq_{\mathsf{HASH}}, \Sigma_{\mathsf{HASH}}, \mathcal{D}_{\mathsf{HASH}})$ where

$$\mathcal{T}_{\mathsf{HASH}} := \{\top, \tau_{\mathsf{HASH}}\}$$

and $\tau_{\mathsf{HASH}} \preccurlyeq_{\mathsf{HASH}} \top$. The signature Σ_{HASH} contains only a randomized function $H : \top \to \tau_{\mathsf{HASH}}$ characterized by the deduction rule:

$$\mathcal{D}_{\mathsf{HASH}} := \left\{ \frac{m}{H^l(m)} \right\}$$

where $l \in \mathsf{labelsH}$.

3.7.6.2 Implementation

The implementation $\mathcal{I}_{\mathsf{HASH}}$ for hash functions is via a randomized function: when called, the function simply returns a random value, and we will require that it does so consistently; Concretely $(M_{\mathsf{HASH}}\ H)(m; r)$ returns $\langle r, \tau_{\mathsf{HASH}} \rangle$.

The open function for hash functions is described in Figure 3.16. If the bitstring to be opened was not the result of a $\mathsf{generate}$ call, then it returns garbage of types either τ_{HASH} or $\top$, depending on what c encodes. Otherwise, it will return the entry in L that corresponds to c: by the requirements posed by $\mathsf{valid}_{\mathsf{HASH}}$ below this will be $H^{l(t)}(m)$ for some bitstring m with $L[[m]] = t$.

A useful observation is that by the description above, the library L will never contain an entry of the form $(c, H^l(m))$ for some adversarial label $l \in \mathsf{labelsA}$; moreover, if $(c, H^l(m))$ is in L, then $l = l(t)$ for some t, and $L[[m]] = t$.

The $\mathsf{valid}_{\mathsf{HASH}}$ predicate

For simplicity we require that no hash is present in init requests (our results easily extend to the case where this restriction is not present). In addition we use the predicate $\mathsf{valid}_{\mathsf{HASH}}$ to enforce deterministic behavior of our hash implementation. We require that for any term t, all occurrences of $H(t)$ in $\mathsf{generate}$ and $\mathsf{sgenerate}$ requests use the same label. Concretely, we demand that for any term t, all generate requests for $H^{\hat{l}}(t)$ are labeled with the honest label $\hat{l} = l(t)$. The choice of label is not important: we could alternatively request that if $H^{l_1}(t)$ and $H^{l_2}(t)$ occur in a generate requests, then $l_1 = l_2$.

3.7.6.3 Hash composability

Theorem 9. *Let $\mathcal{I}$ be a deduction sound implementation of $\mathcal{M}$. If $(\mathcal{M}_{\mathsf{HASH}}, \mathcal{I}_{\mathsf{HASH}})$ and $(\mathcal{M}, \mathcal{I})$ are compatible, then $\mathcal{I} \cup \mathcal{I}_{\mathsf{HASH}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{HASH}}$ in the random oracle model.*

The intuition behind this proof is simple: collisions due to tagging occur only with probability given by the birthday bound (so with negligible probability). Given an adversary that wins the deduction soundness game for the composed libraries, we construct an adversary that breaks deduction soundness of $(\mathcal{M}, \mathcal{I}, \mathsf{valid}_{\mathcal{I}})$. This latter adversary simulates the hash function via a randomized transparent function with no arguments: a generate $H^{l(t)}(t)$ call will be implemented by a generate call to $f^{l(t)}()$. Due to $\mathsf{valid}_{\mathsf{HASH}}$ the knowledge set S does not contain any occurrence of H with a dishonest label, hence the only "useful" deduction soundness rule which allows the adversary to learn/manipulate terms with dishonest labels are not applicable (we can cut them out of any deduction).

Proof. Consider an adversary $\mathcal{A}$ that breaks deduction soundness of implementation $\mathcal{I} \cup \mathcal{I}_{\mathsf{HASH}}$ for $\mathcal{M} \cup \mathcal{M}_{\mathsf{HASH}}$, i.e.

$$\mathbb{P}\left[\mathsf{DS}_{(\mathcal{M}\cup\mathcal{M}_{\mathsf{HASH}})\cup\mathcal{M}_{\mathsf{tran}}(\nu),(\mathcal{I}\cup\mathcal{I}_{\mathsf{HASH}})\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta) = 1\right]$$

is non-negligible for some some choice of $\mathcal{M}_{\mathsf{tran}}, \mathcal{I}_{\mathsf{tran}}$. We consider the transparent model/implementation $\mathcal{M}'_{\mathsf{tran}}, \mathcal{I}'_{\mathsf{tran}}$ obtained by adding to the functions in $\mathcal{M}_{\mathsf{tran}}$ a new (randomized) function f_H of arity 0; the implementation of the function is given by M_{f_H} defined by: $(M_{f_H}\ f_H)(r) = \langle r, \mathsf{HASH}\rangle$, i.e. the machine that simply outputs a proper encoding of its random coins.

We next show that adversary $\mathcal{A}$ yields an adversary $\mathcal{B}$ that contradicts the deduction soundness of $\mathcal{I}$ with respect to $\mathcal{M}$ when the transparent model/implementation is $(\mathcal{M}'_{\mathsf{tran}}, \mathcal{I}'_{\mathsf{tran}})$ defined above. Adversary $\mathcal{B}$ that we construct translates the queries of $\mathcal{A}$ into queries for $(\mathcal{M} \cup \mathcal{M}'_{\mathsf{tran}}, \mathcal{I} \cup \mathcal{I}'_{\mathsf{tran}})$ by using f_H to implement the hash function. This is accomplished using a conversion function $\mathsf{convert}$ from terms in $\mathcal{M} \cup \mathcal{M}_{\mathsf{HASH}} \cup \mathcal{M}_{\mathsf{tran}}$ to terms in $\mathcal{M} \cup \mathcal{M}'_{\mathsf{tran}}$.

- $\mathsf{convert}(f^l(t_1, \ldots, t_n)) = f^l(\mathsf{convert}(t_1), \ldots, \mathsf{convert}(t_n))$ for all $f \neq H$.
- $\mathsf{convert}(H^{l(t)}(t)) = f_H^{l(t)}$

The inverse of the convert function is defined in the obvious way. These conversion of terms will still preserve the validity of $\mathcal{B}$'s trace for every valid trace of $\mathcal{A}$ due to requirement (i) for valid predicates.

Adversary $\mathcal{B}$ processes the queries of $\mathcal{A}$ as follows.

init query. $\mathcal{B}$ forwards the init request to his game and forwards the answer to $\mathcal{A}$.

generate queries. For each request "$\mathsf{generate}\ t$": for any t' such that $H^l(t') \in st(t)$ adversary $\mathcal{B}$ issues "$\mathsf{sgenerate}\ \mathsf{convert}(t')$" (for convenience, we assume the order of these requests is in bottom up manner). These queries are valid by requirement (ii) for valid predicates. It then issues "$\mathsf{generate}\ \mathsf{convert}(t)$" and returns the answer to this last query to $\mathcal{A}$. The additional $\mathsf{sgenerate}$ queries are necessary to preserve an invariant on the libraries needed to show the indistinguishability of the real game and the simulation (see indistinguishability of Game 2 and Game 3 in Theorem 5).

$\mathcal{B}$ proceeds analogously for $\mathsf{sgenerate}$ requests (but no answer is returned to $\mathcal{A}$).

parse queries. For each request "$\mathsf{parse}\ c$" $\mathcal{B}$ sends "$\mathsf{parse}\ c$" to its game and receives a term t. $\mathcal{B}$ sends $\mathsf{convert}^{-1}(t)$ to $\mathcal{A}$.

We conclude by arguing that if $\mathcal{A}$ is successful, then so is $\mathcal{B}$. Let $\mathsf{Terms}_1 = \mathsf{Terms}(\Sigma \cup \Sigma_{\mathsf{HASH}} \cup \Sigma_{\mathsf{tran}})$ and $\mathsf{Terms}_2 = \mathsf{Terms}(\Sigma \cup \Sigma'_{\mathsf{tran}})$. Let $\vdash_1$ be the deduction system defined by $\mathcal{D} \cup \mathcal{D}_{\mathsf{HASH}} \cup \mathcal{D}_{\mathsf{tran}}$, and let $\vdash_2$ the one defined by $\mathcal{D} \cup \mathcal{D}_{\mathsf{tran}}$. Let $\mathcal{R} : \mathsf{Terms}_2 \to \{0,1\}^\eta$ be an arbitrary randomness assignment and $r_\mathcal{A}$ be arbitrary random coins for $\mathcal{A}$. Then adversary $\mathcal{B}$ simulates for $\mathcal{A}$ the game

$$\mathsf{DS}_{(\mathcal{M} \cup \mathcal{M}_{\mathsf{HASH}}) \cup \mathcal{M}_{\mathsf{tran}}(\nu), (\mathcal{I} \cup \mathcal{I}_{\mathsf{HASH}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu), \mathcal{A}}$$

here the coins of adversary $\mathcal{A}$ are $r_\mathcal{A}$, and the randomness assignment $\mathcal{R}_1 : \mathsf{Terms}_1 \to \{0,1\}^\eta$ is defined by $\mathcal{R}_1(t) = \mathcal{R}_2(\mathsf{convert}(t))$.

