Formal Verification of Composable Security Proofs

Seyed Reza Sefidgar

Formal Verification of Composable Security Proofs

Diss. ETH No. 27653

SEYED REZA SEFIDGAR

FORMAL VERIFICATION OF COMPOSABLE SECURITY PROOFS

DISS. ETH NO. 27653

FORMAL VERIFICATION OF COMPOSABLE SECURITY PROOFS

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

SEYED REZA SEFIDGAR
M.Sc., Technische Universität München

born on 1988
citizen of Iran

accepted on the recommendation of

Prof. Dr. David Basin, examiner
Prof. Dr. Ueli Maurer, co-examiner
Prof. Dr. Gilles Barthe, co-examiner
Dr. Andreas Lochbihler, co-examiner

2021

Seyed Reza Sefidgar: *Formal Verification of Composable Security Proofs,* © 2021

doi: 10.3929/ethz-b-000489454

For my parents

ABSTRACT

Computer-aided cryptography improves the rigor of security proofs by mechanizing their verification. Existing formal-methods tools focus primarily on the game-based paradigm, and the results on formalizing simulation-based proofs are limited. Simulation-based frameworks are popular since they support the composition of security proofs, but the level of details in these frameworks surpasses what the formal-methods community can reasonably handle with state-of-the-art techniques. Hence, existing formal results consider streamlined versions of simulation-based frameworks to cope with their complexity.

Recent advancements in cryptography frameworks enable the development of better tools for formalizing composable security proofs. Constructive Cryptography is a generic theory allowing for clean, composable security statements that empowers protocol designers to focus on a particular aspect of security proofs without being distracted by other details. It lays the foundation for formal-methods tools that support mechanized verification of composable security statements.

In this thesis, we formalize an instance of Constructive Cryptography. Our approach is suitable for mechanized verification and we use CryptHOL, a framework for developing mechanized cryptography proofs, to implement it in the Isabelle/HOL theorem prover. We extend CryptHOL with an abstract model of Random Systems and provide proof rules for their equality and composition. We then formalize security as a special kind of random system construction in which a complex system is built from simpler ones. We demonstrate the practicality of our approach by formalizing two different constructions of a secure channel.

Our formal approach enables us to delve into an aspect of composable security proofs that has not been considered in previous formal results: we highlight the importance of system communication modeling in composable security statements. We show that fixing the communication patterns, which is sometimes applied to simplify proofs and overcome their complexity, can affect the reusability of security statements. We propose an abstract approach to modeling systems communication in Constructive Cryptography that avoids this problem.

ZUSAMMENFASSUNG

Computer-gestützte Kryptografie verbessert die Stringenz von Sicherheitsbeweisen durch Mechanisierung der Verifikation. Bestehende Werkzeuge der formalen Methoden beschränken sich vornehmlich auf Sicherheitsspiele, und nur wenige Resultate beziehen sich auf simulationsbasierte Sicherheitsbeweise. Simulationsbasierte Frameworks werden häufig verwendet, da sie die Komposition von Sicherheitsbeweisen erlauben. Ihre Detailliertheit übersteigt jedoch, was formale Methoden mit derzeitigen Techniken leisten können. Bestehende formale Resultate beziehen sich auf reduzierte Versionen von simulationsbasierten Frameworks, um deren Komplexität zu bewältigen.

Jüngste Fortschritte im Bereich von Kryptografie-Frameworks erlauben die Entwicklung von besseren Tools zur Formalisierung komponierbarer Sicherheitsbeweise. Constructive Cryptography ist eine generische Theorie, die klare, komposierbare Sicherheitsaussagen erlaubt. Dies ermöglicht Protokollentwicklern sich auf die einzelnen Aspekte von Sicherheitsbeweisen zu konzentrieren, ohne durch andere Details abgelenkt zu werden. Constructive Cryptography bietet das Fundament für Tools der formalen Methoden, die eine mechanisierte Verifikation von komponierbaren Sicherheitsaussagen ermöglichen.

In dieser Dissertation formalisieren wir eine Instanz von Constructive Cryptography. Unser Ansatz eignet sich für die mechanisierte Verifikation. Wir nutzen CryptHOL, ein Framework für die Entwicklung mechanisierter kryptografischer Beweise, um unsere Resultate in dem Theorembeweiser Isabelle/HOL zu implementieren. Wir erweitern CryptHOL um ein abstraktes Modell von Random Systems und geben Beweisregeln für ihre Gleichheit und Komposition an. Wir formalisieren Sicherheit als eine spezielle Random-System-Konstruktion, in der ein komplexes System aus einfacheren Systemen aufgebaut ist. Wir zeigen die Praktikabilität unseres Ansatzen indem wir zwei verschiedene Typen von sicheren Kommunikationskanälen formalisieren.

Unser Ansatz erlaubt es uns einen Aspekt von komponierbaren Sicherheitsbeweisen vertieft zu erforschen, der in vorherigen formalen Resultaten nicht berücksichtigt wurde: die Wichtigkeit der Modellierung von System-Kommunikation in komponierbaren Sicherheitsaussagen. Wie wir zeigen kann das Fixieren von Kommunikationsmustern, wie es manchmal zur Vereinfachung von Beweisen und zur Komplexitätsreduktion angewandt wird, die Wiederverwendbarkeit von Sicherheitsaussagen beeinträchtigen. Wir schlagen eine abstrakte Methode zur Modellierung von Kommunikationsmustern in Constructive Cryptography vor, welche dieses Problem umgeht.

ACKNOWLEDGEMENTS

I would like to express my gratitude to David Basin for providing the opportunity to shape my research career, which has been influenced by his rigorous approach to security research and profound sense of aesthetics. I would like to thank Andreas Lochbihler for his warm guidance, continuous support, and patience that helped me carry out my research. I would also like to thank Ueli Maurer for many interesting discussions and his insightful explanations; many of the ideas in this thesis stem from his passion for elegant mathematics and cryptography. I am also grateful to Gilles Barthe for his careful and thorough perusal of this dissertation.

My sincere thanks go to my loving and supportive parents, to whom I owe everything that I have achieved in my life. I would also like to thank my sister and my brother in law for their encouraging support.

I would like to specially thank Tobias Klenze who translated the thesis abstract in a short notice. I am very lucky to have had such an office mate, who is now a dear friend of mine, since the beginning of my doctoral studies. I learned many things during our discussions; and I am grateful for all the remarkable memories we had.

My thanks go to Karel Kubiceck for all the joyful conversations and memories we had together. He and Tobias made "F 58" (our office number) the most comforting, vibrant, and motivating place for research. I would also like to thank all the members of the Information Security group (Infsec) and the association of scientific staff (VMI) for making my experience at ETH Zurich such a memorable journey.

I also thank Mohammad, Arsam, and Hassan who enriched my moments with friendship, thoughtful conversations, and delicious food!

CONTENTS

LIST OF FIGURES

NOTATION INDEX

$e : \mathring{t}$	cf. A.1	expression e has type $\mathring{t}$
$\mathring{a}, \mathring{b}, \mathring{c}, \cdots$	cf. A.1	type variables
$\mathbb{A}, \mathbb{B}, \mathbb{C}, \cdots$	cf. A.1	type constructors
$\Rightarrow, \times, +, \cdots$	cf. A.1, A.2.1	infix type constructors
$\mathcal{I}, \oplus, \vdash, \vdash\dashv$	cf. 3.1.2, 3.2	interface typing
R	cf. 3.1	resource
C	cf. 3.2	converter
D	cf. 5.1	distinguisher
F	cf. 6.2.1	fused resource
tr	cf. 3.1	short name for prob. tran. fun. (oracle)
cr	cf. 6.2.3	short name for FRTs'core
rs	cf. 6.2.3	short name for FRTs' rest
$\{\mid name \mid\}$	cf. 6.2.1	FRT specified by *name* core
$\boxed{c}$	cf. 6.2.1	place holder for cores' type
$\boxed{r}$	cf. 6.2.1	place holder for rests' type
$+_{\mathrm{O}}$	cf. 3.1.1	CryptHOL's plus-oracle operator
$\parallel$	cf. 3.1.3	resource's parallel composition
$\odot, \mid$	cf. 3.2	converter's seq. and par. composition
$\rhd$	cf. 3.3	attaching a converter to resource(s)
$\blacktriangleright, \circledblacktriangleright$	cf. 5.1	distinguisher's connect and absorb
$\rhd_c, \parallel_c$	cf. 6.2.3	core's attach and parallel composition
$\rhd_r, \parallel_r$	cf. 6.2.3	rest's attach and parallel composition
NAME	cf. 3.1, 3.2, 6.2.1	definitions like *RES, CNV, FUSE*
`name`	cf. 6.2.1, 6.2.3	artificial entities like interface names
comm	cf. 3.1, 3.2	Isabelle/HOL commands

1 INTRODUCTION

1.1 PROBLEM CONTEXT

Since the emergence of *provable security*, cryptographers have proposed various frameworks to tackle the complexity of security proofs. For example, game-based frameworks [9, 49] support the verification of cryptographic schemes and simulation-based frameworks [1, 16, 27, 35] enable the modular reasoning about cryptographic protocols. These frameworks have significantly improved the comprehensibility of security arguments; however, the resulting proofs are still informal, often sketchy, and sometimes even technically incomplete or wrong.

In response to the *crisis of rigor* in cryptography [9, 25], the formal-methods community has developed tools that enable cryptographers to use computers to mechanically check security proofs. Prominent examples are CryptoVerif [11], CertiCrypt [3], EasyCrypt [5], Verypto [10], FCF [46], and CryptHOL [8]. All these tools focus primarily on the game-based paradigm, and the results on formalizing simulation-based proofs are limited to those that study individual protocols, e.g., [14, 24], or those that consider a simplified version of simulation-based frameworks [17].

The lack of formal-methods tools for simulation-based frameworks is due, by and large, to their complexity. These frameworks are popular since they pave the way for modular reasoning about cryptographic constructs independently of their application context. However, the level of details in these frameworks surpasses what the formal-methods community can reasonably handle with existing techniques. Formalizing security in a framework such as Universal Composability (UC) would require formal notions of algorithms, runtime, and complexity, which would be both challenging and detailed due to the way these foundations are modeled.

Simulation-based frameworks' approach to modeling systems communication is one of the main reasons for the complexity of their resulting proofs. In these frameworks, components communicate via network tapes and a central scheduler called the adversary. This concrete approach increases the complexity of ideal specifications and makes the security arguments intricate [15]. As such, the existing formal results such as EasyUC [17] consider a simplified version of the simulation-based frameworks with restricted communication capabilities between components. However, as we will show in Section 6.1, such simplifications affect the reusability of security statements.

Constructive Cryptography (CC) [40–42] proposes a fundamental shift in how security statements are made and proved. It introduces an abstract approach to composable security arguments in which systems are abstractly modeled as *Resources* and cryptographic schemes *Construct* a new resource from assumed resources. CC allows one to focus on a particular aspect of security proofs without being distracted by other details. This makes composable security statements manageable for protocol designers.

1.2 OUR SOLUTION

We extend CryptHOL [8, 36], a framework for formalizing game-based proofs in Isabelle/HOL [45], to support the formalization of security proofs in CC. This enables us to introduce an abstract approach to modeling systems communication in composable security statements by means of generic specifications. We follow CC's approach to *ideal-world–real-world* paradigm [22] and define a protocol's security by comparing its execution to an idealized specification that intrinsically satisfies the desired properties. We formalize ideal (and real) functionalities as *sets* of resources that exhibit similar behavior. This is in contrast to UC-style frameworks that define such functionalities as single objects and enables us to model systems communication without introducing a centralized adversary machine.

First, we introduce new coalgebraic datatypes that model resources as abstract probabilistic input-output (I/O) systems. Our formalization constitutes an instance of Maurer's theory of Random Systems [43] where the coalgebraic approach makes it amenable for mechanized proof checking. Second, we formalize an algebra of resources built using different composition operators. Third, we define various equivalence relations on resources that enable us to capture the security of individual constructions as indistinguishability of resources. Fourth, we lift the aforementioned building blocks to work on parametrized resources. We propose a semantic domain called *Fused Resource Templates* (FRT) that abstracts over the communication patterns among similar resources. We show how FRTs constitute an instantiation of CC. We generalize the notion of secure construction to FRTs and provide composition theorems for such constructions.

Finally, to demonstrate the applicability of our approach, we formalize two different ways of constructing a secure channel from an authenticated channel and a key using one-time-pad. Each case study considers further refinement of the authenticated channel or the key using our composition theorems. First, we formalize an information-theoretic security argument in which the authenticated channel is constructed using a random function and an insecure channel. Second, we formalize a computational security argument in which the key is constructed using Diffie-Hellman key exchange [20] and two authenticated channels.

The complete formalization, including the case studies, is available online [37, 38]. Appendix B provides entry points to the source files. Besides the fact that CryptHOL uses Isabelle/HOL, our formalization builds on Isabelle/HOL since it supports coalgebraic datatypes that recurse through non-free functors like discrete probability distributions. In principle, we could have carried out our formalization in other proof assistants like Coq or Lean, where constructing the codatatypes probably would require more effort, but dependent types would simplify the formalization of interfaces.

1.3 CONTRIBUTIONS

By combining ideas from cryptography, formal methods, and programming languages research, we develop a framework that facilitates the mechanized checking of composable cryptographic security proofs. In doing so, we make contributions to both the cryptology and the formal-methods community:

- We formalize an instantiation of CC in a theorem prover and thereby enable the mechanized verification of composable security arguments. We formalize the notion of multi-interfaced resources and provide proof rules that enable modular reasoning about concrete (and asymptotic) security statements.

- We propose the first CC instantiation that explicates the details of systems communication. The role of systems communication models has not been studied in the CC literature, although we show that reusability and modularity of CC proofs depend on it (cf. Section 6.1). We propose an abstract approach to capturing systems communication patterns in CC proofs.

- We use the aforementioned foundation to formalize two case studies that consider the one-time-pad construction of a secure channel from an authenticated channel and a key. In the first one, we consider the authenticated channel's realization that uses a shared random function (idealizing a pseudo-random function and a shared key) as a message authentication code; while in the second one, we consider Diffie-Hellman construction of the key. The aforementioned formalization are comparatively simpler and shorter than similar formal results.

This thesis summarizes the papers that I published in collaboration with my co-authors. In the above list of contributions, the case studies demonstrate the applicability of our formalization library; however, we carried out our research differently. The case studies were means for exploring the problem space and enabled us to figure out the formalization obstacles and introduce new abstractions. My advisor, Dr. Andreas Lochbihler, helped me find the right abstractions in the case studies by formalizing the co-algebraic notions of resources and converters, the notion of trace-equivalence on resources, and the composition theorems for indistinguishability. We came up with the solution for abstract modeling of system communication together: I have formalized the algebraic properties of FRTs, and Dr. Lochbihler formalized the composition theorems for FRT's security. The proof methodology and the high-level ideas were contributed by Prof. Ueli Maurer, and the research was supervised by Prof. David Basin.

1.4 STRUCTURE

We start by reviewing the essentials of composable security arguments, CC, and CryptHOL in Chapter 2 and introduce a running example that is the basis of our case studies. In Chapter 3, we describe our formalization of resources and their composition operators. In Chapter 4, we reflect on two notions of equality for resources and derive their corresponding proof rules. In Chapter 5, we define the security of constructions and present our first case study. In Chapter 6, we present FRTs and show their applicability in composable security proofs. We explain the importance of a precise systems communication modeling for the modularity and reusability of security proofs. We introduce FRTs as a semantic domain that enables such modeling. We then present our second case study to demonstrate FRTs' applicability. Finally, in Chapter 7, we compare with related work and draw conclusions.