In addition, if $L_2(\mathcal{R}_2, r_\mathcal{A})$ is the mapping maintained in $\mathsf{DS}_{\mathcal{M} \cup \mathcal{M}'_{\mathsf{tran}}, \mathcal{I} \cup \mathcal{I}'_{\mathsf{tran}}, \mathcal{B}}(\eta)$ then $(c, t) \in L_1$ if and only if $(c, \mathsf{convert}(t)) \in L_2$.

Next we show that if $\mathsf{parse}(\mathsf{convert}(t))$ is a Dolev-Yao request by $\mathcal{B}$, then $\mathsf{convert}(t)$ is a Dolev-Yao request by $\mathcal{A}$. This implies that if $\mathcal{A}$ is non Dolev-Yao, then so is $\mathcal{B}$.

Consider an arbitrary $\mathsf{parse}(c)$ request by $\mathcal{A}$, and let S be the set of terms present in all of the generate requests of $\mathcal{A}$. Per our construction, $\mathsf{convert}(S)$ is the set of terms in the generate requests of $\mathcal{B}$ (where $\mathsf{convert}$ is extended from terms to sets of terms in the obvious way). Assume there exists a proof $\mathsf{convert}(S) = S'_0 \overset{\alpha_1}{\to} S'_1 \overset{\alpha_2}{\to} S'_2 \ldots \overset{\alpha_n}{\to} S'_n$ with $\mathsf{convert}(t) \in S'_n$ for $\mathsf{convert}(S) \vdash_2 \mathsf{convert}(t)$. We show we can construct a proof for $S \vdash_1 t$.

By a previous remark, S and t do not contain any occurence of H^l with an adversarial label l. The only way to introduce instances of f_H labeled with an adversarial label is to use the rule instantiation $\frac{}{f_H^l}$. Assume that for some $i \in \{1, 2, \ldots, n\}$ we have $S'_{i-1} \overset{\alpha_i}{\to} S'_i$ and α_i is the rule $\frac{}{f_H^l}$ for some adversarial label l. To eliminate the use of the rule let t be an arbitrary term in S, and consider the subsitution θ that replaces f_H^l with $\mathsf{convert}(t)$. Then $\mathsf{convert}(S) = S'_0 \overset{\alpha_1}{\to} \theta(S'_1) \overset{\alpha_2}{\to} \theta(S'_2) \ldots \overset{\alpha_{i-1}}{\to} \theta(S_{i-1}) \overset{\alpha_{i+1}}{\to} \theta(S_{i+1}) \ldots \overset{\alpha_n}{\to} \theta(S'_n)$ is a valid derivation for $\mathsf{convert}(t)$ which does not use the rule. Iteratively, we obtain a derivation $\mathsf{convert}(S) = S'_0 \overset{\alpha_1}{\to} S''_1 \ldots \alpha_m {\to} S''_m$ for $\mathsf{convert}(t)$ and if f_H^l occurs in any set, then $l = l(t)$ and is an honest label. We can therefore apply $\mathsf{convert}^{-1}$ to the above proof to obtain a proof for $S \vdash_1 t$. Hence $\mathcal{B}$ wins if $\mathcal{A}$ wins. □

3.8 Forgetfulness

All the theorems from Section 3.7 have one important drawback: Key material cannot be sent around as the valid predicates forbid keys from being used in non-key positions. This takes the analysis of a large class of practical protocols (e.g, many key exchange protocols) outside the scope of our results. The problem is that deduction soundness does not guarantee that no information about non-DY terms is leaked by the computational implementation. E.g., we could think of a deterministic function symbol f that takes arguments of type nonce with only the rule $\frac{n^l()}{f(n^l())}$. An implementation of f could leak half of the bits of its input and still be sound. However, to send key material around, we need to rely on the fact that information theoretically nothing is leaked about the suitable positions for keys.

To solve this problem, we introduce *forgetful* symbolic models and implementations. A forgetful symbolic models features function symbols with positions that are marked as being forgetful. The corresponding implementation has to guarantee, that no information about the arguments at these positions will be leaked (except their length). We will formalize this intuition later in Definition 54. We start off by introducing some necessary extensions of our previous setting to allow for the concept of forgetfulness.

3.8.1 Preliminaries

We need to extend some definitions to capture the concept of forgetfulness.

Changed hybrid terms for function symbols with forgetful arguments

To allow the handling of forgetful positions, extend the definition for hybrid terms with function symbols carrying an honest label in the library. Let f be a function symbol of arity $\text{ar}(f) = \tau_1 \times \ldots \times \tau_n \rightarrow \tau$. Then a hybrid term of f may be $f^l(a_1, \ldots, a_n)$ where each a_i is either a bitstring from $[\![\tau_i]\!]$ or a term of type τ_i for forgetful positions i. For normal positions a_i must be a bitstring from $[\![\tau_i]\!]$ as usual. The definitions for the completeness of a library L and $L[[c]]$ are changed accordingly.

New valid requirements

To allow forgetful arguments to be useful, we have to change the definition of valid requirements. Concretely, we allow the behavior of valid to additionally depend on a signature Σ_{valid} that features forgetful positions, i.e., positions of function symbols in $f \in \Sigma_{\mathsf{valid}}$ may be marked as forgetful. We then restate the requirements for valid as follows:

(i) If $\mathsf{valid}(\mathbb{T} + q) = \mathsf{true}$, then $\mathsf{valid}(\mathbb{T} + \hat{q}) = \mathsf{true}$ where $\hat{q}$ is a *variation* of q: If $q =$“$\mathsf{generate}\ t$”, then $\hat{q} =$“$\mathsf{generate}\ \hat{t}$” (analogously for “$\mathsf{sgenerate}\ t$”). Here, $\hat{t}$ is a variation of t according to the following rule: Any subterm $f^l(t_1, \ldots, t_n)$ of t where $f \notin \Sigma \cup \Sigma_{\mathsf{valid}}$ is a foreign function symbol may be replaced by $\hat{f}^{\hat{l}}(\hat{t}_1, \ldots, \hat{t}_m)$ where $\hat{f} \notin \Sigma \cup \Sigma_{\mathsf{valid}}$ is a foreign function symbol and $\hat{t}_i = t_j$ for some $j \in \{1, \ldots, n\}$ (where each t_j may only be used once) or $\hat{t}_i$ does not contain function symbols from $\Sigma \cup \Sigma_{\mathsf{valid}}$. As a special case we may also replace $f^l(t_1, \ldots, t_n)$ with a term $\hat{t}_1$ (i.e., $\hat{f}$ is “empty”). If $q =$ ‘“$\mathsf{init}\ T, H$” then $\hat{q} =$“$\mathsf{init}\ \hat{T}, \hat{H}$” where $T = (t_1, \ldots, t_n)$ and $\hat{T} = (\hat{t}_1, \ldots, \hat{t}_n)$ and $\hat{t}_i$ is a variation of t_i ($\hat{H}$ analogously).

(ii) If $\mathsf{valid}(\mathbb{T} + q) = \mathsf{true}$ and t is a term occurring in q, then $\mathsf{valid}(\mathbb{T}+$“$\mathsf{sgenerate}\ t'$“$) = \mathsf{true}$ for any subterm t' of t that is not a subterm at a forgetful position.

(iii) $\mathsf{valid}(\mathbb{T})$ can be evaluated in polynomial time (in the length of the trace $\mathbb{T}$).

Basically, valid is now allowed to make statements about how the own fuction symbols (from Σ) are allowed to be used in the context of some foreign function symbols

(Σ_{valid}) with forgetful positions. Consequently, we do require that a trace remains valid if those function symbols are replaced (see new requirement (i)). Furthermore, we do not require valid to allow for silent generation of subterms at forgetful positions because it might be essential that those subterms are never generated (see new requirement (ii)).

3.8.2 Forgetful symbolic models and implementations

We say that a symbolic model $\mathcal{M}$ is a *forgetful symbolic model* if arguments of a function symbol may be marked as *forgetful.* In order to formalize forgetful implementations, the computational counterpart of forgetful positions, we introduce the notion of an oblivious implementation. These are implementations for symbolic functions which can take as input natural numbers instead of actual bitstrings of the appropriate sort.

Definition 53 (oblivious implementation)**.** *Let $\mathcal{M}$ be a forgetful symbolic model. $\overline{\mathcal{I}} = (\overline{M}, [\![\cdot]\!], len, \mathsf{open}, \mathsf{valid})$ is an* oblivious implementation of $\mathcal{M}$ *if $\overline{\mathcal{I}}$ is an implementation of $\mathcal{M}$ with a slightly changed signature: For each function symbol $f \in \Sigma$ with arity $\mathrm{ar}(f) = \tau_1 \times \ldots \times \tau_n \to \tau$ the signature of $(\overline{M}\ f)$ is $\theta(\tau_1) \times \cdots \times \theta(\tau_n) \times \{0,1\}^\eta \to [\![\tau]\!]$ where $\theta(\tau_i) = \mathbb{N}$ if the ith argument of f is forgetful and $[\![\tau_i]\!]$ otherwise.*

Intuitively, oblivious implementations for all forgetful positions, take as input natural numbers; these will be the length of the actual inputs on the forgetful positions.

As indicated above, a forgetful implementation is one which is indistinguishable from an oblivious implementation. To formally define the notion we introduce a distinguishing game $\mathsf{FIN}^b_{\mathcal{M}(\nu),\mathcal{I}(\nu),\overline{\mathcal{I}}(\nu),\mathcal{A}}(\eta)$ where an adversary $\mathcal{A}$ tries to distinguish between the case when he interacts with the real implementation, or with an alternative implementation that is oblivious with respect to all of the forgetful arguments. We say that an implementation is forgetful, if there exists an oblivious implementation such that no adversary succeeds in this task.

Definition 54 (forgetful implementation)**.** *We say that an implementation $\mathcal{I} = (M, [\![\cdot]\!], len, \mathsf{open}, \mathsf{valid})$ is a* forgetful implementation of a forgetful symbolic model $\mathcal{M}$ *if there is an oblivious implementation $\overline{\mathcal{I}} = (\overline{M}, [\![\cdot]\!], len, \mathsf{open}, \mathsf{valid})$ such that for all for all parametrized transparent symbolic models $\mathcal{M}_{\mathsf{tran}}(\nu)$ and for all parametrized transparent implementations $\mathcal{I}_{\mathsf{tran}}(\nu)$ of $\mathcal{M}_{\mathsf{tran}}(\nu)$ compatible with ($\mathcal{M}$,$\mathcal{I}$) we have that*

$$\begin{aligned}&Prob[\mathsf{FIN}^0_{\mathcal{M}\cup\mathcal{M}_{\mathsf{tran}}(\nu),\mathcal{I}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\overline{\mathcal{I}}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta) = 1]\\ -&Prob[\mathsf{FIN}^1_{\mathcal{M}\cup\mathcal{M}_{\mathsf{tran}}(\nu),\mathcal{I}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\overline{\mathcal{I}}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta) = 1]\end{aligned}$$

is negligible for every PPT adversary $\mathcal{A}$.

Lemma 62. *Let $\mathcal{M}$ be an forgetful symbolic model, $\mathcal{I}$ be an forgetful implementation of $\mathcal{M}$ and $\overline{\mathcal{I}}$ a corresponding oblivious implementation. If $\mathcal{I}$ is deduction sound, then $\overline{\mathcal{I}}$ is deduction sound with respect to the deduction soundness game DS' that uses* $\mathsf{generate}^{\mathsf{FIN}}$ *(Figure 3.17) instead of* $\mathsf{generate}$.

Proof. Let $\mathcal{A}$ be a PPT adversary that wins the deduction soundness game for $\overline{\mathcal{I}}$ with non-negligible probability. We construct an adversary $\mathcal{B}$ that plays the game

$$\mathsf{FIN}^b_{\mathcal{M}\cup\mathcal{M}_{\mathsf{tran}}(\nu),\mathcal{I}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\overline{\mathcal{I}}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta)$$

```
$\mathsf{generate}^{\mathsf{FIN}}_{M,\mathcal{R}}(t, L)$:
    if for some $c \in \mathrm{dom}(L)$ we have $L[[c]] = t$ then
        return $c$
    else
        for $i \in \{1, n\}$ do
            if $i$ is a forgetful argument then
                let $c_i := \mathit{len}(t_i)$
                let $a_i := t_i$
            else
                let $(c_i, L) := \mathsf{generate}_{M,\mathcal{R}}(t_i, L)$
                let $a_i := c_i$
        let $r := \mathcal{R}(t)$
        let $c := (M\ f)(c_1, \dots, c_n; r)$
        let $L(c) := f^l(a_1, \dots, a_n))$ ($l \in \mathsf{labelsH}$)
        return $(c, L)$
```

Figure 3.17: The generate function for an oblivious implementation (t is of the form $f^l(t_1, \dots, t_n)$ (with possibly $n = 0$ and no label l for deterministic function symbols f)). The requirements for the input t are those of the normal $\mathsf{generate}$ function.

and simulates the deduction soundness game for $\mathcal{A}$ (by just relaying the queries of $\mathcal{A}$). Depending on b, this is a perfect simulation of

$$\mathbb{P}\left[\mathsf{DS}_{\mathcal{M} \cup \mathcal{M}_{\mathsf{tran}}(\nu), \mathcal{I} \cup \mathcal{I}_{\mathsf{tran}}(\nu), \mathcal{A}}(\eta) = 1\right]$$

or of the variant of the game for $\overline{\mathcal{I}}$

$$\mathbb{P}\left[\mathsf{DS}'_{\mathcal{M} \cup \mathcal{M}_{\mathsf{tran}}(\nu), \overline{\mathcal{I}} \cup \mathcal{I}_{\mathsf{tran}}(\nu), \mathcal{A}}(\eta) = 1\right]$$

If $\mathcal{A}$ wins the deduction soundness game, $\mathcal{B}$ wins its game as well. Otherwise, i.e., if $\mathcal{A}$ is invalid $\mathcal{B}$ picks a random bit b and sends "guess b" to its game. Since $\mathcal{I}$ is deduction sound, $\mathcal{A}$ will only win the first game with negligible probability. If $\mathcal{A}$ wins the game for $\overline{\mathcal{I}}$ with non-negligible probability, $\mathcal{B}$ has a non-negligible advantage. This contradicts the assumption that $\overline{\mathcal{I}}$ is an oblivious implementation corresponding to $\mathcal{I}$. □

Let $\overline{\mathcal{M}_{\mathsf{PKE}}}$ be the forgetful symbolic model derived from the symbolic symbolic model $\mathcal{M}_{\mathsf{PKE}}$ from Section 3.7.2 by marking the message m for honest encryptions $\mathit{enc}_h(\mathit{ek}, m)$ as forgetful. Then Lemma 63 capture the intuition that public key encryption schemes are forgetful with respect to their messages.

Lemma 63. *$\mathcal{I}_{\mathsf{PKE}}$ from Section 3.7.2 is a forgetful implementation of $\overline{\mathcal{M}_{\mathsf{PKE}}}$.*

Proof. We define an oblivious implementation $\overline{\mathcal{I}_{\mathsf{PKE}}}$ with the Turing machine $\overline{M_{\mathsf{PKE}}}$ that differs only for the function symbol enc_h from M_{PKE}. We set $(\overline{M_{\mathsf{PKE}}}\ \mathit{enc}_h)(\mathit{ek}, \ell; r) := (M_{\mathsf{PKE}}\ \mathit{enc}_h)(\mathit{ek}, 0^\ell; r)$. $\overline{\mathcal{I}_{\mathsf{PKE}}}$ witnesses that $\mathcal{I}_{\mathsf{PKE}}$ is a forgetful implementation of $\overline{\mathcal{M}_{\mathsf{PKE}}}$.

```
$\mathsf{FIN}^b_{\mathcal{M}(\nu),\mathcal{I}(\nu),\overline{\mathcal{I}}(\nu),\mathcal{A}}(\eta)$:
    let $S := \emptyset$        (set of requested terms)
    let $L := \emptyset$        (library)
    let $\mathbb{T} := \emptyset$        (trace of queries)
    $\mathcal{R} \leftarrow \{0,1\}^*$    (random tape)

    if $b = 0$ then let $\mathsf{generate} := \mathsf{generate}^{\mathsf{FIN}}_{\overline{M},\mathcal{R}}$ else let $\mathsf{generate} := \mathsf{generate}_{M,\mathcal{R}}$
    Receive parameter $\nu$ from $\mathcal{A}$

    on request "init $T, H$" do
        add "init $T$" to $\mathbb{T}$
        if $\mathsf{valid}(\mathbb{T})$ then
            let $S := S \cup T$
            let $C := \emptyset$    (list of replies)
            for each $t \in T$ do
                let $(c, L) := \mathsf{generate}(t, L)$
                let $C := C \cup \{c\}$
            for each $t \in H$ do
                let $(c, L) := \mathsf{generate}(t, L)$
            send $C$ to $\mathcal{A}$
        else
            return 0 ($\mathcal{A}$ is invalid)

    on request "sgenerate $t$" do
        if $\mathsf{valid}(\mathbb{T}$+"sgenerate $t$") then
            let $(c, L) := \mathsf{generate}(t, L)$

    on request "generate $t$" do
        add "generate $t$" to $\mathbb{T}$
        if $\mathsf{valid}(\mathbb{T})$ then
            let $S := S \cup \{t\}$
            let $(c, L) := \mathsf{generate}(t, L)$
            send $c$ to $\mathcal{A}$
        else
            return 0 ($\mathcal{A}$ is invalid)

    on request "parse $c$" do
        let $(t, L) := \mathsf{parse}(c, L)$
        if $S \vdash_{\mathcal{D}} t$ then
            send $t$ to $\mathcal{A}$
        else
            return 1 ($\mathcal{A}$ produced non-Dolev-Yao term)

    on request "guess $b'$" do
        if $b = b'$ then
            return 1 ($\mathcal{A}$ wins)
        else
            return 0 ($\mathcal{A}$ looses)
```

Figure 3.18: Indistinguishability game for forgetful implementations.