This thesis is based on two of my publications [7, 39]. To keep the text simple, we only provide a high-level overview of the formalization and use diagrams to present the main ideas. Visualizations in this thesis are a direct translation of the formal text, i.e. they have corresponding syntax and semantics, and are thus meaningful. The readers can refer to the formalization source [37, 38] for more details. Appendix A, which is taken from [8], provides more details on the rigor of our results.

2 PRELIMINARIES

This chapter summarizes the topics needed for understanding the rest of the thesis. First, we explain the composability of security proofs and how it is treated in simulation-based frameworks. Our main focus will be on the communication modeling of these frameworks. Second, we review the two bases of the thesis, namely, CryptHOL and Constructive Cryptography. Finally, we illustrate a sample security argument in Constructive Cryptography that serves as a running example through the rest of the thesis.

In each section, we use the notation that is common among the research community for the topic under discussion. The thesis notation will be explained later upon the formal introduction of each concept.

2.1 COMPOSABLE SECURITY ARGUMENTS

The ideal-world–real-world [22] approach to security enables modular reasoning about security statements. In this approach, a protocol's security is defined by comparing its execution, i.e., the real world, to an idealized specification that satisfies the desired properties by definition: a protocol π *realizes* (or implements) an idealized functionality $\mathcal{F}$ if there exists a *simulator* that can simulate π's behavior (in an adversarial environment) by interacting with $\mathcal{F}$. The so called *Composability Theorems* extend the above idea to an arbitrary application context. Consider a protocol $\rho|\mathcal{F}$, where the $|$ operator denotes protocol composition, that realizes an ideal functionality $\mathcal{G}$; then the protocol $\rho|\pi$ also realizes $\mathcal{G}$ if the protocol π realizes the ideal functionality $\mathcal{F}$.

The composability of security proofs depends on how generic their communication modeling is. To enable security proofs to be reused, ideal functionalities must be independent of concrete setups. For instance, suppose that F can be realized by the protocols π, $\alpha|\beta$, and $\gamma|\delta|\mathcal{H}$. Then, F should abstractly represent all of these protocols by capturing their essential properties. In particular, F must be independent of the number of sub-protocols and their interactions, i.e., the semantics of the $|$ operator, in its realizations. Tying an ideal functionality to a particular execution flow essentially prevents all the realizations that do not follow this flow.

2.1.1 *UC-style Frameworks*

In UC-style simulation-based frameworks [1, 16, 27, 35], (sub-)protocols are modeled using interactive Turing machines with network tapes and unique identifiers. At each point of a protocol's execution, only one of these machines is active and the others wait for new inputs. A special Turing machine, called the adversary, schedules the activation of protocol components: it is activated after each non-adversary machine halts and determines the next Turing machine to activate. The adversary plays the role of a mediator too. When two machines want to communicate, the sender informs the adversary about the message[1] content and destination, i.e., the receiving Turing machine's identifier, and the adversary forwards the message according to its underlying corruption model.

The generality of the communication model in simulation-based frameworks stems from the central adversary Turing machine that is universally quantified in security definitions. Consider the compound protocol $\rho|\mathcal{F}$, and interpret $|$ according to the execution model explained above. ρ's subroutine calls to $\mathcal{F}$ result in a transfer of the execution flow that is essential for the completion of ρ's execution. The subroutine call is carried out by two message broadcasts: the first message carries the "request" details while the second one conveys the "response" information. These messages need not be passed consecutively; therefore, each of ρ's subroutine calls may correspond to an arbitrary sequence of Turing machine activations starting with ρ's "request" message and ending with $\mathcal{F}$'s (or any of its realization's) "response" message. As such, $\mathcal{F}$'s semantics abstractly captures the execution of an arbitrary number (and order) of Turing machines realizing it.

The simulation-based frameworks' concrete approach to modeling can affect the rigor of their resulting proofs. As Camenisch et al. put it [15], protocol descriptions in simulation-based frameworks include meta-level and model specific information that makes them unnecessarily complicated; but, ignoring such details can lead to ill-defined specifications, sketchy proofs, and flawed results. The existing formal methods tools have not been able to alleviate the situation either. The level of details in simulation-based frameworks surpasses what the formal-methods community can reasonably handle with existing techniques. As such, the existing formalization results consider stripped-down versions of simulation-based frameworks

1 A message can be any information that is transferred between protocol components, including meta-level information, subroutine calls, or protocol-specific data.

by restricting the communication and corruption models [17]. Hence, they are not applicable in a wide range of scenarios.

2.2 TWO BASES OF THE THESIS

CryptHOL and Constructive Cryptography provide a firm mathematical footing for the thesis. The former facilitates the formalization, while the later provides the methodology.

2.2.1 *CryptHOL*

With the CryptHOL framework [8, 36], game-based cryptographic proofs can be formalized in higher-order logic (HOL) [23]. The proofs are mechanically checked by the proof assistant Isabelle/HOL [45], which ensures that every proof step is a valid application of HOL's logical inference rules. Games in CryptHOL are expressed as probabilistic functional programs whose semantic is expressed using discrete probabilities and *Generative Probabilistic Values (GPV)*, a denotational domain for probabilistic systems with inputs and outputs (I/O). From the semantics, CryptHOL derives proof rules for program equivalences and typical cryptographic arguments in game-based proofs.

Appendix A provides more details on the aforementioned concepts.

2.2.2 *Constructive Cryptography*

Constructive Cryptography (CC) [40–42] is an abstract paradigm for developing a theory of cryptography. Instead of focusing on concrete systems and proving their properties, CC studies system classes, i.e., the shared behavior of similar systems, and their transformations. As such, the notions of runtime, algorithms, and complexity are not intrinsic parts of every definition and proof in CC; they just represent classification strategies.

Modular reasoning about system classifications, which are called *specifications* in CC, is based on the concepts of *Resources* and *Converters*. Every aspect of individual systems, including the adversary's capabilities and information leakage is made explicit as a resource with named I/O interfaces. A specification is a set of resources. Parties can change a resource's behavior by attaching converters to their designated interfaces on that resource. It is possible to access multiple resources at the same time and attach many converters. For example, $x^a y^b \triangleright [R, S]$ denotes the attachment of converters x and y to resources R and S, where a and b are injective mappings between converter and resource interfaces and $[\_,\_]$ denotes the parallel access to two resources.[2] One can combine converters and interface attachments into a single *Protocol*[3] π and use the notation $\pi[R,S]$ instead. Let $\pi\mathcal{R} = \{\pi R \mid R \in \mathcal{R}\}$ denote the lifting of a protocol's attachment to specifications and define $[\mathcal{R},\mathcal{S}] = \{[R,S] \mid R \in \mathcal{R} \wedge S \in \mathcal{S}\}$. According to CC's terminology, π *constructs* the specification $\mathcal{S}$ from $\mathcal{R}$ if and only if $\pi\mathcal{R} \subseteq \mathcal{S}$. In the ideal-world–real-world terminology, $\pi\mathcal{R}$ and $\mathcal{S}$ play the role of the real-world implementation and ideal functionality respectively. The following *composability properties* hold for any protocols π and π' and arbitrary specifications $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{T}$:

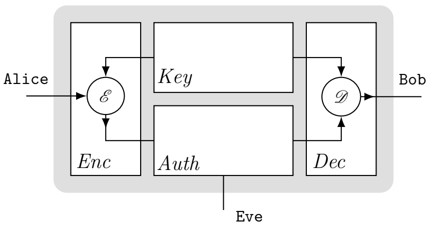
1. If $\pi\mathcal{R} \subseteq \mathcal{S}$ and $\pi'\mathcal{S} \subseteq \mathcal{T}$, then $\pi'\pi\mathcal{R} \subseteq \mathcal{T}$.

2. If $\pi\mathcal{R} \subseteq \mathcal{S}$, then $\pi[\mathcal{R},\mathcal{T}] \subseteq [\mathcal{S},\mathcal{T}]$.

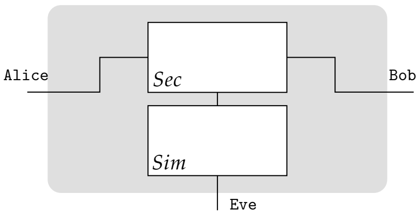
Common security notions are expressed as particular forms of specifications. For example, consider the simulator-based notion of information-theoretic security. Using a simulator σ corresponds to specifications of the form $\sigma\mathcal{S}$, where the simulator is defined in terms of a converter that only attaches to the adversary interfaces. Indistinguishability is analysed by *relaxing* specifications as $\mathcal{R}^{\epsilon} = \{S \mid R \in \mathcal{R} \wedge \mathfrak{d}(R,S) \leq \epsilon\}$, where $\mathfrak{d}(R,S)$ is the least upper bound on the advantages of all distinguishers in distinguishing the resources R and S. To define information-theoretic security, we combine these two forms: in a two-party setting with parties A and B and the adversary E, the construction of a specification $\mathcal{S}$ from $\mathcal{R}$ using the protocol $\pi = x^{\mathtt{A}}y^{\mathtt{B}}$ is information-theoretically secure iff there exist a simulator $\sigma = z^{\mathtt{E}}$ such that $\pi\mathcal{R} \subseteq (\sigma\mathcal{S})^{\epsilon}$. Computational security is defined similarly using computational indistinguishability relaxation $\Box^{<\epsilon>}$, which is defined like $\Box^{\epsilon}$ except that it considers computationally bounded distinguishers. In the above definition, note that we are using party names as a placeholder for interface mappings. Henceforth we drop the distinction between simulators and the converter that they attach and refer to both σ and z as the simulator.

2 The order of resources is unimportant in this notation; however, one can reorder converters only if they attach to disjoint sets of interfaces.

3 The meaning of the term "protocol" here is different from UC-style frameworks, where it refers to one or more Turing machines working together.



(A) The real world



(B) The ideal world

FIGURE 2.1: `Alice`, `Bob`, and `Eve` are the interface names. $\mathscr{E}$ and $\mathscr{D}$ denote the encryption and decryption functions, respectively. Each gray rectangle represents the resource that is constructed from the resources and converters in its interior.

2.3 A RUNNING EXAMPLE

We now introduce our running example: the construction of a secure communication channel. In this section, we consider the case where, using encryption, we construct such a channel from an authenticated channel and a key shared between two parties Alice and Bob. In Section 5.4, we will extend this construction using a message authentication code. Section 6.4 demonstrates another extension in which we use Diffie-Hellman key exchange.

A secure communication channel between two parties Alice and Bob can be established using a shared key and an authenticated channel: the parties use the key to encrypt their message and transmit the ciphertext via the authenticated channel. Figure 2.1a is a pictorial model of such a scenario in CC. Consider the protocol π that consists of the encryption converter *Enc* and the decryption converter *Dec*. Alice and Bob attach *Enc* and *Dec* to their sides respectively. The adversary Eve controls the communication network but does not have access to the pre-shared keys. This is modeled using the adversary interface on resources: the authenticated channel resource *Auth* allows the adversary to look at (but not edit) the channel's content and drop or delay messages; however, the key resource Key does not leak any information through its adversary interface.

Security is expressed using specifications. Let $\mathcal{R}_{auth-key}$ denote the specification that only contains the parallel composition of the $Auth$ and Key resources and let $\mathcal{S}_{sec}$ denote the singleton set that contains the ideal secure channel Sec. Here, Sec is a resource that is similar to $Auth$ except that it leaks the length of the channel's content. The encryption scheme used by the converters Enc and Dec is information-theoretically secure if there exists a simulator σ, represented as the Sim converter in the diagram, for which $\pi\mathcal{R}_{auth-key} \subseteq (\sigma\mathcal{S}_{sec})^{\epsilon}$. So, to prove the encryption scheme secure, it suffices to prove the indistinguishability of the two composed resources (the grey rectangles) in Figures 2.1a and 2.1b.

3 CONSTRUCTIONS' BASICS

This chapter formalizes the building blocks of cryptographic systems' constructions. As mentioned earlier, in CC every aspect of an individual system is made explicit as a resource with IO interfaces. Converters construct a new resource from assumed resource(s). They allow one to define complex resources in terms of simpler ones. We formalize resources in Section 3.1 and converters in Section 3.2. We then show they can be put together to build more complex systems in Section 3.3.

3.1 RESOURCES

A resource is a probabilistic reactive system that responds to inputs with outputs. For example, a randomness resource takes a natural number n as input and outputs n random bits. In general, the resource's outputs may depend on its previous inputs and outputs. For example, a random oracle takes an input and produces a random output for that input—unless the same input has previously been queried, in which case it returns the same output as before. It is therefore convenient to think of a resource as a probabilistic transition system given by a transition function tr and an initial state s_0; the transition function $tr(s, x)$ returns for every state s and every possible input x a probability distribution over the responses and the successor states.

In this representation, the internal state s is explicit. This complicates composition arguments that involve distinguishing resources based on their I/O behavior. We therefore introduce an abstraction that hides the resource's internal state. Formally, the resource type quantifies existentially over the state type in the pair of the transition function and the initial state:

$$\mathbb{R}(\mathring{q}, \mathring{r}) \equiv \exists \mathring{s}.\ (\mathring{s} \Rightarrow \mathring{q} \Rightarrow \mathbb{D}(\mathring{r} \times \mathring{s})) \times \mathring{s},$$

where $\mathring{q}$ represents the type of inputs, $\mathring{r}$ the type of outputs, and $\mathbb{D}(\mathring{a})$ probability distributions over $\mathring{a}$.

Fortunately, this abstraction can be expressed without existential types, which do not exist in HOL. In the co-algebraic view on reactive systems [48], the transition function $run : \mathbb{R}(\mathring{q}, \mathring{r}) \Rightarrow \mathring{q} \Rightarrow \mathbb{D}(\mathring{r} \times \mathbb{R}(\mathring{q}, \mathring{r}))$ for interacting with a resource defines a co-algebra on resources. That is, when we supply a resource with an input, we obtain a probability distribution over an output and a successor resource. In Isabelle/HOL, we formalize this view using the following co-datatype:

$$\textbf{codatatype}\ \mathbb{R}(\mathring{q},\mathring{r}) = \mathit{Resource}\ (\mathring{q} \Rightarrow \mathbb{D}(\mathring{r} \times \mathbb{R}(\mathring{q},\mathring{r}))).$$

The codatatype is the final coalgebra for the transition function *run*. Finality means that two resources with the same I/O behavior are equal. As we will show in Section 4.2, equality corresponds to bisimilarity of the probabilistic transition systems.

CryptHOL's GPVs are dual to our resources in the sense that GPVs query the environment and process the responses whereas resources produce responses to queries. In Section 5.1, we use GPVs to model distinguishers of resources.

3.1.1 *Defining Concrete Resources*

While hiding the internal state is suitable for reasoning abstractly about resources, keeping the state explicit is convenient for defining concrete resources and reasoning about them, since we can then specify properties of the internal states. We therefore introduce a function $\mathit{RES}(tr, s_0)$ that converts a probabilistic transition system with explicit states (given by the transition function tr and initial state s_0) into a resource. Categorically, this conversion is the morphism that makes the codataype the final coalgebra. Conceptually, it seals the resource and makes the state inaccessible, i.e., it introduces the existential type quantifier.

Such probabilistic transition systems with explicit states have already been formalized as part of CryptHOL, where they are used to model cryptographic oracles. We use the terms "oracle" and "probabilistic transition function" interchangeably in this thesis and reuse CryptHOL's infrastructure for formalizing and reasoning about them. In particular, CryptHOL's oracle composition operator $+_{\mathbb{O}}$ makes it possible to construct a resource from simpler parts. It interleaves two transition functions $tr_i : \mathring{s} \Rightarrow \mathring{q}_i \Rightarrow \mathbb{D}(\mathring{r}_i \times \mathring{s})$ (for $i = 1, 2$) operating on a shared state of type $\mathring{s}$ into one transition function $tr_1 +_{\mathbb{O}} tr_2 : \mathring{s} \Rightarrow \mathring{q}_1 + \mathring{q}_2 \Rightarrow \mathbb{D}((\mathring{r}_1 + \mathring{r}_2) \times \mathring{s})$, where ":" denotes each term's type and $\mathring{q} + \mathring{r}$ denotes the disjoint union of the types $\mathring{q}$ and $\mathring{r}$.