Let $\mathcal{A}$ be a PPT adversary such that the probability from Definition 54 is non-negligible. We can then use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that wins the IND-CCA game from Figure 3.8 with non-negligible probability. $\mathcal{B}$ simulates

$$\mathsf{FIN}^{b}_{\overline{\mathcal{M}_{\mathsf{PKE}}}\cup\mathcal{M}_{\mathsf{tran}}(\nu),\mathcal{I}_{\mathsf{PKE}}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\overline{\mathcal{I}_{\mathsf{PKE}}}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta)$$

for $\mathcal{A}$ where the bit b corresponds the the bit picked by the IND-CCA game from Figure 3.8 ($b = 0$: produce encryptions of 0, $b = 1$ produce encryptions of the real messages). The simulation works analogously to Game 2 in Theorem 5. Since $\mathcal{B}$ does not know the encryption keys while playing the IND-CCA game, we need to randomize them in the library. The arguments from the proof of indistinguishability of Game 1 and Game 2 in Theorem 5 can be easily translate to the setting at hand and show that the simulation, although not perfect, is indistinguishable from FIN^0 and FIN^1 respectively. Hence, $\mathcal{B}$ would break the IND-CCA security of the public key encryption scheme if such an adversary $\mathcal{A}$ would existed. □

3.8.3 Sending keys around

To be able to consider the case when symmetric keys are sent encrypted we introduce an extension of the model for symmetric key encryption of Section 3.7.4. The extension is that the $\mathsf{valid}_{\mathsf{SKE}}$ predicate can now depend on a signature Σ_{valid} that contains functions with forgetful positions. The new predicate allows for standard generation of keys for symmetric encryption (with the same restrictions as those in Section 3.7.4), but in addition it also allows for generate requests that contain occurrences of symmetric keys under functions from signature Σ_{valid}, as long as the occurrences are on forgetful positions.

Concretely, based on $\mathcal{I}_{\mathsf{SKE}}$ from Section 3.7.4 we introduce the implementation $\mathcal{I}_{\mathsf{SKE}}[\Sigma_{\mathsf{valid}}]$ for a signature Σ_{valid} featuring forgetful positions. We define the $\mathsf{valid}_{\mathsf{SKE}}$ predicate based on Σ_{valid} and, instead of requirement (2) for $\mathsf{valid}_{\mathsf{SKE}}$, now require:

1. For the query "$\mathsf{init}\ T, H$", the function symbol k_c may only occur in a term $k_c^l() \in T$. Analogously, k_h may only occur in H. Any label l for $k_x^l()$ must be unique in $T \cup H$.

2. Any occurrence of $k_x^l()$ in a $\mathsf{generate}$ query must have occurred in the init query. $k_x^l()$ may only occur as the first argument to E_x or as a subterm of a forgetful position for a function symbol $f \in \Sigma_{\mathsf{valid}}$.

We show in Theorem 10, that we can compose our extended implementation $\mathcal{I}_{\mathsf{SKE}}[\Sigma_{\mathsf{valid}}]$ (extended in the sense that its valid predicate allows for more scenarios) with any deduction sound forgetful implementation and preserve deduction soundness. Since the implementation for public key encryption $\mathcal{I}_{\mathsf{PKE}}$ from Section 3.7.2 is a forgetful implementation for the forgetful symbolic model $\overline{\mathcal{M}_{\mathsf{PKE}}}$ by Lemma 63, queries like "$\mathsf{generate}\ enc_h^{\hat{l}}(ek_h^l(), k_h^{\tilde{l}}())$" are now possible. Intuitively, this corresponds to sending around symmetric keys encrypted under asymmetric keys in a protocol.

Furthermore, we show that, in the case of secret key encryption, forgetfulness is preserved as well (Theorem 11). This even holds for the obvious forgetful symbolic model of secret key encryption where the message position for honest encryptions under honest keys is a forgetful one. I.e., we could add several layers of secret key encryption to allow for the encryption of symmetric keys under other symmetric keys.

The last aspect shows why we need to fix the set of function symbols Σ_{valid} at the time of composition: We cannot allow to encrypt keys under forgetful positions in general since it would be impossible for $\mathsf{valid}_{\mathsf{SKE}}$ to detect key cycles. E.g., assume that Σ_{valid} contains a function symbol f with a forgetful second position. Do the terms $f^{\hat{l}}(t', k_h^l())$ and $E_h^{\tilde{l}}(k_h^l(), t')$ contain a key cycle? We cannot tell without knowing the implementation of f and t'. Therefore we have to require that the valid predicate of the implementation we are composing $\mathcal{I}_{\mathsf{SKE}}$ with does rely on the forgetfulness of function symbols from Σ_{SKE} in Theorem 10.

Theorem 10. *Let $\mathcal{M}$ be a forgetful symbolic model and $\mathcal{I}$ be a forgetful deduction sound implementation of $\mathcal{M}$. $\mathcal{I}_{\mathsf{SKE}}$ denotes $\mathcal{I}_{\mathsf{SKE}}[\Sigma]$ where Σ is the signature from $\mathcal{M}$. If $(\mathcal{M}_{\mathsf{SKE}},\mathcal{I}_{\mathsf{SKE}})$ and $(\mathcal{M},\mathcal{I})$ are compatible (see requirements in Section 3.5) and the valid predicate of $\mathcal{I}$ does not depend on function symbols from Σ_{SKE}, then $\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{SKE}}$.*

Proof. This proof is very similar to that for Theorem 7. Basically, we just introduce an additional game hop where we replace $\mathcal{I}$ by an oblivious implementation $\overline{\mathcal{I}}$. This guarantees that, even if the adversary requests to generate a term t with honest keys at forgetful positions, the bitstring interpretation of those keys are not used to compute the bitstring corresponding to t. We can then follow the strategy from the proof for Theorem 7 and replace honest keys in the library with random bitstrings.

Game 0

In Game 0 $\mathcal{A}$ plays the original deduction soundness game $\mathsf{DS}_{(\mathcal{M}\cup\mathcal{M}_{\mathsf{SKE}})\cup\mathcal{M}_{\mathsf{tran}}(\nu),(\mathcal{I}\cup\mathcal{I}_{\mathsf{SKE}})\cup\mathcal{I}_{\mathsf{tran}}(\nu)}(\eta)$.

Game 1

In Game 1 we replace the implementation $\mathcal{I}$ with a corresponding oblivious implementation $\overline{\mathcal{I}}$ (which exists since $\mathcal{I}$ is a forgetful implementation according to Definition 54). Note that $\overline{\mathcal{I}}$ must be composable with $\mathcal{I}_{\mathsf{SKE}}$ since $\mathcal{I}$ is composable with $\mathcal{I}_{\mathsf{SKE}}$. For this to work we also have to replace the $\mathsf{generate}$ function by $\mathsf{generate}^{\mathsf{FIN}}$ from Figure 3.17.

Claim: Game 0 and Game 1 are indistinguishable

Basically, this indistinguishability holds due to the fact that $\mathcal{I}$ is a forgetful implementation. Let $\mathcal{A}$ be a distinguisher between Game 0 and Game 1. Then we construct an adversary $\mathcal{B}$ that plays the game

$$\mathsf{FIN}^b_{\mathcal{M}\cup\mathcal{M}'_{\mathsf{tran}}(\nu'),\mathcal{I}\cup\mathcal{I}'_{\mathsf{tran}}(\nu'),\overline{\mathcal{I}}\cup\mathcal{I}'_{\mathsf{tran}}(\nu'),\mathcal{B}}(\eta)$$

and simulates Game 0 or Game 1 for $\mathcal{A}$ (depending on the value of b). $\mathcal{B}$ simulates $\mathcal{I}_{\mathsf{SKE}}$ using transparent functions (as a part of $\mathcal{M}'_{\mathsf{tran}}(\nu')$ together with $\mathcal{M}_{\mathsf{tran}}(\nu')$. $\mathcal{B}$ checks the DY-ness of $\mathcal{A}$'s requests with respect to Game 1. Note that the simulation is perfect since $\mathcal{B}$ can know all generate all the keys and does not need to hide any arguments when simulating $\mathcal{I}_{\mathsf{SKE}}$ with transparent functions. If $\mathcal{A}$ can distinguish Game 0 from Game 1, $\mathcal{B}$ can break the indistinguishability of the oblivious implementation according to Definition 54. This can only happen with negligible probability.

Game 2

In Game 2 we replace the $\mathsf{generate}^{\mathsf{FIN}}$ function with a collision-aware variant (similar to Figure 3.3. The indistinguishability is guaranteed analogously to Theorem 7.

Game 3

Game 3 is analogous to Game 2 from Theorem 7: We replace honest encryptions under honest keys by encryptions of 0 and replace honest encryption keys in the library by random bitstrings. Note that we need that fact that we replaced $\mathcal{I}$ by $\overline{\mathcal{I}}$ here: The oblivious implementation guarantees that the bitstrings representing honest keys are not used for the generation of other terms (in particular this is interesting when honest keys appear at forgetful positions). Hence we can replace them with random bitstrings and still have an indistinguishable game. The rest of the indistinguishability argument is based on the IND-CCA security of the SKE scheme and analogous to Theorem 7.

Game 4

In Game 4, analogously to Game 3 from Theorem 7, we show that the adversary cannot win by producing encryptions under honest keys. To show the indistinguishability of Game 4 and Game 3 we use the same arguments for "reconstructions" and "forgeries" as in Theorem 6. Note that we simulate $\mathcal{I}_{\mathsf{SKE}}$ using transparent functions within this process. Here, we need the requirement that valid predicate of $\mathcal{I}$ does not depend on function symbols from Σ_{SKE}. Without this, we couldn't replace the function symbols from Σ_{SKE} with their transparent counterparts and still expect to have a valid trace when we are playing the deduction soundness game for $\mathcal{I}$ in the simulation.

Game 5

Finally, analogously to Game 4 from Theorem 7, the simulator $\mathcal{B}$ plays the variation of the deduction soundness game for $\overline{\mathcal{I}}$ which it cannot win with non-negligible probability by Lemma 62. □

Let $\overline{\mathcal{M}_{\mathsf{SKE}}}$ be the forgetful symbolic model based on $\mathcal{M}_{\mathsf{SKE}}$ when we mark the message m for honestly generated encryptions under honest keys $E_h^{\hat{l}}(k_h^l(), m$ as a forgetful position and pick $\mathcal{I}_{\mathsf{SKE}}[\Sigma]$ as an implementation of $\overline{\mathcal{M}_{\mathsf{SKE}}}$. Then the following holds:

Theorem 11. *Let $\mathcal{M}$ be a forgetful symbolic model and $\mathcal{I}$ be a forgetful deduction sound implementation of $\mathcal{M}$. $\mathcal{I}_{\mathsf{SKE}}$ denotes $\mathcal{I}_{\mathsf{SKE}}[\Sigma]$ where Σ is the signature from $\mathcal{M}$. If $(\overline{\mathcal{M}_{\mathsf{SKE}}}, \mathcal{I}_{\mathsf{SKE}})$ and $(\mathcal{M}, \mathcal{I})$ are compatible (see requirements in Section 3.5), then $\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}$ is a forgetful implementation of $\mathcal{M} \cup \overline{\mathcal{M}_{\mathsf{SKE}}}$*

Proof. We pick the obvious oblivious implementation $\overline{\mathcal{I}_{\mathsf{SKE}}}$ for $\mathcal{I}_{\mathsf{SKE}}$ and set $(\overline{M_{\mathsf{SKE}}}\ E_h)(k, ell; r) := (M_{\mathsf{SKE}}\ E_h)(k, 0^\ell; r)$ and proof the theorem with a sequence of games:

Game 0

Game 0 is the game

$$\mathsf{FIN}^1_{(\mathcal{M} \cup \overline{\mathcal{M}_{\mathsf{SKE}}}) \cup \mathcal{M}_{\mathsf{tran}}(\nu), (\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu), (\overline{\mathcal{I}} \cup \overline{\mathcal{I}_{\mathsf{SKE}}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu), \mathcal{A}}(\eta)$$

Game 1

In Game 1 we replace the implementation $\mathcal{I}$ with a corresponding oblivious implementation $\overline{\mathcal{I}}$ (which exists since $\mathcal{I}$ is a forgetful implementation according to Definition 54). We can do this analogously to Game 1 from Theorem 10 and the indistinguishability of Game 0 and Game 1 holds for the same reasons. Game 1 is

$$\mathsf{FIN}^0_{(\mathcal{M} \cup \overline{\mathcal{M}_{\mathsf{SKE}}}) \cup \mathcal{M}_{\mathsf{tran}}(\nu), (\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu), (\overline{\mathcal{I}} \cup \mathcal{I}_{\mathsf{SKE}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu), \mathcal{A}}(\eta)$$

Game 2

In Game 2 we replace $\mathcal{I}_{\mathsf{SKE}}$ by $\overline{\mathcal{I}_{\mathsf{SKE}}}$ and the honest keys in the library by random values. We have indistinguishability of Game 1 and Game 2 by the IND-CCA security of the SKE scheme. Game 2 is indistinguishable[6] from

$$\mathsf{FIN}^0_{(\mathcal{M}\cup\overline{\mathcal{M}_{\mathsf{SKE}}})\cup\mathcal{M}_{\mathsf{tran}}(\nu),(\mathcal{I}\cup\mathcal{I}_{\mathsf{SKE}})\cup\mathcal{I}_{\mathsf{tran}}(\nu),(\overline{\mathcal{I}}\cup\overline{\mathcal{I}_{\mathsf{SKE}}})\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta)$$

In conclusion, Game 0 and Game 2 are indistinguishable. Hence $\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}$ is a forgetful implementation of $\mathcal{M} \cup \overline{\mathcal{M}_{\mathsf{SKE}}}$. □

[6] Note that we only have indistinguishability here due to the random values for honest keys in the library.

4. Outlook

Finally, we would like to give a short outlook on interesting directions for further research.

Analyze Existing Protocols.

With the formal foundations and symbolic UC framework in place, the modular analysis of an existing protocol would be one interesting next step. Certainly TLS would be an appealing target. However, due to its known flaws and the design, we could expect a result for a modified version of the protocol at best. Fortunately, there are modern protocols that were designed with UC security in mind. One interesting candidate is the Direct Anonymous Attestation (DAA) protocol [35] that has been adopted by the Trusted Computing Group in the latest version of its Trusted Platform Module specification. The protocol enables the remote authentication of a trusted platform whilst preserving the user's privacy [76].

Improve Tools for Automated Proofs.

Although we partially used Proverif for some of the proof steps in Section 2.7, the analysis of our example protocols still required a lot of manual work. This situation could be improved by augmenting the capabilities of tools like Proverif to show observational equivalence. First steps in this direction have been made by [45]. In the long run, our goal is to establish proofs for cryptographic protocols as a part of their standardization process. To this end, having powerful and user-friendly tools is crucial.

Extend Composable Computational Soundness to Protocols.

One interesting direction for future work in the field of computational soundness would be to extend our results for composable soundness from Chapter 3 to protocols. Partially, this has already been discussed in [48]. An interesting way to achieve composable computational soundness for protocols would be to transfer deduction soundness to a suitable framework like CoSP [7]. The idea behind CoSP is to decouple computational soundness results from the a concrete process calculus like the applied pi calculus.

Extend Symbolic Universal Composability.

Although the UC framework provides a good foundation for the modular analysis of cryptographic protocols, certain scenarios require extensions. For example, incoercibility is an important security property for voting protocols and cannot be captured by the standard UC framework in the computational model. [74] provides an extension that could hopefully be carried over to the symbolic setting to analyze incoercibility with the help of automated tools.

Bibliography

[1] Martin Abadi and Cedric Fournet. “Mobile values, new names, and secure communication”. In: *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages.* ACM New York, NY, USA. 2001, pp. 104–115.

[2] Martín Abadi and Phillip Rogaway. “Reconciling two views of cryptography (The computational soundness of formal encryption)”. In: *Proc. 1st IFIP International Conference on Theoretical Computer Science (IFIP–TCS'00).* Vol. 1872. LNCS. 2000, pp. 3–22.

[3] Benny Applebaum. “Garbling XOR Gates "For Free" in the Standard Model”. In: *TCC.* 2013, pp. 162–181.

[4] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. “Analysing Unlinkability and Anonymity Using the Applied Pi Calculus”. In: *CSF.* 2010, pp. 107–121.

[5] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. “Analysing Unlinkability and Anonymity Using the Applied Pi Calculus”. In: *CSF 2010, 23rd IEEE Computer Security Foundations Symposium.* IEEE, 2010, pp. 107–121.

[6] Gildas Avoine. “Cryptography in radio frequency identification and fair exchange protocols”. PhD thesis. EPFL, Lausanne, Switzerland, 2005.

[7] Michael Backes, Dennis Hofheinz, and Dominique Unruh. “CoSP: a general framework for computational soundness proofs”. In: *ACM Conference on Computer and Communications Security.* ACM, 2009, pp. 66–78.

[8] Michael Backes and Birgit Pfitzmann. “Symmetric Encryption in a simulatable Dolev-Yao style cryptographic library”. In: *Proc. 17th IEEE Computer Science Foundations Workshop (CSFW'04).* 2004, pp. 204–218.