3.1.1.1 *Channel Resource Example*

As an example, we define a generic two-party single-usage communication channel. Following CC, we model this as a resource with three asynchronous interfaces for the sender (Alice), the receiver (Bob), and the adversary (Eve). They can all query their designated interface and receive an answer that depends on the channel's state. The channel can have the following states:

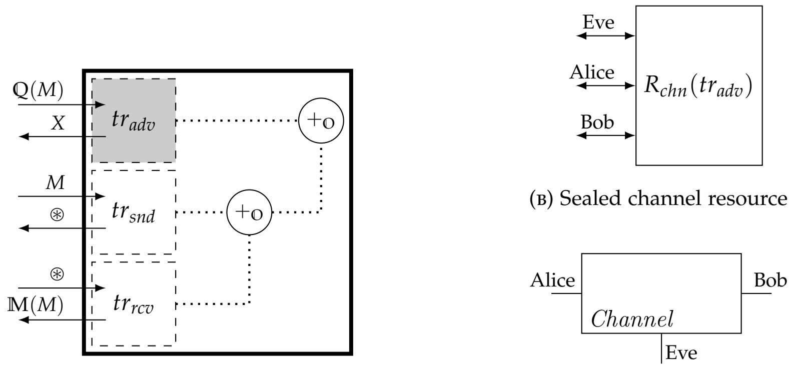
- empty, i.e the initial state.
- unusable, which cannot be revived to become usable again.
- containing a message of type M, which is on Alice's side or on Bob's side, and where Bob can only receive messages that are on his side.

Alice can send a message to an empty channel and receives $\circledast$, an arbitrary but fixed symbol, as an acknowledgement. Bob's query polls the channel; the response is either the message that is on his side or otherwise the special symbol *None* (the type constructor $\mathbb{M}(\mathring{a})$ adds this symbol to the type $\mathring{a}$). The adversary's interface determines the kind of channel. In all channels that we consider in this thesis, the adversary Eve can forward a message from Alice's side to Bob's side and drop it. Eve can read the message sent over an insecure or authenticated channel, but a secure channel leaks only the message's length, not its contents. An insecure channel additionally allows Eve to replace the message in the channel or insert a new message. Let $\mathbb{Q}(M)$ denote the type of Eve's queries. The outputs of type X on Eve's interface also depend on the channel's type. For example, a secure channel responds to a read request with the message length, whereas authentic and insecure channels return the message itself.

In Figure 3.1a, we show how $+_O$ and *RES* can be used to formalize the notion of a channel. The parties' interaction with the channel is modeled using three probabilistic transition systems tr_{snd}, tr_{rcv}, and tr_{adv}. The operator $+_O$ composes these oracles into a single transition system. Formally, inputs and outputs are tagged according to their location in the dotted term tree using the injections *Left* and *Right* for disjoint unions. For example, the queries to tr_{rcv} and their corresponding answers are tagged with *Right Right*. In the presentation, we usually omit these tags when they are clear from the context. The *RES* operator, visualized as the bold rectangle, seals the result of the composition and produces a channel resource $R_{chn}(tr_{adv})$ as shown in Fig. 3.1b. Formally, we write this as follows, where *empty* denotes the initial state:

$$R_{chn}(tr_{adv}) \equiv RES(tr_{adv} +_O tr_{snd} +_O tr_{rcv}, empty).$$

Figure 3.1c shows the same resource in the notation from Section 2.3. In the remainder of this thesis, we use the notation from Fig. 3.1b, where



(a) Internal construction

(b) Sealed channel resource

(c) Channel resource visualized in the style of Section 2.3

Figure 3.1: Formalization of a communication channel between Alice and Bob.

the different interfaces are combined into one using disjoint unions. In particular, we will not further describe the internal construction of resources, although all our concrete example resources are defined using *RES*.

The above construction accepts the transition function tr_{adv} for the adversary interface as an argument. We thus obtain secure R_{sec}, authentic R_{aut}, and insecure R_{isc} channels by providing the appropriate argument. For example, $R_{sec} \equiv R_{chn}(tr_{sec})$ gives a secure channel, for an appropriate definition of tr_{sec}.

3.1.2 *Type System for Interfaces*

Encoding several interfaces into one using tagging has a drawback: The type of resources does not enforce that the response is sent via the same interface as the query. For example, in Fig. 3.1, a query $\mathbb{Q}(M)$ to tr_{adv} must be responded over tr_{adv}'s response port (of type X) and not over tr_{rcv}'s. Clearly, resources constructed with *RES* and $+_{\mathbb{O}}$ ensure this property. But unless we look at the internal construction, it is not obvious that this property holds—and we do not want to constantly unfold our definitions. Yet, this property is crucial when reasoning about resources and converters. We therefore introduce a type system for interfaces that allows us to express arbitrary relations between inputs and outputs, i.e. arbitrary non-temporal, non-probabilistic properties of the resource.

An interface type $\mathcal{I} = (A, B)$ consists of a set A of inputs and a non-empty set of responses $B(a)$ for each input $a \in A$. We write $\mathcal{I}^A$ and $\mathcal{I}^B(a)$ for A and $B(a)$, respectively. Note that $\mathcal{I}^A$ is characterized by $\mathcal{I}^A = \{a \mid \mathcal{I}^B(a) \neq \{\}\}$.

Definition 1 (Respectfulness). *A resource R respects the interface type $\mathcal{I}$ (notation $\mathcal{I} \vdash R$) iff upon any input $a \in \mathcal{I}^A$, all responses of R are in $\mathcal{I}^B(a)$ and the resulting resource also respects $\mathcal{I}$. Formally, $\mathcal{I} \vdash R$ is defined co-inductively by the following rule:*

$$\frac{\forall a \in \mathcal{I}^A.\ \mathcal{P}\big[run(R,a) \in \mathcal{I}^B(a) \times \{R' \mid \mathcal{I} \vdash R'\}\big] = 1}{\mathcal{I} \vdash R},$$

where $\mathcal{P}[X \in A]$ denotes the probability that the discrete random variable X takes a value in the set A[1]*.*

For example, let $\mathcal{I}_{\text{len}}$ be given by $\mathcal{I}^B_{\text{len}}(m) = \{c \mid \|c\| = \|m\|\}$, where $\|x\|$ denotes the length of a list. Then, $\mathcal{I}_{\text{len}} \vdash R$ denotes that R is length preserving, i.e., the response has the same length as the input.

This notion of a resource respecting an interface type can express our desired property that responses are sent with the same tag as the queries. To that end, we define an operator $\oplus$ that combines interface types similarly to how $+_{\text{O}}$ combines transition functions. Formally, given two interface types $\mathcal{I}_1$ and $\mathcal{I}_2$, their combination $\mathcal{I}_1 \oplus \mathcal{I}_2$ is given by

$$(\mathcal{I}_1 \oplus \mathcal{I}_2)^B(a) = \begin{cases} \mathcal{I}_1^B(a) & \text{if } a \in \mathcal{I}_1^A \\ \mathcal{I}_2^B(a) & \text{if } a \in \mathcal{I}_2^A. \end{cases}$$

For example, let $\mathcal{I}_x$ be tr_x's interface type for $x \in \{\text{adv}, \text{snd}, \text{rcv}\}$. Then the channel resource R_{chn} in Fig. 3.1 respects $\mathcal{I}_{\text{adv}} \oplus \mathcal{I}_{\text{snd}} \oplus \mathcal{I}_{\text{rcv}}$ by construction. Once we have proven respectfulness, Isabelle's reasoning engine can exploit this property without exposing the resources' internal construction.

3.1.3 *Parallel Composition of Resources*

When building complex systems from simple resources, many resources are typically available at the same time. For example, in Fig. 2.1a, both a key and an authentic channel are available We model this using parallel composition for resources.

1 In case of a singleton set $A = \{\, x \,\}$, we write $\mathcal{P}[X = x]$ instead of $\mathcal{P}[X \in \{\, x \,\}]$.

Definition 2 (Parallel composition). *Let $R_1 : \mathbb{R}(\mathring{q}_1, \mathring{r}_1)$ and $R_2 : \mathbb{R}(\mathring{q}_2, \mathring{r}_2)$ be resources. The parallel composition $R_1 \mid\mid R_2 : \mathbb{R}(\mathring{q}_1 + \mathring{q}_2, \mathring{r}_1 + \mathring{r}_2)$ directs queries $\mathring{q}_1$ to R_1 and $\mathring{q}_2$ to R_2 and forwards the responses accordingly.*

In our formalization, we define parallel composition by primitive corecursion, exploiting the coalgebraic structure of the codataype $\mathbb{R}$. This applies to all operators on resources and converters that we present in this section. Often, CryptHOL provides a suitable operator on probabilistic transition systems that we just have to wrap into a resource or converter using primitive corecursion. This operator respects interface types: If $\mathcal{I}_1 \vdash R_1$ and $\mathcal{I}_2 \vdash R_2$, then $\mathcal{I}_1 \oplus \mathcal{I}_2 \vdash R_1 \mid\mid R_2$.

Note that $RES\ (tr_1 +_{O} tr_2, s)$ and $RES\ (tr_1, s) \mid\mid RES\ (tr_2, s)$ are *not* the same. Parallel resource composition ensures that the two resources do not share their state. In particular, probabilistic choices in one resource are independent of those in other resources. So $\mid\mid$ allows the occurrences of the states s to evolve independently, whereas $+_{O}$ interleaves tr_1 and tr_2 on the shared state s. Consequently, if correlation or state sharing is required, $+_{O}$ and RES must be used. Hence, $+_{O}$ cannot be lifted to the abstract level of resources.

3.2 CONVERTERS

Converters are probabilistic reactive systems that internally use other reactive systems. In other words, a converter transforms a resource into another resource. For example, suppose that we want to construct a uniform randomness resource, whose outputs are uniformly distributed over $\{1, \ldots, n\}$ upon input n, from the randomness resource mentioned in Section 3.1. Such a converter could be deterministic, taking all randomness from the interaction with the randomness resource.

A converter has two interfaces: Inputs and outputs are sent over the external interface, and to compute a response, the converter itself may send queries over the internal interface and wait for responses. So a converter is a probabilistic transition system with two layers: On the outer, external layer, it appears like a resource. However, every transition may consist of many internal transitions that drive the interaction with the resource that the converter transforms. This is the second, internal layer. In Fig. 2.1a, the converters *Enc* and *Dec* transform the resources *Key* and *Auth*. In our diagrams in the remainder of the thesis, the external interface is always on the left and the internal interface on the right of a converter.

The probabilistic transition system for the internal layer has already been formalized in CryptHOL as generative probabilistic values (GPVs): $\mathbb{G}(\mathring{o},\mathring{q},\mathring{r})$ represents the GPVs with result $\mathring{o}$, queries $\mathring{q}$, and responses $\mathring{r}$. To obtain a converter, we embed them into the outer layer using a transition function that returns a GPV rather than a probability distribution over the response and successor state. Analogous to resources, we define converters coalgebraically to hide the converter's state:

$$\textbf{codatatype}\ \mathbb{C}(\mathring{x},\mathring{y},\mathring{q},\mathring{r}) = \mathit{Converter}\ (\mathring{x} \Rightarrow \mathbb{G}(\mathring{y} \times \mathbb{C}(\mathring{x},\mathring{y},\mathring{q},\mathring{r}),\mathring{q},\mathring{r})).$$

Note the similarity to $\mathbb{R}$'s definition: we have merely replaced the probability functor $\mathbb{D}$ with the GPV functor $\mathbb{G}(\_,\mathring{q},\mathring{r})$.

Similar to *RES* for resources, we define the operator $\mathit{CNV}(\delta, s_0)$ that seals a CryptHOL interceptor and hides the internal state. In CryptHOL, interceptors express reductions between games. As before, this operator allows us to reuse CryptHOL's infrastructure for GPVs. Furthermore, we extend the notion of respecting interface types to converters.

Definition 3 (Respectfulness). *A converter C respects interface types $\mathcal{I}_1$ and $\mathcal{I}_2$ (notation $\mathcal{I}_1 \vdash C \dashv \mathcal{I}_2$) iff for any input $x \in \mathcal{I}_1^A$, the converter C will only issue queries in $\mathcal{I}_2^A$ on its internal interface and, provided that the responses to these queries y are in $\mathcal{I}_2^B(y)$, the converter's response will be in $\mathcal{I}_1^B(x)$ and the resulting converter also respects $\mathcal{I}_1$ and $\mathcal{I}_2$.*

Converters come with sequential and parallel composition, enabling complex systems to be built from simpler ones.

- Let $C_1 : \mathbb{C}(\mathring{x}_1,\mathring{y}_1,\mathring{q}_1,\mathring{r}_1)$ and $C_2 : \mathbb{C}(\mathring{x}_2,\mathring{y}_2,\mathring{q}_2,\mathring{r}_2)$ be two converters. Their parallel composition $C_1 \mid C_2 : \mathbb{C}(\mathring{x}_1 + \mathring{x}_2,\mathring{y}_1 + \mathring{y}_2,\mathring{q}_1 + \mathring{q}_2,\mathring{r}_1 + \mathring{r}_2)$ makes both converters available at the same time, analogous to parallel resource composition. If $\mathcal{I}_1 \vdash C_1 \dashv \mathcal{I}_1'$ and $\mathcal{I}_2 \vdash C_2 \dashv \mathcal{I}_2'$, then $\mathcal{I}_1 \oplus \mathcal{I}_2 \vdash C_1 \mid C_2 \dashv \mathcal{I}_1' \oplus \mathcal{I}_2'$.

- Let $C_1 : \mathbb{C}(\mathring{x},\mathring{y},\mathring{m},\mathring{n})$ and $C_2 : \mathbb{C}(\mathring{m},\mathring{n},\mathring{q},\mathring{r})$ be two converters. Their sequential composition $C_1 \odot C_2 : \mathbb{C}(\mathring{x},\mathring{y},\mathring{q},\mathring{r})$ uses C_2 to answer C_1's queries on C_1's internal interface. If $\mathcal{I}_1 \vdash C_1 \dashv \mathcal{I}_2$ and $\mathcal{I}_2 \vdash C_2 \dashv \mathcal{I}_3$, then $\mathcal{I}_1 \vdash C_1 \odot C_2 \dashv \mathcal{I}_3$.

- The identity converter $C_{\mathbb{1}} : \mathbb{C}(\mathring{q},\mathring{r},\mathring{q},\mathring{r})$ simply forwards all queries and responses from the external to the internal interface and vice versa. It is the neutral element for sequential composition: $C_{\mathbb{1}} \odot C = C \odot C_{\mathbb{1}} = C$. Clearly, $\mathcal{I} \vdash C_{\mathbb{1}} \dashv \mathcal{I}$.

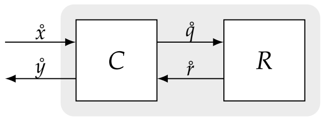


FIGURE 3.2: Attaching a converter to a resource.

Sometimes we will embed one interface type $\mathcal{I}_1$ into another $\mathcal{I}_2$. An embedding (f, g) is a pair of functions f and g such that f maps $\mathcal{I}_1^A$ to $\mathcal{I}_2^A$ and g maps $\mathcal{I}_2^B(f(x))$ to $\mathcal{I}_1^B(x)$ for all $x \in \mathcal{I}_1^A$. An *embedding converter* for an embedding is a converter C that merely applies these two functions. That is, when it receives an input $x \in \mathcal{I}_1^A$, it sends $f(x)$ on its internal interface, and when it receives the response $y \in \mathcal{I}_2^B(f(x))$ on the internal interface, it outputs the response $g(y)$ on its external interface. In particular, we have $\mathcal{I}_1 \vdash C \dashv \mathcal{I}_2$. The identity converter $C_{\mathbb{1}}$ is an embedding converter for the trivial embedding $f = g = id$. The sequential composition of embedding converters is again an embedding converter.