[9] Michael Backes, Birgit Pfitzmann, and Michael Waidner. “A Composable Cryptographic Library with Nested Operations”. In: *Proc. 10th ACM CCS.* 2003, pp. 220–230.

[10] Michael Backes, Birgit Pfitzmann, and Michael Waidner. “Symmetric authentication within simulatable cryptographic library”. In: *Proc. 8th European Symposium on Research in Computer Security (ESORICS'03).* Lecture Notes in Computer Science. 2003, pp. 271–290.

[11] Michael Backes, Birgit Pfitzmann, and Michael Waidner. “The Reactive Simulatability (RSIM) Framework for Asynchronous Systems”. In: *Information and Computation* 205.12 (2007), pp. 1685–1720.

[12] Boaz Barak and Amit Sahai. “How To Play Almost Any Mental Game Over The Net — Concurrent Composition via Super-Polynomial Simulation”. In: *Proc. 46th IEEE Symposium on Foundations of Computer Science (FOCS).* 2005, pp. 543–552.

[13] Christoph Baumann, Bernhard Beckert, Holger Blasum, and Thorsten Bormer. “Lessons Learned From Microkernel Verification – Specification is the New Bottleneck”. In: *SSV.* 2012, pp. 18–32.

[14] Bernhard Beckert, Sarah Grebing, and Florian Böhl. “A Usability Evaluation of Interactive Theorem Provers Using Focus Groups”. In: *Proceedings, Workshop on Human-Oriented Formal Methods (HOFM), Grenoble, September 2014.* LNCS. to appear. Springer, 2014.

[15] Bernhard Beckert, Sarah Grebing, and Florian Böhl. “How to Put Usability into Focus: Using Focus Groups to Evaluate the Usability of Interactive Theorem Provers”. In: *Proceedings, Workshop on User Interfaces for Theorem Provers (UITP), Vienna, July 2014.* Ed. by Christoph Benzmüller and Bruno Woltzenlogel Paleo. EPTCS. to appear. 2014.

[16] Mihir Bellare and Phillip Rogaway. “Random Oracles are Practical: A Paradigm for Designing Efficient Protocols”. In: *ACM Conference on Computer and Communications Security.* Ed. by Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby. ACM, 1993, pp. 62–73. ISBN: 0-89791-629-8.

[17] Bruno Blanchet. Personal communication. Aug. 2012.

[18] Bruno Blanchet. “A Computationally Sound Mechanized Prover for Security Protocols”. In: *IEEE Symposium on Security and Privacy.* Oakland, California, 2006, pp. 140–154.

[19] Bruno Blanchet. *Automatic Proof of Strong Secrecy for Security Protocols.* Tech. rep. MPI-I-2004-NWG1-001. Saarbrücken, Germany: Max-Planck-Institut für Informatik, July 2004.

[20] Bruno Blanchet. “Automatic verification of correspondences for security protocols”. In: *Journal of Computer Security* 17.4 (2009). Preprint available as arXiv:0802.3444v1 [cs.CR], pp. 363–434.

[21] Bruno Blanchet. *ProVerif 1.86pl4: Automatic Cryptographic Protocol Verifier - User Manual and Tutorial.* `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf`. 2012.

[22] Bruno Blanchet, Martín Abadi, and Cédric Fournet. “Automated Verification of Selected Equivalences for Security Protocols”. In: *Journal of Logic and Algebraic Programming* 75 (2008). Online available at `http://www.di.ens.fr/~blanchet/publications/BlanchetAbadiFournetJLAP07.pdf`, pp. 3–51.

[23] Florian Böhl. “On Symbolic Simulatability”. Diploma thesis. Karlsruhe Institute of Technology, Germany, 2010.

[24] Florian Böhl, Véronique Cortier, and Bogdan Warinschi. “Deduction soundness: prove one, get five for free”. In: *ACM Conference on Computer and Communications Security.* ACM, 2013, pp. 1261–1272.

[25] Florian Böhl, Véronique Cortier, and Bogdan Warinschi. “Deduction Soundness: Prove One, Get Five for Free”. In: *IACR Cryptology ePrint Archive* 2013 (2013), p. 457.

[26] Florian Böhl, Gareth T. Davies, and Dennis Hofheinz. “Encryption Schemes Secure under Related-Key and Key-Dependent Message Attacks”. In: *Public Key Cryptography*. LNCS. Springer, 2014, pp. 483–500.

[27] Florian Böhl, Simon Greiner, and Patrik Scheidecker. “Proving Correctness and Security of Two-Party Computation Implemented in Java in Presence of a Semi-Honest Sender”. In: *CANS*. LNCS. to appear. Springer, 2014.

[28] Florian Böhl, Dennis Hofheinz, and Daniel Kraschewski. “On Definitions of Selective Opening Security”. In: *Public Key Cryptography*. LNCS. Springer, 2012, pp. 522–539.

[29] Florian Böhl and Dominique Unruh. Proverif examples from the present thesis: `http://boehl.name/publications/symbolic-uc/proverif-files.zip`. 2013.

[30] Florian Böhl and Dominique Unruh. “Symbolic Universal Composability”. In: *CSF*. IEEE, 2013, pp. 257–271.

[31] Florian Böhl and Dominique Unruh. “Symbolic Universal Composability”. In: *IACR Cryptology ePrint Archive* 2013 (2013), p. 62.

[32] Florian Böhl, Dennis Hofheinz, Tibor Jager, Jessica Koch, and Christoph Striecks. “Confined Guessing: New Signatures From Standard Assumptions”. In: *IACR Cryptology ePrint Archive* 2013 (2013), p. 171.

[33] Florian Böhl, Dennis Hofheinz, Tibor Jager, Jessica Koch, and Christoph Striecks. “Confined Guessing: New Signatures From Standard Assumptions”. In: *J. Cryptology* to appear (2015).

[34] Florian Böhl, Dennis Hofheinz, Tibor Jager, Jessica Koch, Jae Hong Seo, and Christoph Striecks. “Practical Signatures from Standard Assumptions”. In: *EUROCRYPT*. LNCS. Springer, 2013, pp. 461–485.

[35] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. “Direct anonymous attestation”. In: *Proc. 11th ACM Conference on Computer and Communications Security*. ACM Press, 2004, pp. 132–145.

[36] Ran Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols”. In: *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*. Extended version in Cryptology ePrint Archive, Report 2000/67, `http://eprint.iacr.org/`. 2001, pp. 136–145.

[37] Ran Canetti and Marc Fischlin. “Universally Composable Commitments”. In: *Advances in Cryptology, Proceedings of CRYPTO 2001*. Ed. by Joe Kilian. Lecture Notes in Computer Science 2139. Full version online available at `http://eprint.iacr.org/2001/055.ps`. Springer-Verlag, 2001, pp. 19–40.

[38] Ran Canetti and Jonathan Herzog. “Universally Composable Symbolic Security Analysis”. In: *J Cryptology* 24.1 (Jan. 2011), pp. 83–147.

[39] Ran Canetti and Tal Rabin. “Universal Composition with Joint State”. In: *Proc. CRYPTO 2003*. Vol. 2729. LNCS. Springer, 2003, pp. 265–281.

[40] Ran Canetti and Margarita Vald. “Universally Composable Security with Local Adversaries”. In: *SCN 2012*. Ed. by Ivan Visconti and Roberto De Prisco. Vol. 7485. Lecture Notes in Computer Science. Springer, 2012, pp. 281–301. ISBN: 978-3-642-32927-2.

[41] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. *Task-Structured Probabilistic I/O Automata.* Tech. rep. MIT-CSAIL-TR-2006-060. Online available at `http://dspace.mit.edu/handle/1721.1/33964`. MIT CSAIL, Sept. 2006.

[42] Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Moses Liskov, Nancy A. Lynch, Olivier Pereira, and Roberto Segala. “Time-Bounded Task-PIOAs: A Framework for Analyzing Security Protocols”. In: *DISC.* 2006, pp. 238–253.

[43] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. “Universally Composable Security with Global Setup”. In: *Proc. 4th Theory of Cryptography Conference (TCC).* 2007, pp. 61–85.

[44] Cher. *Bang Bang (My Baby Shot Me Down).* 7" single. Written by Sonny Bono, sound sample: `http://en.wikipedia.org/wiki/File:Cher-_Bang_Bang_My_Baby_Shot_Me_Down.ogg`. 1966.

[45] Vincent Cheval and Bruno Blanchet. “Proving More Observational Equivalences with ProVerif”. In: *POST 2013*. Ed. by David Basin and John Mitchell. Vol. 7796. LNCS. Springer, 2013, pp. 226–246.

[46] Hubert Comon-Lundh and Véronique Cortier. “Computational soundness of observational equivalence”. In: *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS’08).* Alexandria, Virginia, USA: ACM Press, Oct. 2008.

[47] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. “A Survey of Symbolic Methods in Computational Analysis of Cryptographic Systems”. In: *J. Autom. Reasoning* 46.3-4 (2011), pp. 225–259.