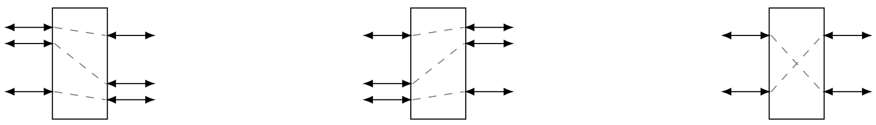
3.3 CONSTRUCTING SYSTEMS

We now show how to build complex systems. In the simplest case, we want to attach a converter to a resource (Fig. 3.2). Let $C : \mathbb{C}(\mathring{x}, \mathring{y}, \mathring{q}, \mathring{r})$ be a converter and $R : \mathbb{R}(\mathring{q}, \mathring{r})$ be a resource. Attaching C to R creates a new resource $C \rhd R : \mathbb{R}(\mathring{x}, \mathring{y})$, where R responds to C's queries.

Formally, the attachment operator $\rhd$ is defined using CryptHOL's *exec* operator for composing GPVs and oracles. Attachment respects interface types: If $\mathcal{I}_1 \vdash C \dashv \mathcal{I}_2$ and $\mathcal{I}_2 \vdash R$, then $\mathcal{I}_1 \vdash C \rhd R$. Attachment also interacts nicely with the other composition operators:

$$\begin{gathered} (C \odot C') \rhd R = C \rhd (C' \rhd R) \qquad (3.1) \\ (C \mid C') \rhd (R \,||\, R') = (C \rhd R) \,||\, (C' \rhd R') \\ C_{\mathbb{1}} \rhd R = R \end{gathered}$$

These properties are much more concise than the corresponding equations in CryptHOL, where the internal state is not hidden in the coalgebraic view. For example, the CryptHOL equivalent to (3.1) reads as follows, where $\widehat{\mathbb{D}}(f)(X)$ applies the function f to the random variable X.



(A) *lassocr* (B) *rassocl* (C) *swap*

FIGURE 3.3: Wiring converters. To signify the order of interfaces composition, we depict their representing arrows close (or far) from each other.

$$\begin{aligned}
&exec((R,s), inline((C',s'),C)) = \widehat{\mathbb{D}}(\lambda(x,(s',s)).\,((x,s'),s))(\\
&\quad exec((\lambda((s',s),y).\, \widehat{\mathbb{D}}(\lambda((x,s'),s).\,(x,(s',s)))(\\
&\qquad\qquad exec((R,s), C'(s',y))\\
&\qquad),(s',s)),C))
\end{aligned}$$

The equation is structurally the same (*inline* corresponds to $\odot$), but the essence is buried under the clutter that explicit state-passing introduces. This demonstrates the gain in abstraction that our formalization provides over CryptHOL.

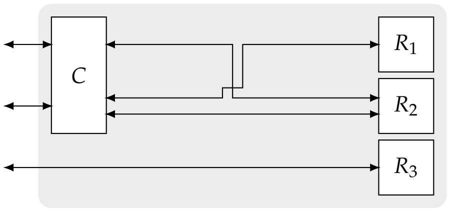
3.3.1 *Wirings*

The order of converter queries does not always correspond to the resource interfaces, which happens when resources' and converters' interfaces are composed in a different order. For example, $\mathcal{I}_1 \oplus (\mathcal{I}_2 \oplus \mathcal{I}_3)$ is not the same as $(\mathcal{I}_1 \oplus \mathcal{I}_2) \oplus \mathcal{I}_3$ because disjoint union is not associative in HOL. We therefore introduce three embedding converters, called *Wiring Converters*, that rearrange interfaces into the desired order, as shown in Fig. 3.3.[2]

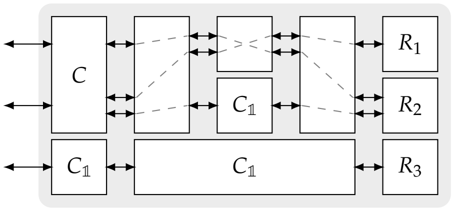
1. *lassocr* re-associates disjoint unions from left to right,
2. *rassocl* re-associates disjoint unions from right to left, and
3. *swap* exchanges the two sides of a disjoint union.

By composing the identity and wiring converters sequentially and in parallel, we can express arbitrary attachments of converters and resources.

2 In Fig. 3.3, the arrow's closeness indicates the association of the disjoint union, i.e., the order of interface composition. Except for Figs. 3.3 and 3.4, we omit this detail in the diagrams. Of course, we take care of re-associations in our formalization.



(A) Abstract representation



(B) Detailed construction

FIGURE 3.4: The result of attachment is the new resource shown with dark-gray.

More precisely, we can express that a converter should be attached only to a subset of the interfaces of a resource, and we can arbitrarily re-associate and permute this subset since every permutation can be expressed as a sequence of transpositions of adjacent positions. Figure 3.4a shows an example where a converter C with two external interfaces and three internal interfaces (associated to the right) is attached to two resources R_1 and R_2. Another resource R_3 is available, but not involved in the attachment. Figure 3.4b visualizes how such a diagram is formalized using the wiring converters:

- Parallel composition of C with $C_{\mathbb{1}}$ focuses on the interfaces R_1 and R_2 and leaves R_3's interface unchanged.
- The converter C', which is attached in the middle, combines three wiring converters: *rassocl* on the left and *lassocr* on the right bring together the two interfaces whose order must be swapped; and *swap* then swaps the order, where the parallel composition with $C_{\mathbb{1}}$ determines on which interfaces *swap* acts.

Formally, the attachment focused on R_1 and R_2's interfaces is expressed as $C' = \mathit{rassocl} \odot (\mathit{swap} \mid C_{\mathbb{1}}) \odot \mathit{lassocr}$ and the whole system is given by $(C \mid C_{\mathbb{1}}) \rhd (C' \mid C_{\mathbb{1}}) \rhd ((R_1 \mid\mid R_2) \mid\mid R_3)$.

4 CONSTRUCTIONS' EQUIVALENCE

We need equivalence notions that identify different constructions of a particular resource. As mentioned in Section 2.2.2, proofs in CC abstract over the unnecessary details of cryptography systems and focus on their shared behavior. In particular, security statements must be independent of how resources are constructed.

We introduce three equivalence notions in this thesis: bisimilarity, trace equivalence, and indistinguishability. This chapter introduces the first two notions, and we explain indistinguishability in Chapter 5. As we will show it, bisimilarity is stronger than trace equivalence (Cor. 1) and trace equivalence is stronger than indistinguishability (Thm. 3). Indistinguishability is the notion that we will later use for the security definitions in Section 5.2. However, the other two notions are also useful since many steps in security proofs actually establish one of these stronger equivalences.

In particular, notions of bisimilarity and trace equivalence simplify the reasoning in two respects. First, they provide simple proof rules. For example, trace equivalence only requires relational reasoning about distribution on states (Thm. 1), while relations between individual states suffice for bisimulation. Second, when two systems satisfy a stronger equivalence, we may replace one with the other in more contexts, so we must check fewer conditions on the context to justify the replacement. The case study in Section 5.4 gives some examples on where to use these two notions.

We introduce bisimilarity in Section 4.1 and demonstrate why it is too strong sometimes. We then explain trace equivalence in Section 4.2 and present its proof rules.

4.1 BISIMILARITY

In the co-algebraic view of systems, bisimilarity is the canonical equivalence notion. For resources, it is defined as follows:

Definition 4 (Bisimilarity). *Consider two resources R_1 and R_2 of type $\mathbb{R}(\mathring{q}, \mathring{r})$ that respect the interface type $\mathcal{I}$. Without loss of generality, assume that $R_i =$*

RES (tr_i, s_i) *for* $i \in \{1,2\}$.[1] *A relation* X *between the state spaces* $\mathring{s}_1$ *and* $\mathring{s}_2$ *is a bisimulation relation if and only if*

1. *X relates the two initial states* s_1 *and* s_2*, and*
2. *whenever* $(s'_1, s'_2) \in X$ *and* $a \in \mathcal{I}^A$*, there is a joint probability distribution* $d : \mathbb{D}(\mathring{r} \times \mathring{s}_1 \times \mathring{s}_2)$ *with marginal distributions* $tr_1(s'_1, a)$ *and* $tr_2(s'_2, a)$ *whose support is contained in* $\mathcal{I}^B(a) \times X$.

Two resources are bisimilar iff there is a bisimulation relation for them.

Bisimilarity is well-suited for proving resource equivalence since it is compositional and has elegant connections with relational parametricity (see Basin et al. [8]). For example, probabilistic relational Hoare logic [4, 6] as used in EasyCrypt [3] establishes bisimilarity. In fact, if the interface type $\mathcal{I}$ allows all queries and responses (i.e., $\mathcal{I}(a) = UNIV$ for all a where $UNIV$ denotes the set of all elements of a type), then bisimilarity coincides with logical equality in HOL as we model resources as a codatatype.

But, bisimilarity sometimes is too strong. For example, consider two resources that accept $\circledast$ as input and respond with an element from the set $\{a, b, c\}$. The diagrams in Fig. 4.1 show the behavior of the two resources as probabilistic transition systems. Every edge is labelled with the response and the probability that the edge is taken during an interaction. Starting in the state at the top, both systems respond with a to the first query and with b and c with probability 1/2 each to the second query. Thus, no distinguisher can distinguish the two systems through interaction. However, the system on the left decides already in the first interaction whether it will respond with b or c in the second interaction. In contrast, the system on the right makes this choice only during the second interaction. The two systems are therefore not bisimilar: For if X was a bisimulation relation, it would have to relate s_1 and s'_1 each with t_1. However, this is impossible because t_1 can output both b and c whereas s_1 and s'_1 can each only produce one of them.

In a non-deterministic setting, when considering outputs without probabilities, this is the standard example that separates bisimilarity from the coarser notion of trace equivalence. Intuitively, the problem in Fig. 4.1 is that the states s_1 and s'_1 encode hidden non-determinism that will be unveiled later. An observer of the system on the left does not yet know whether the system is in s_1 or s_2. It knows only a *distribution* over those states, where each state is assigned with the likelihood that the system is in that state. This distribution is given by the transition probabilities of the system

1 Every resource R can be expressed as *RES* (tr, s_0) by choosing $tr = run$ and $s_0 = R$.

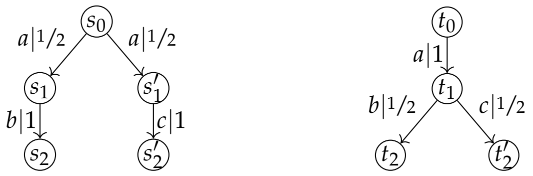


FIGURE 4.1: A simple example with two probabilistic transition systems that are trace equivalent, but not bisimilar.

conditioned on the past interactions. Here, this conditional distribution is uniform over s_1 and s'_1. These conditional distributions are the probabilistic equivalent of the well-known powerset construction for non-deterministic automata. If we ignore the probabilities, then the support of the conditional distribution denotes the set of states reachable under the given input.

4.2 TRACE EQUIVALENCE

In this section, we will define trace equivalence for resources and present a sound and complete proof rule for trace equivalence. The trace definition and our proof rule follows the *unwinding* style of bisimilarity in Def. 4: We use the conditional distribution to summarize a partial trace. This way, we can extend the partial trace with the next interaction by combining the transitions from the states in the support of the conditional distribution according to their weight in the distribution. This is a local operation since we only need to consider the transitions from states in the support. Accordingly, we can establish trace equivalence by a local argument too: it suffices to exhibit a relation between these summaries that is preserved by the transitions. In Section 5.1, we will show that trace equivalence of resources captures the notion of perfect indistinguishability that is typically used in cryptography. We thus obtain a local proof rule that is complete for perfect indistinguishability.

Traces and trace equivalence and their unwinding characteristics have previously been studied in the coalgebraic approach to modelling systems [26, 28, 34, 50]. Unfortunately, none of these results can be applied directly to resources in our setting because they need either a ccpo structure or a decomposition of the coalgebra's functor $\mathtt{F}$ into a functor $\mathtt{G}$ and a monad $\mathtt{T}$ such that $\mathtt{F}(\mathtt{x}) = \mathtt{G}(\mathtt{T}(\mathtt{x}))$ or $\mathtt{F}(\mathtt{x}) = \mathtt{T}(\mathtt{G}(\mathtt{x}))$. For resources, we have $\mathtt{F}(\mathtt{r}) = \mathring{q} \Rightarrow \mathbb{D}(\mathring{r} \times \mathtt{r})$ with $\mathtt{T} = \mathbb{D}$ as the monad, but no ccpo structure and no suitable decomposition. Abstractly, this is because the summaries, i.e., conditional distributions over states, are not explicit in our representation. Fortunately, a distribution $\mathbb{D}(\mathring{r} \times r)$ over pairs can be equivalently represented as two parts: the marginal distribution over the first component $\mathbb{D}(\mathring{r})$ and a family of conditional distributions over the second component (the summaries), indexed by the support of the first component: $\mathring{r} \Rightarrow \mathbb{D}(r)$. Thus, we can think of resources as a coalgebra of the functor $\mathtt{F}'(\mathtt{r}) = \mathring{q} \Rightarrow (\mathbb{D}(\mathring{r}) \times (\mathring{r} \Rightarrow \mathbb{D}(\mathtt{r})))$, which we can decompose as $\mathtt{G}(\mathbb{D}(r))$ with $\mathtt{G}(\mathtt{y}) = \mathring{q} \Rightarrow (\mathbb{D}(\mathring{r}) \times (\mathring{r} \Rightarrow \mathtt{y}))$. To our knowledge, this functor has not yet been considered in the literature. In the remainder of this section, we apply the abstract theory to resources and define traces and trace equivalence.

The trace of a resource R is a function from past I/O pairs to a family of sub-probability distributions over outputs, indexed by inputs. That is,

$$trace : \mathbb{R}(\mathring{q}, \mathring{r}) \Rightarrow \mathbb{L}(\mathring{q} \times \mathring{r}) \Rightarrow \mathring{q} \Rightarrow \mathbb{D}(\mathring{r}),$$

where $\mathbb{L}(\mathring{q} \times \mathring{r})$ denotes the type of lists of pairs $\mathring{q} \times \mathring{r}$.

In the example resource on the left of Fig. 4.1, the trace of the top state probabilistically combines the traces of its two successor states. When a system interacts with this resource, it knows after the first interaction only that the resource is in state s_1 with probability $1/2$ and in s_2 with the same probability. The system will learn whether it is s_1 or s_2 only during the second interaction.