[48] Véronique Cortier and Bogdan Warinschi. “A Composable Computational Soundness Notion”. In: *18th ACM Conference on Computer and Communications Security (CCS’11).* Chicago, USA: ACM, 2011, pp. 63–74.

[49] Véronique Cortier and Bogdan Warinschi. “Computationally Sound, Automated Proofs for Security Protocols”. In: *European Symposium on Programming (ESOP’05).* Vol. 3444. LNCS. Edinburgh, UK: Springer, 2005, pp. 157–171.

[50] Véronique Cortier, Steve Kremer, Ralf Küsters, and Bogdan Warinschi. “Computationally Sound Symbolic Secrecy in the Presence of Hash Functions”. In: *Proceedings of the 26th Conference on Fundations of Software Technology and Theoretical Computer Science (FSTTCS’06).* Vol. 4337. LNCS. Kolkata, India: Springer, 2006, pp. 176–187.

[51] Anupam Datta, Ralf Küsters, John C. Mitchell, and Ajith Ramanathan. “On the Relationships Between Notions of Simulation-Based Security”. In: *Theory of Cryptography, Proceedings of TCC 2005.* Ed. by Joe Kilian. Lecture Notes in Computer Science. Springer-Verlag, 2005, pp. 476–494.

[52] Anupam Datta, Ante Derek, John C. Mitchell, Vitaly Shmatikov, and Mathieu Turuani. "Probabilistic Polynomial-time Semantics for a Protocol Security Logic". In: *Proc. of 32nd International Colloquium on Automata, Languages and Programming, ICALP*. Vol. 3580. LNCS. Lisboa, Portugal. Springer, 2005, pp. 16–29.

[53] Stephanie Delaune, Steve Kremer, and Olivier Pereira. *Simulation based security in the applied pi calculus*. IACR ePrint 2009/267, version 5 June 2009. Full version of [54].

[54] Stephanie Delaune, Steve Kremer, and Olivier Pereira. "Simulation based security in the applied pi calculus". In: *FSTTCS*. Ed. by Ravi Kannan and K. Narayan Kumar. Vol. 4. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2009, pp. 169–180.

[55] Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. "Public-Key Encryption Schemes with Auxiliary Inputs". In: *TCC*. Ed. by Daniele Micciancio. Vol. 5978. Lecture Notes in Computer Science. Springer, 2010, pp. 361–381. ISBN: 978-3-642-11798-5.

[56] Danny Dolev and Andrew Chi-Chih Yao. "On the Security of Public Key Protocols (Extended Abstract)". In: *FOCS*. IEEE, 1981, pp. 350–357.

[57] Dana Ford. *Cheney's defibrillator was modified to prevent hacking*. 2013. URL: `http://edition.cnn.com/2013/10/20/us/dick-cheney-gupta-interview/` (visited on 03/15/2014).

[58] Flavio D. Garcia and Peter van Rossum. "Sound and Complete Computational Interpretation of Symbolic Hashes in the Standard Model". In: *Theoretical Computer Science* 394 (2008), pp. 112–133.

[59] Shafi Goldwasser and Silvio Micali. "Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information". In: *STOC*. ACM, 1982, pp. 365–377.

[60] Matthew Green. *On the (provable) security of TLS: Part 1*. 2012. URL: `http://blog.cryptographyengineering.com/2012/09/on-provable-security-of-tls-part-1.html` (visited on 03/15/2014).

[61] Gordon Hall. *Mt. Gox Blames Bitcoin – Core Developer Greg Maxwell Responds*. 2014. URL: `http://www.cryptocoinsnews.com/2014/02/10/mt-gox-blames-bitcoin-core-developer-greg-maxwell-responds/` (visited on 03/15/2014).

[62] Dennis Hofheinz and Victor Shoup. *GNUC: A New Universal Composability Framework*. IACR ePrint 2011/303. 2011.

[63] Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. "Completing the Picture: Soundness of Formal Encryption in the Presence of Active Adversaries". In: *European Symposium on Programming (ESOP'05)*. Vol. 3444. LNCS. 2005, pp. 172–185.

[64] Ralf Küsters. "Simulation-Based Security with Inexhaustible Interactive Turing Machines". In: *CSFW 2006, Computer Security Foundations Workshop*. Long version available as IACR eprint 2006/151. IEEE Computer Society, 2006, pp. 309–320.

[65] Ralf Küsters, Anupam Datta, John C. Mitchell, and Ajith Ramanathan. "On the Relationships between Notions of Simulation-Based Security". In: *J. Cryptology* 21.4 (2008), pp. 492–546.

[66] Gavin Lowe. "An attack on the Needham-Schroeder public-key authentication protocol". In: *Information Processing Letters* 56 (3 1995), pp. 131–133. ISSN: 0020-0190. DOI: `10.1016/0020-0190(95)00144-2`.

[67] Jörn Müller-Quade and Dominique Unruh. "Long-term Security and Universal Composability". In: *Theory of Cryptography, Proceedings of TCC 2007.* Vol. 4392. Lecture Notes in Computer Science. Preprint on IACR ePrint 2006/422, superseeded by [67]. Springer-Verlag, 2007, pp. 41–60.

[68] Moni Naor and Gil Segev. "Public-Key Cryptosystems Resilient to Key Leakage". In: *SIAM J. Comput.* 41.4 (2012), pp. 772–814.

[69] Manoj Prabhakaran and Amit Sahai. "New Notions of Security: Achieving Universal Composability without Trusted Setup". In: *Proc. 36th Annual ACM Symposium on Theory of Computing (STOC).* 2004, pp. 242–251.

[70] The EasyCrypt Project. *EasyCrypt: Computer-Aided Cryptographic Proofs.* 2014. URL: `https://www.easycrypt.info` (visited on 03/15/2014).

[71] Jae Hong Seo. "Short Signatures From Diffie-Hellman: Realizing Short Public Key". In: *IACR Cryptology ePrint Archive* 2012 (2012), p. 480.

[72] Dominique Unruh. "Concurrent composition in the bounded quantum storage model". In: *Eurocrypt 2011.* Vol. 6632. LNCS. Preprint on IACR ePrint 2010/229. Springer, 2011, pp. 467–486.

[73] Dominique Unruh. "Universally Composable Quantum Multi-Party Computation". In: *Eurocrypt 2010.* LNCS. Preprint on arXiv:0910.2912 [quant-ph]. Springer, 2010, pp. 486–505.

[74] Dominique Unruh and Jörn Müller-Quade. "Universally Composable Incoercibility". In: *Crypto 2010.* Vol. 6223. LNCS. Preprint on IACR ePrint 2009/520. Springer, 2010, pp. 411–428.

[75] Wikipedia. *Digital Enhanced Cordless Telecommunications.* 2014. URL: `https://en.wikipedia.org/wiki/DECT` (visited on 03/15/2014).

[76] Wikipedia. *Direct Anonymous Attestation.* 2014. URL: `https://en.wikipedia.org/wiki/Direct_Anonymous_Attestation` (visited on 03/17/2014).

[77] Wikipedia. *Global System for Mobile Communications.* 2014. URL: `https://en.wikipedia.org/wiki/GSM` (visited on 03/15/2014).

[78] Wikipedia. *Transport Layer Security.* 2014. URL: `https://en.wikipedia.org/wiki/Transport_Layer_Security` (visited on 03/15/2014).

Symbol Index

$\mathcal{C}_{x,a}^{SID_{bits}}$	A concrete fixed SID_{bits}-indexing context	42
$\mathcal{G}_{x,a}^{n}$	Auxiliary definition in analysis of $\mathcal{C}_{x,a}^{SID_{bits}}$	42
$\mathcal{C}_{x,a}^{(sID,gID,n)}$	Auxiliary definition in analysis of $\mathcal{C}_{x,a}^{SID_{bits}}$	42
sID	Auxiliary definition in analysis of $\mathcal{C}_{x,a}^{SID_{bits}}$ – set of spawned IDs	42
nil	Constructor denoting the empty bitstring	42
$zero$	Constructor prefixing a bitstring with 0	42
one	Constructor prefixing a bitstring with 1	42
SID_{bits}	Concrete set of session IDs built from bitstrings	42
gID	Auxiliary definition in analysis of $\mathcal{C}_{x,a}^{SID_{bits}}$ – set of generator IDs	42
$\Sigma_{x\in S}P$	Short for $P\{s_1/x\}\|P\{s_2/x\}\|\dots$ for $S = \{s_1, s_2, \dots\}$	42
$\equiv_{\mathrm{E}}$	Structural equivalence modulo equational theory E	16
$\rlap{\nearrow}\equiv$	Asymmetric variant of structural equivalence	26
$crseqv$	Constructor: CRS for equivocation	82
$crsext$	Constructor: CRS for extraction	82
$\mathcal{F}_{COM}$	Commitment functionality	81
KE^*	Protocol for generating many keys	76
$\mathcal{M}_{real}$	Symbolic model without virtual primitives	82
$\mathcal{M}_{virt}$	Symbolic model with virtual primitives	82
$sdec$	Destructor: symmetric decryption	67
$pdec$	Destructor: public key decryption	67
$pkofenc$	Destructor extracting public key from ciphertext	67
$pkofsk$	Destructor extracting secret from public key	67
$bv(P)$	Bound variables in P	13
$P \equiv Q$	Structural equivalence of P and Q	14
$M(x)$	Receiving x on channel N	13
$\overline{M}\langle N\rangle$	Sending N on channel N	13