This example shows that we must support reasoning about probability distributions on the states. This entails that we define traces for distributions over resources.[2] The trace of a single resource R then is the special case for the one-point distribution $dirac(R)$. Formally, if $p : \mathbb{D}(\mathbb{R}(\mathring{q}, \mathring{r}))$ is a

2 If we ignore probabilities and consider non-deterministic systems, we can directly define the traces for a state, without first going to sets of states. We now show that this does not work for probabilistic systems. In a non-deterministic system, the corresponding trace function returns the set of possible responses (rather than their distribution) for every input given the previous I/O pairs, i.e., $trace : \mathbb{R}(\mathring{q}, \mathring{r}) \Rightarrow \mathbb{L}(\mathring{q} \times \mathring{r}) \Rightarrow \mathring{q} \Rightarrow \mathbb{P}(\mathring{r})$ where $\mathbb{P}(\mathring{r})$ denotes the powerset of $\mathring{r}$. This function *can* be defined recursively over the list of I/O pairs by taking the union over the possible successor states: $trace(s, (x, y) \cdot l, a) = \bigcup_{(y,s') \in \delta(s,x)} trace(s', l, a)$.
This definition only works because conditioning on the output y distributes over unions: $\{s \mid (s, y) \in \bigcup \mathfrak{A}\} = \bigcup_{A \in \mathfrak{A}} \{s \mid (s, y) \in A\}$ holds for all y and $\mathfrak{A}$. For probability distributions, the corresponding identity, taking the weighted combination instead of the union, does not hold. Therefore, our generalization to distributions over states (or resources) is necessary. (The pointwise non-deterministic definition can be interpreted in probabilities too, but this leads to the wrong definition of trace equivalence.) Categorically speaking, the powerset functor $\mathbb{P}$ is additive, but the distribution functor $\mathbb{D}$ is not [19].

distribution of resources, let $\overline{run}(p,a)$ denote the weighted combination of running the resources from p with input a, i.e.,

$$\mathcal{P}\left[\overline{run}(p,a) = (b,R')\right] = \sum_{R \in support(p)} \mathcal{P}[p = R] \cdot \mathcal{P}\left[run(R,a) = (b,R')\right].$$

We next define the traces by recursion over the list of past I/O pairs:

$$\begin{aligned}
&trace : \mathbb{D}(\mathbb{R}(\mathring{q},\mathring{r})) \Rightarrow \mathbb{L}(\mathring{q} \times \mathring{r}) \Rightarrow \mathring{q} \Rightarrow \mathbb{D}(\mathring{r}) \\
&trace(p,[\,],a) = \widehat{\mathbb{D}}(\pi_1)(\overline{run}(p,a)) \\
&trace(p,(x,y)\cdot l,a) = trace(\overline{run}(p,x)\upharpoonleft_y, l, a)
\end{aligned}$$

Here, $[\,]$ denotes an empty list and $(x,y)\cdot l$ represents the prepending of list l with a pair (x,y). Furthermore, $\widehat{\mathbb{D}}(\pi_1)(p)$ denotes p's marginal on the first component and $p\upharpoonleft_x$ conditions the subprobability distribution $p : \mathbb{D}(\mathring{q} \times \mathring{r})$ on the event $\{(x,y) \mid y \in \mathring{r}\}$ and projects the result to the second component. That is,

$$\mathcal{P}[p\upharpoonleft_x = y] = \begin{cases} \frac{\mathcal{P}[p=(x,y)]}{\sum_{y'} \mathcal{P}[p=(x,y')]} & \text{if } x \in support(p) \\ 0 & \text{otherwise.} \end{cases}$$

Two resources are trace equivalent if they have the same traces. Formally:

Definition 5 (Trace equivalence). *Let p_1 and p_2 be two distributions over resources that respect $\mathcal{I}$. They are* trace equivalent *iff $trace(p_1,l,x) = trace(p_2,l,x)$ for all $x \in \mathcal{I}^A$ and lists l of pairs whose first component is in $\mathcal{I}^A$. Two resources R_1 and R_2 respecting $\mathcal{I}$ are* trace equivalent *iff $dirac(R_1)$ and $dirac(R_2)$ are trace equivalent.*

Trace equivalence is a property of a resource as a whole as the above example has shown. Consequently, we cannot easily prove trace equivalence by inspecting the individual steps of the underlying transition function like in a bisimulation proof. Nevertheless, the following characterization yields a proof principle for establishing trace equivalence similar to a bisimulation-style proof rule:

Theorem 1 (Trace equivalence characterisation). *Two resources R_1 and R_2 respecting $\mathcal{I}$ are trace equivalent iff there exists a relation X between distributions of resources such that*

1. *X relates $dirac(R_1)$ to $dirac(R_2)$, and*

2. *Whenever $(p_1, p_2) \in X$ and $a \in \mathcal{I}^A$, then $\overline{run}(p, a)$ and $\overline{run}(p_2, a)$ have the same marginal distribution on outputs and $(run(p_1, a)\upharpoonright_b, run(p_2, a)\upharpoonright_b) \in X$ for all $(b, \_) \in support(\overline{run}(p_2, a))$.*

Corollary 1. *Bisimilar resources are trace equivalent.*

For example, Thm. 1 allows us to prove trace equivalence of the two resources in Fig. 4.1 in a bisimulation style. We pick as the relation X the following four pairs of probability distributions, identifying a state with the resource with that initial state. The notation $[x_1|p_1, x_2|p_x, \ldots, x_n|p_n]$ denotes the probability distribution where the elementary event x_i has probability p_i.

- $([s_0|1], [t_0|1])$
- $([s_1|1/2, s'_1|1/2], [t_1|1])$
- $([s_2|1], [t_2|1])$
- $([s'_2|1], [t'_2|1])$

It is easy to verify that X satisfies the two conditions in Thm. 1. The key is the second tuple, which relates the uniform distribution over s_1 and s'_1 to the one-point distribution on t_1. Although the transition system on the left has already decided in states s_1 and s'_1 what the next output will be, the uniform distribution hides this decision. Accordingly, both outputs b and c have probability 1/2 as the transition probabilities for the two states are combined according to the distribution on the states, i.e., uniformly.

4.2.1 *Trace Equivalence Upto*

In Theorem 1, the distributions in the relation X capture the so-far unobserved probabilistic choices of the resources; but in practice, the randomness may be observed only in parts or not at all in some runs. We frequently experienced such situations in our case studies, especially when the adversary interferes or a failure event happens. Yet, the closure condition on X in Thm. 1 requires that X contains all reachable combinations of distributions in such cases, even though we know that there is no point in conserving the randomness in this branch of the proof. This adds considerable bloat to the definition of X and to the trace equivalence proof because condition (2) must hold for every pair of distributions in X, which may require even further pairs in X, and so on. In some examples, these "unnecessary" cases significantly outnumbered the relevant cases.

To counter this case explosion, we introduce a closure operator $[\![X]\!]$ on X and generalize the above theorem to an up-to proof rule. The closure $[\![X]\!]$ of a relation X of distributions is defined inductively as follows, where $p \gg\!= f$ denotes the p-weighted combination of a family f of distributions. That is, $\mathcal{P}_{(p \gg= f)}[x] = \sum_y \mathcal{P}_p[y] \cdot \mathcal{P}_{f(y)}[x]$, where $\mathcal{P}_p[x]$ denotes the probability mass that p assigns to the elementary event x.

1. Whenever $(p_1, p_2) \in X$, then $(p_1, p_2) \in [\![X]\!]$.

2. Let f, g be two families of distributions over the same countable index set I and p be a distribution over I. If $(f(i), g(i)) \in [\![X]\!]$ for all i in p's support, then $(p \gg\!= f, p \gg\!= g) \in [\![X]\!]$.

We can now show the up-to version of the trace equivalence proof rule. It differs from Thm. 1 only in the closure $[\![X]\!]$ instead of X in condition 2.

Theorem 2 (Trace equivalence upto)**.** *Two resources R and S are trace equivalent iff there exists a relation X between distributions of resources such that*

1. *X relates $dirac(R)$ to $dirac(S)$.*

2. *Whenever $(p_1, p_2) \in X$, then for all queries a, $\overline{run}(p_1, a)$ and $\overline{run}(p_2, a)$ have the same marginal distribution on responses and, for each possible response b to the query a, $(\overline{run}(p_1, a)\upharpoonright_b, \overline{run}(p_2, a)\upharpoonright_b) \in [\![X]\!]$ holds.*

This theorem can significantly reduce the number of cases in a trace equivalence proof. In the above example with the unnecessary randomness, we discard the randomness with two constant families $f(i) = dirac(R)$ and $g(i) = dirac(S)$. Not only does this step reduce the number of proof cases, it also simplifies the remaining cases because these cases start from the single resource in the one-point distribution rather than a distribution of resources.

5 CONSTRUCTIONS' SECURITY

In this chapter, we define a coarser equivalence notion based on distinguishers and the ideal-world–real-world paradigm [22]. This enables us to identify the class of resources that are "almost" the same as a given idealized resource. The notion of trace equivalence, which captures systems equality from an observer's point of view, is still too strong for such identification since there is a small probability that the two systems are not the same, e.g. on the event that the observers guesses a secret.

We capture the coarse equivalence of resources using the concept of secure constructions. Consider an ideal resource that is intrinsically secure, i.e. it satisfies the property that we are interested in by definition. We obtain a relaxed description of the ideal resource by attaching a special converter, called the simulator. Definition 7 identifies any other resource, often called the real resource, with the ideal resource if it is *indistinguishable* from the relaxed ideal resource. In that case, the construction that results in the real resource[1] is deemed secure. Or, as it is common in UC-style frameworks, the real resource (securely) *realizes* the ideal resource[2].

We formalize distinguishers and the notion of secure constructions in Sections 5.1 and 5.2. In Section 5.3, we explain how secure constructions associate with resources and converters' composition operators (cf. Ch. 3). In Section 5.4, we show the applicability of the aforementioned ideas in a case study.

5.1 DISTINGUISHERS

A distinguisher is a probabilistic system that interacts with a resource and returns a Boolean. Unlike resources and converters, a distinguisher is not activated by external inputs; the distinguisher itself drives the system. Formally, a distinguisher of type $\mathbb{A}(\mathring{q}, \mathring{r})$ is a generative probabilistic value (GPV) that outputs a Boolean:

$$\textbf{type-synonym}\ \mathbb{A}(\mathring{q}, \mathring{r}) = \mathbb{G}(\mathbb{B}, \mathring{q}, \mathring{r}).$$

1 The real resource may be the outcome of attaching various resources and converters.
2 In this context, a resource resembles the idea of *functionalities* in UC-style frameworks.

When we connect a distinguisher $D : \mathbb{A}(\mathring{q}, \mathring{r})$ to a resource $R : \mathbb{R}(\mathring{q}, \mathring{r})$, we obtain a probability distribution $D \blacktriangleright R : \mathbb{D}(\mathbb{B})$ over Booleans. The connect operator $\blacktriangleright$ corresponds to CryptHOL's operator for connecting an adversary with an oracle.

A distinguisher $D : \mathbb{A}(\mathring{q}, \mathring{r})$ can absorb a converter $C : \mathbb{C}(\mathring{q}, \mathring{r}, \mathring{x}, \mathring{y})$. The result $D \ogreaterthan C : \mathbb{A}(\mathring{x}, \mathring{y})$ is again a distinguisher, but for resources of type $\mathbb{R}(\mathring{x}, \mathring{y})$. The absorption operator ($\ogreaterthan$) corresponds to CryptHOL's inline operator. Connection and absorption satisfy two important identities:

$$D \blacktriangleright (C \rhd R) = (D \ogreaterthan C) \blacktriangleright R$$
$$D \ogreaterthan (C \odot C') = (D \ogreaterthan C) \ogreaterthan C'$$

Like converters, a distinguisher must respect the interface type of a resource. The predicate $D \dashv \mathcal{I}$ expresses that the distinguisher D queries a resource only with inputs $x \in \mathcal{I}^A$, provided that the resource's responses are in $\mathcal{I}^B(x)$. The absorption operator preserves the interface types: If $D \dashv \mathcal{I}_1$ and $\mathcal{I}_1 \vdash C \dashv \mathcal{I}_2$, then $D \ogreaterthan C \dashv \mathcal{I}_2$.

Definition 6 (Advantage). *Let R_1, R_2 be two resources for the same interface type $\mathcal{I}$. The advantage adv(D, R_1, R_2) of a distinguisher D with $D \dashv \mathcal{I}$ is given by $|\mathcal{P}[D \blacktriangleright R_1 = \mathit{True}] - \mathcal{P}[D \blacktriangleright R_2 = \mathit{True}]|$.*

The next theorem shows that trace equivalence captures the notion of perfect indistinguishability, i.e., with advantage zero.

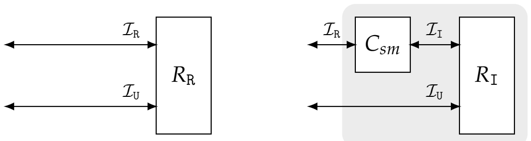
Theorem 3 (Characterisation of trace equivalence). *Let R_1 and R_2 be two resources respecting $\mathcal{I}$. Then the following are equivalent:*

- *R_1 and R_2 are trace equivalent.*
- *$D \blacktriangleright R_1 = D \blacktriangleright R_2$ for all distinguishers D with $D \dashv \mathcal{I}$.*

5.2 DEFINING SECURITY

We now define the security of a construction following the ideal-world–real-world paradigm. The ideal resource defines the secure "functionality", and we would like to show that a real resource realizes the same functionality, although it may differ with small probability.

Formally, we consider three interfaces: the user interface with type $\mathcal{I}_\mathrm{U}$ to the actual functionality, the adversary's interfaces with types $\mathcal{I}_\mathrm{I}$, and $\mathcal{I}_\mathrm{R}$ to the ideal and real resources, respectively. The ideal resource $R_\mathrm{I} : \mathbb{R}(\mathring{q}_\mathrm{I} + \mathring{q}_\mathrm{U}, \mathring{r}_\mathrm{I} + \mathring{r}_\mathrm{U})$ provides the ideal interface and the user interface: $\mathcal{I}_\mathrm{I} \oplus \mathcal{I}_\mathrm{U} \vdash R_\mathrm{I}$.



(A) Real resource (B) Ideal resource with simulator

FIGURE 5.1: Secure realization of an ideal resource.

The real resource $R_\mathtt{R} : \mathbb{R}(\mathring{q}_\mathtt{R} + \mathring{q}_\mathtt{U}, \mathring{r}_\mathtt{R} + \mathring{r}_\mathtt{U})$ provides the real interface and the user interface: $\mathcal{I}_\mathtt{R} \oplus \mathcal{I}_\mathtt{U} \vdash R_\mathtt{R}$.

In the running example, the two users Alice and Bob have their own interface. So in this setting, we combine Alice's interface $\mathcal{I}_{\text{snd}}$ and Bob's $\mathcal{I}_{\text{rcv}}$ into a single channel user interface $\mathcal{I}_\mathtt{U} = \mathcal{I}_{\text{snd}} \oplus \mathcal{I}_{\text{rcv}}$. Similarly, when the channel resource is combined with a key resource like in Fig. 2.1a, parallel resource composition leads to a combined interface that is the disjoint union of the adversary and user interfaces. We therefore rearrange the interfaces using wiring converters such that they obey the format "adversary interface $\oplus$ user interface"; Theorem 6 below shows the construction.

For a secure realization, we demand that a simulator C_{sm} can mimic the real interface based only on the ideal. The real resource typically provides a richer adversary interface than the ideal resource. Figure 5.1 illustrates the indistinguishability condition.

We only provide the security definition in a computational setting here. Information-theoretic security is defined analogously without putting a bound on the distinguisher's queries. Note that we cannot formally express the efficiency of computations due to the shallow embedding of our formalization library in HOL, which identifies terms up to computations. We follow CryptHOL's approach and make the reductions explicit to overcome this limitation. That is, in the formalization source [37, 38], we write ϵ as a function of a particular adversaryi's advantage in some computational assumption, e.g. the Decisional Diffie-Hellman game (DDH), instead of quantifying over all distinguishers like the following definition. As such, the formalization audience need to check the definitions and convince themselves that such an assumption makes sense.

Definition 7 (Computatational concrete security). *Let the resources $R_\mathtt{I}$ and $R_\mathtt{R}$ be given with the above interfaces Then, $R_\mathtt{R}$ ϵ-securely realizes $R_\mathtt{I}$ up to $\mathscr{B}$ interactions iff there exists a simulator, i.e., a converter $C_{sm} : \mathbb{C}(\mathring{q}_\mathtt{R}, \mathring{r}_\mathtt{R}, \mathring{q}_\mathtt{I}, \mathring{r}_\mathtt{I})$*

with $\mathcal{I}_\mathtt{R} \vdash C_{sm} \dashv \mathcal{I}_\mathtt{I}$ that is attached only to the adversary interface of the ideal resource, such that $adv(D, R_\mathtt{R}, (S \mid C_{\mathbb{1}}) \rhd R_\mathtt{I}) \leq \epsilon$ holds for all distinguishers $D : \mathbb{A}(\mathring{q}_\mathtt{R} + \mathring{q}_\mathtt{U}, \mathring{r}_\mathtt{R} + \mathring{r}_\mathtt{U})$ that respect the interface $\mathcal{I}_\mathtt{R} \oplus \mathcal{I}_\mathtt{U}$ and make at most $\mathscr{B}$ queries.

We extend this notion to asymptotic security by introducing a security parameter η and requiring that ϵ is negligible in η[3]. As is customary in CryptHOL [8], our formalization uses Isabelle/HOL's module system for that. Accordingly, we obtain an asymptotic security statement for every concrete security statement essentially for free.

Theorem 4 (Reflexivity)**.** *Every resource 0-securely realizes itself up to any bound with simulator $C_{\mathbb{1}}$.*

5.3 COMPOSABILITY OF SECURE CONSTRUCTIONS

We now show that the notion of secure realization composes. That is, secure realization is closed under three kinds of composition:

1. concatenation (transitivity),
2. parallel composition, and
3. attaching a converter to the user interface.

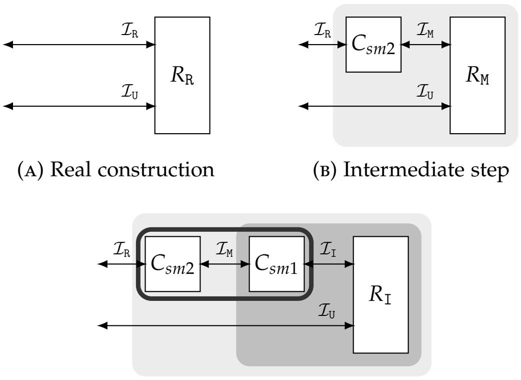
We will illustrate each of these in the case study in Section 5.4.

Concatenation stacks secure realizations on top of each other. Suppose that R_1 securely realizes R_2 and R_2 itself securely realizes R_3. Then, R_1 securely realizes R_3. So, secure realization is a transitive relation. The next theorem captures this property and the associated construction is visualized in Fig. 5.2.

Theorem 5 (Transitive composition)**.** *Let $\mathcal{I}_\mathtt{I} \oplus \mathcal{I}_\mathtt{U} \vdash R_\mathtt{I}$ and $\mathcal{I}_\mathtt{M} \oplus \mathcal{I}_\mathtt{U} \vdash R_\mathtt{M}$ and $\mathcal{I}_\mathtt{R} \oplus \mathcal{I}_\mathtt{U} \vdash R_\mathtt{R}$ such that $R_\mathtt{M}$ ϵ-securely realizes $R_\mathtt{I}$ up to $\mathscr{B}_1$ interactions and $R_\mathtt{R}$ ϵ'-securely realizes $R_\mathtt{M}$ up to $\mathscr{B}_2$ interactions. Let C_{sm1} and C_{sm2} be the respective simulators and $\mathscr{B}_s$ be a bound on the number of queries that C_{sm2} makes during one invocation. If $\mathscr{B}_2 * max(\mathscr{B}_s, 1) \leq \mathscr{B}_1$, then $R_\mathtt{R}$ $(\epsilon + \epsilon')$-securely realizes $R_\mathtt{I}$ up to $\mathscr{B}_1$ interactions with simulator $C_{sm1} \odot C_{sm2}$.*

Moreover, secure realizations is closed under parallel composition. Figure 5.3 visualizes this property.

3 An advantage is negligible if, as a function of the security parameter, it approaches 0 faster than any inverse polynomial.


$\mathcal{I}_{\mathtt{R}}$
$\mathcal{I}_{\mathtt{U}}$
$R_{\mathtt{R}}$
C_{sm2}
$\mathcal{I}_{\mathtt{M}}$
$R_{\mathtt{M}}$
C_{sm1}
$\mathcal{I}_{\mathtt{I}}$
$R_{\mathtt{I}}$

(A) Real construction (B) Intermediate step

(C) Ideal construction

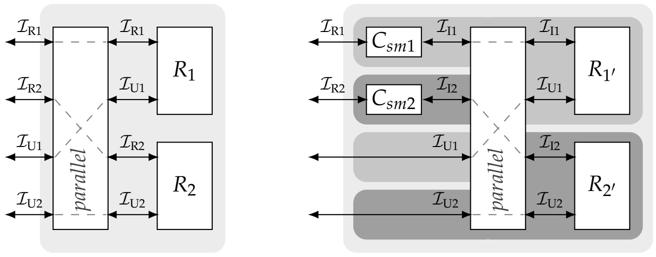
FIGURE 5.2: Transitivity of secure realizations. The real resource $R_{\mathtt{R}}$ is constructed from the middle resource $R_{\mathtt{M}}$ using converter S_2. $R_{\mathtt{M}}$ is constructed from the ideal resource $R_{\mathtt{R}}$ using converter S_1. Therefore, $R_{\mathtt{R}}$ can be directly constructed from $R_{\mathtt{I}}$ using the sequential composition of of S_1 and S_2, depicted using bold black rectangle.

Theorem 6 (Parallel composition)**.** *Let R_1 ϵ_1-securely realise R_1' up to $\mathscr{B}_1$ interactions and let R_2 ϵ_2-securely realise R_2' up to $\mathscr{B}_2$ interactions. Let parallel denote the converter lassocr $\odot$ $(C_{\mathbb{1}} \mid (rassocl \odot (swap \mid C_{\mathbb{1}}) \odot lassocr)) \odot rassocl$. Then, the parallel composition parallel $\odot$ $(R_1 \mid R_2)$ $(\epsilon_1 + \epsilon_2)$-securely realizes parallel $\odot$ $(R_1' \mid R_2')$ up to $min(\mathscr{B}_1, \mathscr{B}_2)$ interactions. The new simulator is the parallel composition of the old simulators.*

Together with Thm. 4, Thm. 6 yields that secure realizations is preserved under parallel composition contexts, i.e., if we add an arbitrary resource in parallel to a real and an ideal resource, then the resulting real resource securely realizes the resulting ideal resource.

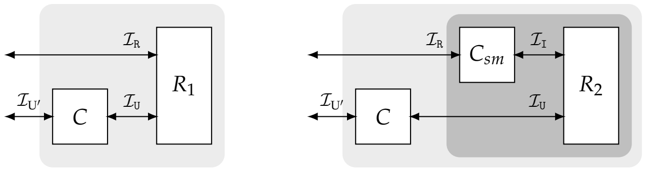
Third, secure realization is preserved when we attach a converter to the user interfaces (Fig. 5.4). There is no corresponding theorem for attaching a converter to the adversary interface. Changes to the adversary interface must be expressed as a composition of secure realizations (Thm. 5).

Theorem 7 (User's interfaces' composition)**.** *Let R_1 ϵ-securely realise R_2 up to $\mathscr{B}$ interactions. Let C be a converter on the user interface that performs at most $\mathscr{B}_C$ many interactions in a single invocation. Then, $(C_{\mathbb{1}} \mid C) \rhd R_1$ ϵ-securely realizes $(C_{\mathbb{1}} \mid C) \rhd R_2$ up to $\mathscr{B}'$ interactions if $\mathscr{B}' * max(\mathscr{B}_C, 1) \leq \mathscr{B}$ with the same simulator.*



(A) Real Construction (B) Ideal construction

Figure 5.3: Parallel composition of secure realizations.



(A) Real Construction (B) Ideal construction

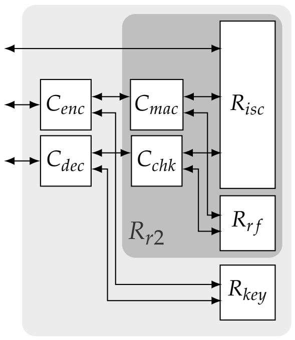
Figure 5.4: Attaching a converter to the user interface

5.4 A RUNNING EXAMPLE: MAC EXTENSION

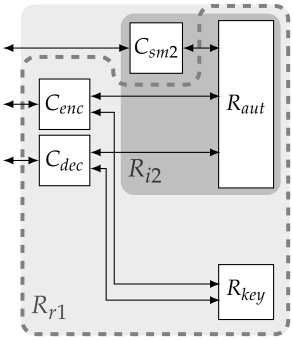
We now use the concepts presented in the previous sections to formalize the construction of a secure channel from a key, a random function, and an insecure channel. For brevity, we only provide a high-level summary and use pictures to present the main proof steps.

In each subfigure in the Figure 5.5, resources are depicted on the right, and the attachment of abstract systems $\triangleright$ is presented using arrows. Moreover, the resources or converters that are in the same column are composed in parallel using $\|$ or $|$ respectively. Following the lemmas about the sequential composition $\odot$ of converters in the Section 3.3, the arrow connecting two converters, e.g. the arrow between C_{sm2} and C_{sm1}, can be understood as either the attach operation or the sequential composition of converters.

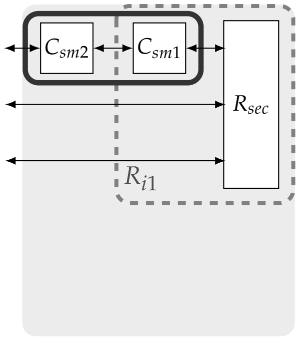
We use two cryptographic schemes in our construction. First, an encryption scheme with converters C_{enc} and C_{dec} that is used to construct a secure channel from an authentic channel and a key. Second, a message authentication code (MAC) scheme with converters C_{mac} and C_{chk} that is



(A) Construction using an insecure channel



(B) Replacing an authentic channel



(C) Replacing a secure channel

FIGURE 5.5: Construction of a secure channel using one-time-pad and a message authentication code.

used to construct an authentic channel from an insecure channel and a random function.[4] We have formalized these schemes in a generic way that admits multiple instantiations. We instantiate the encryption scheme with a one-time-pad, and we use the random function's mapping of each message as the message authentication code.

We work with five kinds of resources in our proof. The first three are the insecure channel R_{isc}, the authentic channel R_{aut}, and the secure channel R_{sec} that, as discussed in the Section 3.1, are simply built by instantiating the

4 It is possible to construct these resources differently. For instance, an authentic channel can be constructed from a key, an unpredictable function, and an insecure channel.

channel resource R_{chn}. Furthermore, we use the random function resource R_{rf} and the key resource R_{key} that behave as their names suggest: R_{key} always outputs a key that is generated upon the first query to it; and R_{rf} returns a new random value for each new message query. Similar to the converters, the resources are generic and can be instantiated with respect to the cryptographic schemes used in the construction. In our case, both R_{rf} and R_{key} draw values according to a uniform distribution.

In the first step of the proof, we use the notion of trace equivalence (Def. 5) to prove that the encryption scheme, instantiated with one-time pad, securely constructs a secure channel from a key and an authentic channel. That is, we show that a distinguisher, subject to the side conditions of our security definition in the Section 5.2, cannot distinguish between resources R_{r1} and R_{i1}, the areas with dashed lines in the Figure 5.5. R_{r1} is the resource resulting from attaching C_{enc} and C_{dec} to R_{aut} and R_{key}; and R_{i1} is the resource resulting from attaching C_{sm1} to R_{sec}. For the one-time-pad encryption, the simulator C_{sm1} generates random bit-strings of a given length obtained from querying the secure channel R_{sec}.

Interestingly, the necessity of trace equivalence over bisimulation can be seen even in a simple security proof that involves a one-time-pad. Consider the case where a distinguisher is attached to the open interfaces of R_{i1} and R_{r1} and where the distinguisher submits a message m to the send interface. The outputs that the distinguisher expects to receive for his subsequent queries can be modeled using random variables that depend on R_{i1}'s and R_{r1}'s internal states after the send query. In the distinguisher's view, the channel inside R_{i1}, i.e. R_{sec}, can only contain m. However, the channel inside R_{r1}, i.e R_{aut}, contains every bit-string of length $\|m\|$ with the same probability since C_{enc} uses a one-time-pad. Therefore, to prove the perfect indistinguishability of R_{i1} and R_{r1} with a local argument, one must correlate a single-point state distribution with a uniform distribution over a set of states. Standard bisimulation relations are not suitable for this purpose since they always relate individual states. Our proof rule for trace equivalence handles such a case because it demands a relation between *distributions* of states.

In the next step, we prove the security of our simple MAC scheme by showing that it securely constructs an authentic channel from a random function and an insecure channel. This construction is presented using dark gray areas in Figure 5.5. Let R_{r2} denote the resource resulting from attaching C_{mac} and C_{chk} to R_{isc} and R_{rf}, and let R_{i2} be the resource resulting from attaching the simulator C_{sm2} to R_{aut}. We prove the indistinguishability of R_{r2} and R_{i2} in three intermediate steps. (i) We show the trace equivalence of R_{r2} with its lazy variant. The main difference between the standard (eager) and the lazy systems is the way in which they treat MACs: in a lazy-R_{r2}, messages are inserted into the channel without generating a MAC and the receiver does not check MACs by default; however, in a scenario where an adversary looks into the channel's content, a MAC is generated on the fly and the receiver is triggered to check its correctness later. This preparatory step aligns the samplings in the real system with those in the ideal system with the simulator, where the simulator generates a fake MAC on the fly. (ii) We define a restricted variant of lazy-R_{r2} with a special random function that avoids generating the authentication codes queried by the adversary, and we show that any distinguisher has negligible advantage for distinguishing the restricted and non-restricted instances of lazy-R_{r2}. Here, the requirement for the notion of indistinguishability with negligible advantage becomes clear: neither the bisimulation nor the trace equivalence notions are sufficient for this step since the two systems are actually not equal. In the final intermediate step, (iii) we show that the restricted-lazy-R_{r2} and R_{i2} are trace equivalent. Hence, using the triangular inequality, we can combine the three steps and show that R_{r2} and R_{i2} are indistinguishable.

In the final proof step, we just instantiate our composition theorem, i.e. Theorem 5 with the two aforementioned constructions. Figure 5.5 depicts how the composition theorem combines the two constructions.

6 ABSTRACT COMMUNICATION MODELING

In this chapter, we explain how the properties of individual resources can be generalized to work on specifications, i.e. many resources that share a common behavior. We consider the impact of rigid communication modeling as an instance that requires such a generalization. We then propose Fused Resource Templates (FRT) as a semantic domain for formalizing CC specifications that is tailored for abstracting over the communication patterns in security proofs[1].

We start with motivating examples and explain why security statements should not be tied to fixed communication patters in Section 6.1. We then formalize FRTs and their security properties in Sections 6.2 and 6.3. Finally, in Section 6.4, we use FRTs to formalize the construction of a secure channel using one-time-pad and Diffie-Hellman key exchange.

To simplify our presentation, we refer to interfaces with their names and omit typing details (cf. Sec. 3.1.2) in this chapter. Interested readers should refer to the formalization source [38], where such details are explicit.

6.1 MOTIVATION

It is common to simplify the presentation of security proofs by omitting details that are considered somehow trivial; however, leaving out such details sometimes does more harm than good. Consider the construction example in Section 2.3. There, we limit the extent of our proofs' reusability by defining the specifications $\mathcal{R}_{auth-key}$ and $\mathcal{R}_{Sec}$ as singleton sets. That is, the security of the protocol π, representing the *Enc* and *Dec* converters, is only guaranteed in contexts that use the concrete resources *Auth*, *Key*, and *Sec*. But is it possible to ignore this limitation and interpret the aforementioned resources as placeholders for arbitrary keys and channels? Such simplifications are omnipresent in the existing CC case studies: proofs are carried out in a concrete (and usually underspecified) context and readers are left to generalize them. In this section, we will show that singleton specifications do not suffice for composable security proofs.