$!P$	Concurrent executions of instances of P (applied pi calculus	13
νa	Restriction of the name a (applied pi calculus)	13
$\mathit{fv}(P)$	Free variables in P	13
$\mathit{bn}(P)$	Bound names in P	13
let $x = D$ in P else Q	Let it be	13
$\mathit{fn}(P)$	Free names in P	13
snd	Destructor: Extracts the second component of a tag	16
$\{a/b\}$	Substitution replacing b with a	19
$\mathcal{D}$	The deduction system	104
$\leq$	Subtype relation	103
labels	The set of labels	103
$\mathcal{T}$	The set of data types	103
$\top$	The data type *base type*	103
$\mathrm{dom}(f)$	The domain of a function f	104
$\vdash$	The deduction relation	104
$\mathsf{labelsA}$	The set of adversarial labels	103
$\mathsf{labelsH}$	The set of honest labels	103
$\sim_{\mathcal{S}_{sid}}$	An $\mathcal{S}_{sid}$-observational equivalence relation	54
untag	Untag channel identifiers	52
n_{sid}	Fixed name for sid-sensitive processes	48
$\approx^{n}_{\mathcal{S}}$	Observational equivalence restricted to processes that do not contain n and contexts build from $\mathcal{S}$	47
$!!P$	Concurrent composition of P with session ids	45
$\langle\cdot\rangle$	Span of a set of IDs	42
tag	Tag channel identifiers	50
Φ	Transformation of a generic plain process into a sid-sensitive process	48
$\mathcal{S}_{sid}$	The set of sid-sensitive processes	48
M_{sid}	Fixed term for sid-sensitive processes	48
fakeH	Constructor: Fake (equivocal) hash	82
fake	Constructor: Randomness for fake hash	82
$\mathit{extract}$	Destructor: Extracting from a hash	82
$\mathcal{F}_{CRS}$	Common reference string functionality	82
COM	Commitment protocol	82
$\mathcal{F}_{KE}$	Key exchange functionality	67
$\mathcal{F}_{PKI}$	Public key infrastructure functionality	67
NSL	Needham-Schroeder-Lowe protocol	67
SC	Secure channel protocol	69
$\mathcal{N}$	The set of names	12
fst	Destructor: Extracts the first component of a tag	16
$\prod_{x\in S} P$	Indexed replication of the process P	19
$r \leftarrow R$	r is sampled uniformly from R	103
η	The security parameter	103
$P \downarrow_M$	The process P emits on a channel M	14
$P \to Q$	Process P reduces to Q	14

$P \updownarrow_M$	The process P communicates on a channel M	14
$P \uparrow_M$	The process P reads on a channel M	14
if $M = N$ then P else Q	Syntactic sugar for let x = $\mathit{equals}(M, N)$ in P else Q	15
$P \approx Q$	Observational equivalence of the closed processes P and Q	15
$\overline{C}\langle\rangle.P$	Syntactic sugar for $C(\mathit{empty}).P$	15
$C().P$	Syntactic sugar for $C(x).P$ with fresh variable x	15
$\mathit{plain}^s(P)$	P with synchronization channel s removed	30
$\mathit{event}\ f(t)$	Raise event $f(t)$	29
$\mathit{syncout}^s(t_1 \mapsto t'_1, \ldots; u_1 \mapsto u'_1, \ldots)$	Outputs on synchronization channel s	31
$\mathit{ev}^s(P)$	P with synchronization channel s replaced by events	30
NET	Set of all network names	37
IO	Set of all I/O names	37
$P(\!(M)\!)$	Process P with session-id M	41
$P \leq Q$	P emulates Q	38
$\mathcal{C}^{SID}_{x,\underline{n}}$	An arbitrary but fixed *SID*-indexing context	41
SID	Set of all session IDs	41
hash	Constructor: hash function	66
empty	Constructor: empty message	66
sk	Constructor: secret key	66
senc	Constructor: symmetric encryption	66
penc	Constructor: public key encryption	66
pk	Constructor: public key	66
$P \leq^{\mathsf{SS}} Q$	P emulates Q in the sense of Delaune et al. [54].	63
$\mathcal{F}_{anon}$	Insecure but anonymous channel functionality	64
$\mathcal{F}_{SC}$	Secure channel functionality	62
$\preceq$	Observational preorder	63
equals	Destructor equals	16
$P \approxeq Q$	Full observational equivalence of the non-closed processes P and Q	16
$\ncong_E$	$\ncong$ modulo equational theory	26
$DM \Downarrow$	Term D evaluates to M	12
R	Finite set of rewrite rules for destructors	12
$D(M_1, \ldots, M_n) \rightarrow M$	Reduction rule for destructor D	12
$M =_{\mathrm{E}} N$	Terms M and N are equal with respect to the equational theory E	12
E	The finite set of equations that are to hold in the equational theory (applied pi calculus)	12
$\mathcal{T}$	The set of terms	12
Σ	The Signature – a set of function symbols	12, 103
$\mathcal{V}$	The set of variables	12, 103
$\mathbf{0}$	Empty process (applied pi calculus)	13
$\mathcal{M}$	Symbolic model $\mathcal{M}$	12

Index

$\mathcal{S}$-n-bisimulation, 49
$\mathcal{S}$-n-observational equivalence, 49
$\mathcal{S}$-n-simulation, 49
0-1-context, 17

adversary, 39
  dummy, 39
α-conversion, 16

base type, 105
bisimulation, 17
bitstring
  interpretation, 108
black-box simulatability, 40

channel identifiers, 17
communicate, 16
compatible, 113
complete (set of processes), 49
composition
  concurrent, 42
concurrent composition, 42
constructor, 14
context, 16
  0-1-, 17
  evaluation, 16
  indexing, 43
  multi-hole, 37

data types, 105
deduction
  system, 106
  proof, 106
  relation, 106
deduction soundness, 121
destructor, 14
destructor term, 14
  $\mathcal{M}$-, 90
DKP-security, 65
dummy adversary, 39

emit, 16
empty, 18
emulate, 40
equals, 18
equivalence
  full observational, 18
  observational, 17
  structural, 16
event process, 31
EVENT rule, 31
extension
  safe, 90

full observational equivalence, 18
function
  negligible, 105
  transparent, 113
function symbol, 14, 105
  constant, 105
  constructor, 14
  destructor, 14
  deterministic, 105
  foreign, 112
  garbage, 106
  randomized, 105

generate function, 108

hash function, 152

if-statement, 17
implementation, 108
  collision free, 110
  composable, 114
  deduction sound, 121
  forgetful, 156
  good, 110
  oblivious, 156
  parametrized, 120
  transparent, 113
  type safe, 111
IND-CCA security, 124, 143
indexed replication, 21
indexing context, 43
INT-CTXT security, 144
internal reduction, 16
IREPL, 21

label, 105
  adversarial, 105
  honest, 105
length regularity, 110
library, 155

$\mathcal{M}$-destructor term, 90
$\mathcal{M}$-process, 90
$\mathcal{M}$-term, 90
MAC scheme, 149
model
  symbolic, 15, 105
multi-hole context, 37

name, 14
  bound, 15
name-reduced, 33
natural symbolic model, 18
negligible, 105
NET-stable, 39
nonce, 105

observational equivalence, 17
  full, 18
observational preorder, 65
open function, 109

parsing function, 109
preorder
  observational, 65
process
  $\mathcal{M}$-, 90
  closed, 15
  event, 31
  product, 21
product process, 21
protected, *see* unprotected
public-key encryption, 124

query, 112
  variation, 112

random oracle, 152
read, 16
relay, 67
replication
  indexed, 21

safe extension, 90
satisfy
  trace property, 31
secret key encryption, 143
security
  IND-CCA, 124, 143
  INT-CTXT, 144
  strong EUF-CMA, 134, 149
security parameter, 105
signature, 14, 105
signature scheme, 134
simulatability
  black-box, 40
  strong, 40
  universally-composable, 40
simulation, 17
simulator, 40
strong EUF-CMA security, 134, 149
strong simulatability, 40
strong unlinkability, 65
structural equivalence, 16
substitution, 14, 106
  closing, 18
subtype relation, 105
symbolic model, 15, 105
  composable, 114
  forgetful, 156
  natural, 18
  parametrized, 120
  transparent, 113

term
  $\mathcal{M}$-, 90

garbage, 106
hybrid, 108, 155
of type τ, 105
trace, 112
trace property, 31
satisfy, 31
transparent function, 113
type
base, 105
universally-composable simulatability, 40
unlinkability
strong, 65
unprotected, 16

variable, 14, 105
bound, 15
free, 15
virtual primitives, 81