1 CC formalizes specifications in their most generic form as *sets of resources*. FRTs are a particular instance of such an abstraction.

We start by considering the impact of singleton specifications on the modularity and reusability of security proofs. The point of composability is that the realization of a specification can be decomposed into small and easy-to-understand steps, which can then be combined modularly. For example, in Figure 2.1, we may combine π with various key-exchange protocols to obtain different realizations of a secure channel; however, we want to prove π's security only once. That is, we do want to abstract away from the details of the key-exchange protocol in π's proof.

This cannot be achieved if we start from a specification with just a single resource like Key, because different key exchange protocols have different activation patterns for the parties. This is how the problem with generic communication modeling, which we explained in Section 2.1, arises in CC proofs. For example, in a three-round Diffie-Hellman protocol π_{3dh}, Alice asks for Bob's half-key and sends her half-key back after receiving Bob's answer. So before Alice puts her message on $Auth$, the execution of π_{3dh} would lead to the sequence `Alice` $\rightarrow$ `Bob` $\rightarrow$ `Alice` of party activations. Hence, Key would have to already model this activation sequence so that π_{3dh} can securely construct Key and be composed with π. Clearly, the security proof for π should not have to deal with this particular sequence of activations (and all the possible failed executions of the key exchange protocol). Even worse, another key-exchange protocol may lead to a different activation sequence. For example, a two-round Diffie-Hellman key exchange π_{2dh} yields either the activation sequences `Alice` $\rightarrow$ `Bob` $\rightarrow$ `Alice` or `Bob` $\rightarrow$ `Alice` before Alice sends her message on the authenticated channel. Those activation sequences cannot be added to the single Key resource as each key exchange protocol would then have to support *all* these activation sequences. So, the security argument for the secure channel construction is entangled with a specific key exchange protocol. This hinders the modular composition of protocols.

Varying party activations is not the only problem of singleton specifications. The information that the resource Key should leak to Eve also depends on the key exchange protocol. For example, the two-phase Diffie-Hellman protocol π_{2dh} uses two authenticated channels to transport the half-keys from Alice to Bob and vice versa. Those channels expose Eve interfaces; queries on those interfaces will be rejected until a message has been entered into the channel. The simulator in π_{2dh}'s security proof must account for this behavior. Yet this is not possible when Eve's interface of the ideal resource Key leaks no information to the simulator. What must be leaked depends on the particular key exchange protocol, e.g. adapting the key specification's leakage to π_{2dh} becomes problematic in π_{3dh}'s security proof that utilizes three authenticated channels.

These two problems can be avoided if specifications contain many resources. For example, the specification for keys would contain one resource for each type of key-exchange protocols: one that is suitable for constructions with two authenticated channels, another one for constructions with three authenticated channels, and so forth. However, the security proofs must treat all resources in such a specification uniformly; otherwise we would again need a separate proof for each protocol. So, we need a semantic domain that defines each specification by describing the common aspects of the resources that it contains.

6.2 FUSED RESOURCE TEMPLATES

We define FRTs in terms of resources. Each FRT describes a parametrized resource: all instances share a common behavior while each instance exhibits a particular party-activation pattern.

6.2.1 *Fused Resources*

FRT instances correspond to particular members of a resource family, but how do we know if an instance belongs to a family? Remember that family members share a common behavior while each of them will exhibit a particular behavior too. FRTs only focus on the common behaviors and treat the particular ones abstractly as a parameter in their description. Nevertheless, we must ensure that every instantiation of such parameter will not affect the "common behavior". For example, consider the specification for secure channels. The common behavior of every resource in the family of secure channels is that they would *at most* leak the length of the transmitted messages. So, the FRT describing this family must prohibit any instantiation that can leak more information.

Each FRT consists of a *Core* part, describing the common behavior, and a set of *Rest* parts. When an FRT is instantiated, the core part is *Fused* with one of the rest parts and results in a special kind of resource, which we call a *Fused Resource*, that enforces one-way information flow from its rest to core. In what follows, we explain how a fused resource can be defined in terms of a *Fusing* function.

The core part is implemented as a record $core \equiv (\!| cinit := \cdots, cpoke := \cdots, cfunc := \cdots |\!)$ with three fields: an initial core state $cinit : \mathring{s}$, a probabilistic event handler *cpoke* : $\mathring{s} \Rightarrow \mathring{e} \Rightarrow \mathbb{D}(\mathring{s})$, and a probabilistic transition function *cfunc* : $\mathring{s} \Rightarrow \mathring{q}_c \Rightarrow \mathbb{D}(\mathring{r}_c \times \mathring{s})$. The *cpoke* field is an event handler that defines a notification mechanism: given the current core state and an event, it defines a probability distribution on the successor state. The *cfunc* field describes the I/O behavior of the interfaces: given the current core state and an input, it defines a probability distribution on the pair of the output and the successor state.

The rest part communicates with the core ones via poke events. They are defined in terms of a record $rest \equiv (\!| rinit := \cdots, rfunc := \cdots |\!)$ that consists of an initial rest state *rinit* : $\mathring{t}$ and a probabilistic event-augmented transition function *rfunc* : $\mathring{t} \Rightarrow \mathring{q}_r \Rightarrow \mathbb{D}((\mathring{r}_r \times \mathbb{L}(\mathring{e})) \times \mathring{t})$, where $\mathbb{L}(\mathring{e})$ denotes the type of events list. The *cfunc* field defines the I/O behavior of rest interfaces as well as the information they leak to the core part: given the current rest state and an input, it defines a probability distribution on the output, the successor rest state, and a list of events that the fused resource's core part will be notified about.

Given a $rest \equiv (\!| rinit := ri_{def}, rfunc := rf_{def} |\!)$ and a $core \equiv (\!| cinit := ci_{def}, cpoke := cp_{def}, cfunc := cf_{def} |\!)$ with the types defined above, the Fusing function $fuse(core, rest)$ outputs a probabilistic transition function with type $(\mathring{s} \times \mathring{t}) \Rightarrow \mathring{q}_c + \mathring{q}_r \Rightarrow \mathbb{D}((\mathring{r}_c + \mathring{r}_r) \times (\mathring{s} \times \mathring{t}))$ that enforces a one-way information flow from *rest* to *core* by means of events. Consider the value constructors *Left* _ and *Right* _ for the disjoint union + type, and let tr_{fuse} and (s, t) denote the resulting oracle and its internal state respectively. For a given input x, the output of $tr_{fuse}((s, t), x)$ is a probability distribution on the pair of the output y and the successor state (s', t') that is determined in one of two ways. First, if x is of the form *Left* x', then y and s' are determined by $cf_{def}(s, x')$ and $t' = t$. Second, if x is of the form *Right* x', then $rf_{def}(t, x')$ determines y, t', and a list of events es from which we calculate $s' = foldl(cp_{def}, s, es)$.

The Fusing function provides a failure mechanism too. Every probability distribution in this thesis considers a sample space that provides a distinguished bottom element $\perp$ for modeling failures. By combining the probability and option monads, we define the *fuse* function such that any failures in calls to cp_{def}, cf_{def}, and rf_{def} will fail the whole fused resource, i.e., all subsequent queries to tr_{fuse} are answered with $\perp$ independently of which interface they are sent to.

We can combine all the aforementioned building blocks into a one-liner using the technique explained in Section 3.1. Let $\bullet$ denote the operator for accessing record fields. For core and rest records *core* and *rest* mentioned above, $FUSE(core, rest) \equiv RES(fuse(core, rest), (core \bullet cinit, rest \bullet rinit))$ is a fused resource of type $\mathbb{R}(\mathring{q}_c + \mathring{q}_r, \mathring{r}_c + \mathring{r}_r)$. We denote the type of such fused resource as $\mathbb{F}(\mathring{q}_c, \mathring{q}_r, \mathring{r}_c, \mathring{r}_r)$ to distinguish it from typical resources that have the same type.

Core records suffice to describe the behavior that is common to every instantiation of an FRT. Let $\boxed{c}$ and $\boxed{r}$ respectively denote the types of *core* and *rest* records mentioned above. Given a core $core : \boxed{c}$, an FRT $\{\!|\, core \,|\!\} : \boxed{r} \Rightarrow \mathbb{F}(\mathring{q}_c, \mathring{q}_r, \mathring{r}_c, \mathring{r}_r)$ is a function from rest records to fused resources such that:

$$\{\!|\, core \,|\!\}\,(rest) \equiv FUSE(core, rest). \tag{6.1}$$

When essential, we specify the type of suitable rest records for an FRT using a subscript $\{\!|\, core \,|\!\}_{\boxed{r}}$.

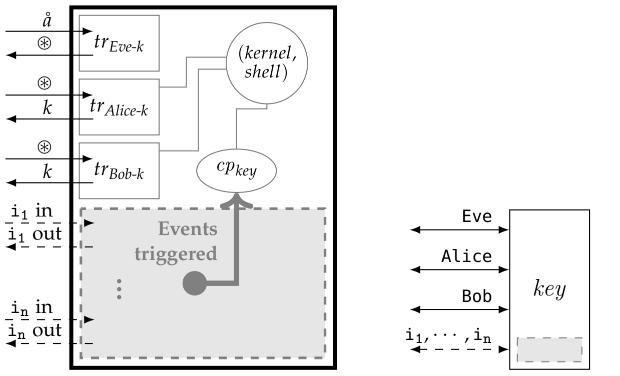
6.2.2 *Key FRT Example*

We now explain how FRTs enable the modeling of specifications that include resources with arbitrary interfaces and activation patterns. The main idea is to define core such that it returns $\perp$ when an unwanted sequence of poke events has been received.

As an example, we formalize the ideal specification for keys as the FRT $\{\!|\, key \,|\!\}$. It represents all realizations in which two parties Alice and Bob execute a protocol to share a symmetric key. We define the fields of the core record $key \equiv (\!|cinit := ci_{key}, cpoke := cp_{key}, cfunc := cf_{key}|\!)$ in the given order. The state consists of a pair $(kernel, shell)$. Here, $kernel$ is a value of the form *Void* or *Hold k* that indicates whether the key has already been generated; and $shell \subseteq \{\texttt{Alice}, \texttt{Bob}\}$ keeps track of the enabled interfaces, where "disabled" means the interface would return $\perp$ if queried. The initial state ci_{key} is $(Void, \{\})$, which indicates that the key has not been generated yet and both parties' interfaces are disabled.

The probabilistic event handler cp_{key} keeps track of two classes of events: the event *Init* updates the state to *Hold k* by uniformly sampling *k* from the key's domain and the events of the form *Open party* will update the state by inserting *party* in *shell*. Repeated events are not allowed and immediately invoke the failure mechanism.

The probabilistic transition function $cf_{key} \equiv tr_{Eve\text{-}k} +_{\mathrm{O}} tr_{Alice\text{-}k} +_{\mathrm{O}} tr_{Bob\text{-}k}$ describes the interfaces for Alice, Bob, and the adversary, also called Eve, that are composed using the oracle composition operator. The adversary interface's functionality $tr_{Eve\text{-}k}$ leaks no information to Eve; it is a dummy oracle that does not change the state and answers every $\mathring{a}$ input with a unit



(A) Internal construction (B) Abstract representation

FIGURE 6.1: Formalizing the ideal specification for keys. The rest part is shown as a gray area that accepts various instantiations. rest interfaces are shown using dashed arrows. To simplify the diagrams, we may merge multiple interfaces into a single arrow like $\mathtt{i}_1, \cdots, \mathtt{i}_\mathtt{n}$ above.

value $\circledast$. The interface functionality for Alice is defined using $tr_{Alice\text{-}k}$ which outputs a key k upon a unit input $\circledast$ if the state is a pair $(Hold\ k, shell)$ such that `Alice` $\in$ *shell*. Queries to $tr_{Alice\text{-}k}$ will invoke the failure mechanism if the state does not satisfy the aforementioned conditions. Bob's interface functionality $tr_{Bob\text{-}k}$ is the same as Alice's interface except that it checks for `Bob` $\in$ *shell* upon receiving queries.

Figure 6.1 is a pictorial representation of the ideal specification for keys. $\{\!|key|\!\}$ represents all fused resources that accept queries on `Alice` and `Bob` interfaces after one or more queries to the $\mathtt{i}_1, \cdots, \mathtt{i}_\mathtt{n}$ interfaces, which trigger the key generation event and enable (at least) one of the parties' interfaces. Either Alice or Bob may attempt to receive the key first; it is only important that their corresponding interface has been enabled during one of the event triggers.

6.2.3 *Constructions Using FRTs*

Defining complex systems in terms of simpler ones is the essence of composable security statements. CC captures this using the concepts of simultaneously-available resources and converter attachments, which are lifted to specifications as explained in Section 2.2.2. That is, compound resources describing complex probabilistic systems can be defined as the attachment of simpler building blocks, i.e., converters and simpler resources. We show how these concepts are expressed using FRTs.

The aforementioned operators do not preserve the structure of fused resources. For instance, given the fused resource $F : \mathbb{F}(\mathring{q}_c, \mathring{q}_r, \mathring{r}_c, \mathring{r}_r)$ and the converter $C : \mathbb{C}(\mathring{q}_r + \mathring{q}_c, \mathring{r}_r + \mathring{r}_c, \mathring{q}_c + \mathring{q}_r, \mathring{r}_c + \mathring{r}_r)$ that swaps interface positions, then $C \rhd F$ may be inexpressible in terms of $FUSE$ due to its inverted information flow. We now show typical cases that do preserve the structure of fused resources, and thus of FRTs. As we explain in Section 6.3.1, these cases suffice for the step in the security proof that allows us to abstract over the communication behavior and focus on the common properties. In the following four lemmas, consider the core records $cr_i : \boxed{c}_i$, the rest records $rs_i : \boxed{r}_i$, and the fused resources $F_i \equiv FUSE(cr_i, rs_i)$ for $i \in \{1, 2\}$. The first lemma explains the simultaneous access to fused resources.

Lemma 1. *For $F_i : \mathbb{F}(\mathring{q}^i_c, \mathring{q}^i_r, \mathring{r}^i_c, \mathring{r}^i_r)$, consider the fused resource $F : \mathbb{F}(\mathring{q}^1_c + \mathring{q}^2_c, \mathring{q}^1_r + \mathring{q}^2_r, \mathring{r}^1_c + \mathring{r}^2_c, \mathring{r}^1_r + \mathring{r}^2_r)$ defined as $F \equiv FUSE(cr_1 \|_c cr_2, rs_1 \|_r rs_2)$, where $\|_c$ and $\|_r$ stand for the parallel composition of core and rest records respectively, analogous to $+_{\mathbb{O}}$ for oracles. Then $F = C_{F-PL} \rhd (F_1 \,||\, F_2)$, where C_{F-PL} is the wiring that groups the core and rest interfaces of F_1 and F_2 together.*

As a corollary of the following lemma, converters may attach to just the core (or similarly the rest) interfaces of a fused resource since either of the parallel converters could be replaced with the identity converter $C_{\mathbb{1}}$.

Lemma 2. *For $F_1 : \mathbb{F}(\mathring{q}_c, \mathring{q}_r, \mathring{r}_c, \mathring{r}_r)$ and $F_2 : \mathbb{F}(\mathring{x}_c, \mathring{x}_r, \mathring{y}_c, \mathring{y}_r)$, consider the converters $C_c : \mathbb{C}(\mathring{x}_c, \mathring{y}_c, \mathring{q}_c, \mathring{r}_c)$ and $C_r : \mathbb{C}(\mathring{x}_r, \mathring{y}_r, \mathring{q}_r, \mathring{r}_r)$. Then $F_2 = (C_c \mid C_r) \rhd F_1$ if $cr_2 = C_c \rhd_c cr_1$ and $rs_2 = C_r \rhd_r rs_1$, where $\rhd_c$ attaches a converter to the cfunc oracle of a core and $\rhd_r$ does so for the rfunc oracle of a rest part.*

The next two lemmas determine if a converter's simultaneous attachment to core and rest interfaces results in a new fused resource. Using them, one may rewrite an arbitrary converter C's attachment to F_2, i.e., $C \rhd F_2$, into $(C \odot \_) \rhd F_1$, where _ is filled with some wiring, and then (if possible) apply the previous lemma. These lemmas stem from the one-way information flow from rest to core interfaces of fused resources.

Lemma 3. *For $F_1 : \mathbb{F}(\mathring{x} + \mathring{q}_c, \mathring{q}_r, \mathring{y} + \mathring{r}_c, \mathring{r}_r)$, let `I` refer to the left-most (w.l.o.g.) core interface. Assume that `I`'s $\mathring{y}$ responses do not depend on $\mathring{q}_c$ queries, and let cr_2 and rs_2 be the records that result from removing `I`'s functionality from cr_1 and adding it to rs_1. Then $(F_2 : \mathbb{F}(\mathring{q}_c, \mathring{x} + \mathring{q}_r, \mathring{r}_c, \mathring{y} + \mathring{r}_r)) = C_{F-CR} \rhd F_1$, where C_{F-CR} is the wiring converter that reorders the interface positions accordingly.*

Lemma 4. *For $F_1 : \mathbb{F}(\mathring{q}_c, \mathring{x} + \mathring{q}_r, \mathring{r}_c, \mathring{y} + \mathring{r}_r)$, let* `I` *refer to the left-most (w.l.o.g.) rest interface. Assume that $\mathring{r}_r$ responses do not depend on* `I`*'s $\mathring{x}$ queries, and let rs_2 and cr_2 be the records that result from removing* `I`*'s functionality from rs_1 and adding it to cr_1. Then $(F_2 : \mathbb{F}(\mathring{x} + \mathring{q}_c, \mathring{q}_r, \mathring{y} + \mathring{r}_c, \mathring{r}_r)) = C_{F-RC} \rhd F_1$, where C_{F-RC} is the wiring converter that reorders the interface positions accordingly.*

While these lemmas refer to individual fused resources, they naturally lift pointwise to FRTs. So these cases also preserve the FRT structure. We write $C\{\!|\, cr \,|\!\}_{\boxed{r}}$ for the FRT that consists of the fused resources $C \rhd \{\!|\, cr \,|\!\}(rs)$ for $rs \in \boxed{r}$. This notation is analogous to Section 2.2.2 where $\pi\mathcal{R}$ represented the attachment of the protocol π to the specification $\mathcal{R}$.

For Lemma 1, the resulting FRT $\{\!|\, cr_1 \|_c cr_2 \,|\!\}_{\boxed{r}_1 \times \boxed{r}_2}$ is indexed over $\boxed{r}_1 \times \boxed{r}_2 = \{rs_1 \|_r rs_2 \mid rs_1 \in \boxed{r}_1 \wedge rs_2 \in \boxed{r}_2\}$. This set does not include all rest records that could be fused to the parallel composition of the core records cr_1 and cr_2. This is because the internal state of rest records in $\boxed{r}_1 \times \boxed{r}_2$ can be split into two independent components, one for each of the fixed rest records rs_1 and rs_2; but such a decomposition is not possible in general.

6.3 SECURE CONSTRUCTIONS WITH FRTS

As explained in Section 2.2.2, simulation-based security for specifications $\mathcal{R}$ and $\mathcal{S}$ and a protocol π demands that there is a simulator $\sigma = z^{\mathtt{E}}$, attaching a converter z to the adversary interface `E` (short form of `Eve`), such that

$$\forall R \in \mathcal{R}.\ \exists S \in \mathcal{S}.\ \mathfrak{d}(\pi R, \sigma S) \leq \epsilon. \tag{6.2}$$

Note that the same simulator σ must work for all $R \in \mathcal{R}$. Accordingly, a formal proof of simulation-based security must also work for all $R \in \mathcal{R}$. To that end, $\mathcal{R}$ must be expressed in such a way that the properties relevant for the security proof are explicit and the irrelevant details can be abstracted away. In this section, we show that FRTs are well-suited for composable simulation-based security: the core record captures the relevant properties and the rest record hides the details of the party activation patterns.

First, we specialize (6.2) to the case where $\mathcal{R}$ and $\mathcal{S}$ are FRTs. We only define the information-theoretic notion of security. Computational security is formalized analogously by introducing a bound on distinguisher's queries. The extension to asymptotic security is similar to Section 5.2.

Definition 8 (Information-theoretic concrete security). *Let $\{\!|\, real \,|\!\}_{\boxed{r}_R}$ and $\{\!|\, ideal \,|\!\}_{\boxed{r}_I}$ be FRTs and let C_π be a protocol*[2]*, i.e., a converter that attaches to*

2 To keep our presentation uniform, we use $C_\pi, C_\sigma, \cdots$ instead of $\pi, \sigma, \cdots$ that are common in CC literature (cf. Sec. 2.2.2).

the core user interface of $\{\!|\mathit{real}|\!\}$*. Then* $C_\pi\{\!|\mathit{real}|\!\}$ *is an information-theoretically* ϵ*-secure realization of the ideal specification* $\{\!|\mathit{ideal}|\!\}$ *if there exist a simulator* C_σ*that attaches to the adversary interface of core and rest, and a rest embedding function* $f : \boxed{r}_R \Rightarrow \boxed{r}_I$ *such that for all distinguishers D*

$$adv(D, C_\pi \rhd \{\!|\mathit{real}|\!\}(rs), C_\sigma \rhd \{\!|\mathit{ideal}|\!\}(f(rs))) \leq \epsilon \qquad (6.3)$$

for all $rs \in \boxed{r}_R$[3]*.*

Note that the rest embedding f in (6.3) skolemizes the existential $\exists S \in \mathcal{S}$ in (6.2). This works thanks to the shape of FRTs, namely both $\mathcal{R}$ and $\mathcal{S}$ are parametrized by the rest records.

6.3.1 *Proving a Protocol Secure*

Proving a protocol C_π secure boils down to establishing a bound on the advantage of a distinguisher (cf. Section 5.2). This is typically done as a sequence of game transformations using the existing tool set from the CryptHOL extension that we have presented in the previous chapters. We now explain the common formalization pattern for FRTs.

The proof of (6.3) must work for all $rs \in \boxed{r}_R$. It should therefore concentrate on the cores *real* and *ideal* of the fused resources $\{\!|\mathit{real}|\!\}(rs)$ and $\{\!|\mathit{ideal}|\!\}(f(rs))$. The first step is therefore to pretend that the fusing is part of the distinguisher. For comparison, this step corresponds to applying the dummy adversary lemma in UC.

On the real side $C_\pi \rhd \{\!|\mathit{real}(rs)|\!\}$, Lemma 2 shows that the converter C_π can be attached directly to *real* rather than the fused resource, because C_π attaches only to the core interface. This transformed core *real′* yields a transformed FRT $\{\!|\mathit{real}'|\!\}_{\boxed{r}_R}$.

On the ideal side, however, we cannot directly use Lemma 2 to push the simulator C_σ through *fuse* because C_σ attaches to the adversary core and rest interfaces. This way, the simulator can create a flow of information from the core part to the rest part, which cannot happen in a fused resource. We now exploit that the simulator must work for all rs in the same way, in particular for the rest record that does not provide any interfaces at all. Accordingly, C_σ uses only to the interfaces that f adds on top of rs's. By a similar argument, $f(rs)$ answers queries on these additional interfaces

3 Note that $\mathfrak{d}(R, S) \leq \epsilon$ iff $adv(D, R, S) \leq \epsilon$ for all distinguishers D. In the information-theoretic setting, D ranges over all distinguisher; while in the computational setting, it ranges over computationally bounded ones.

without consulting rs itself. Moreover, f may translate poke events from rs to poke events for the ideal core and this translation is independent of rs's state.[4] So we can move these additional interfaces to *ideal* by Lemma 4. Then, C_σ only attaches to core interfaces and can thus be integrated into the core by Lemma 2. This new core *ideal*′ yields a new FRT $\{\!|\mathit{ideal}'|\!\}_{\boxed{r}_R}$.

In summary, it suffices to prove

$$adv(D, \{\!|\mathit{real}'|\!\}(rs), \{\!|\mathit{ideal}'|\!\}(rs)) \leq \epsilon$$

for all D. Now, the fused resources $\{\!|\mathit{real}'|\!\}(rs)$ and $\{\!|\mathit{ideal}'|\!\}(rs)$ use the same rest record rs. Since the core and rest in a fused resource operate on separate states, we can understand a core record as a resource of its own, with the poke events as an additional interface that returns only dummy responses $\circledast$. Accordingly, we have

$$adv(D, \{\!|\mathit{real}'|\!\}(rs), \{\!|\mathit{ideal}'|\!\}(rs)) = adv(D', \mathit{real}', \mathit{ideal}')$$

for some D'. This key proof step abstracts over the party activation patterns: from now on, the security proof can focus on the cores *real*′ and *ideal*′. Since these are like resources, this proof can be approached as described in Chapter 5.

6.3.2 *Composabile Security with FRTs*

We now show that our security definition 8 yields composable security statements. In CC, a protocol C_π typically uses multiple resources $R_1, \ldots, R_n$ to create a resource $C_\pi \rhd (R_1 \| \ldots \| R_n)$, which should be hard to distinguish from a simulated ideal resource $C_\sigma \rhd S$. Similarly, another protocol ρ may use S and possibly other resources to achieve another ideal resource T. To abstract over the party activation patterns of these resources, we consider FRTs rather than individual resources. This ensures that the security proof for ρ is independent of the patterns that we need for R_i during the composition.

4 This argument can be made precise in the computational setting: The skolemization f in (6.3) of the existential $\exists S \in \mathcal{S}$ in (6.2) ensures that f chooses the rest records in $\{\!|\mathit{ideal}|\!\}$ uniformly. Formally, this translates to f being relationally parametric in the queries and responses of rs. So we can decompose $f(rs)$ into two parts with disjoint state: (i) rs and (ii) additional independent interfaces and a translator for poke events coming from rs.
In the information-theoretic setting, this uniformity argument does not apply and the argument therefore need not hold in pathological cases. We have not yet encountered such a case in practice though.

Clearly, if $C_\pi \{\!|real|\!\}_{\boxed{r}_R}$ ϵ-securely realizes the FRT $\{\!|ideal|\!\}_{\boxed{r}_I}$ (Def. 8), then $C_\pi \rhd \{\!|real|\!\}(rs)$ ϵ-securely realizes the fused resource $\{\!|ideal|\!\}(f(rs))$ in the sense of Def. 7 for all $rs \in \boxed{r}_R$ as witnessed by the simulator C_σ.

These composability results suffice to show that secure realization between FRTs is also compositional. As a FRT $\{\!|cr|\!\}_{\boxed{r}}$ is a family of fused resources $\{\!|cr|\!\}(rs)$ for $rs \in \boxed{r}$, we obtain composability by lifting Thm. 8 pointwise to specifications. Note that the simulator and the rest embeddings remain independent of the chosen fused resources, as required by Def. 8.

Theorem 8 (Composability for FRTs).

1. *The identity protocol $C_{\mathbb{1}}$ applied to a FRT $\{\!|cr|\!\}_{\boxed{r}}$ 0-securely realizes the FRT $\{\!|cr|\!\}_{\boxed{r}'}$ for $\boxed{r}' \supseteq \boxed{r}$ with the simulator being the identity converter and the rest embedding being the identity function.*

2. *Let $C_{\pi_i} \{\!|real_i|\!\}_{\boxed{r}_{R_i}}$ ϵ_i-securely realizes the FRT $\{\!|ideal_i|\!\}_{\boxed{r}_{I_i}}$ with simulator C_{σ_i} and rest embedding f_i for $i = 1, 2$. The parallel composition*

$$(C_{\pi_1} \mid C_{\pi_2}) \{\!|real_1 \|_c real_2|\!\}_{\boxed{r}_{R_1} \times \boxed{r}_{R_2}}$$

   $(\epsilon_1 + \epsilon_2)$-securely realizes

$$\{\!|ideal_1 \|_c ideal_2|\!\}_{\boxed{r}_{I_1} \times \boxed{r}_{I_2}}$$

   with $C_{\sigma_1} | C_{\sigma_2}$ as the simulator and $f(rs_1, rs_2) = (f_1(rs_1), f_2(rs_2))$ as the rest embedding.

3. *Let $C_\pi \{\!|real|\!\}_{\boxed{r}_R}$ ϵ-securely realize the FRT $\{\!|middle|\!\}_{\boxed{r}_M}$ with simulator C_σ and rest embedding f and let $\rho \{\!|middle|\!\}_{\boxed{r}'_M}$ ϵ'-securely realize $\{\!|ideal|\!\}_{\boxed{r}_I}$ with simulator C_τ and rest embedding g such that $\boxed{r}_M \subseteq \boxed{r}'_M$. Then, $C_\pi \{\!|real|\!\}_{\boxed{r}_R}$ $(\epsilon + \epsilon')$-securely realizes $\{\!|ideal|\!\}_{\boxed{r}_I}$ with simulator $C_\tau \odot C_\sigma$ and rest embedding $g \circ f$.*

4. *If $C_\pi \{\!|real|\!\}_{\boxed{r}_R}$ ϵ-securely realizes the FRT $\{\!|ideal|\!\}_{\boxed{r}_I}$ and C_u is a converter that attaches to the core user interface, then $C_u C_\pi \{\!|real|\!\}_{\boxed{r}_R}$ ϵ-securely realize $C_u \{\!|ideal|\!\}_{\boxed{r}_I}$ with the same simulator and rest embedding*[5].

5 As explained in Section 2.2.2, $C_u C_\pi$ represents the attachment of converters to disjoint interfaces. We formalize such an attachment using wiring converters (cf. Sec. 3.3) and converters' composition operators (cf. Sec. 3.2).

6.4 A RUNNING EXAMPLE: DDH EXTENSION

We now use our framework to formalize the construction of a secure channel using three authenticated channels. For brevity, we only provide a high-level overview and use diagrams to present the main steps of our formalization [38].

We apply the following conventions in diagrams. fused resources are depicted on the right; we use subscripted R as short names for them, e.g. R_{aut1} corresponds to the FRT $\{\!|aut1|\!\}$ that is instantiated with the rest record rs_1 in Figure 6.2. Converters' attachment to fused resources is represented using arrows, where solid and dashed arrows represent the attachment to core and rest interfaces respectively. Such an attachment results in new fused resources that are presented using rounded rectangles with different colors or borders. fused resources or converters that are in the same column are composed in parallel using $\|$ or $|$ respectively. The left-most interfaces of every diagram belong to Eve, Alice, Bob, and "Rest" from top to bottom. For simplicity, we merge multiple rest or adversary interfaces into a single arrow and mark their meeting point using a black circle. The interfaces are preserved among all the diagrams, so we name them only in some of the subfigures.

Our example construction combines two cryptographic primitives in Figure 6.2a. First, Diffie-Hellman key exchange protocol with the converters C_{dhA} and C_{dhB} is used to construct a key R_{key} from two of the authenticated channels, i.e., R_{aut1} and R_{aut2} that are in the opposite directions. Second, we use one-time-pad in the converters C_{enc} and C_{dec} to construct a secure channel R_{sec} from the key R_{r1}, which is the result of Diffie-Hellman key exchange, and the last authenticated channel R_{aut3}.

The aforementioned components behave as their name suggests. The key and channel resources are defined as we have explained in Section 6.2.2. The C_{enc} and C_{dec} converters are stateless and each provide a single external interface for Alice and Bob respectively. When the C_{enc} converter is queried with a message m, it fetches a key k via its internal interface that is attached to C_{dhA} and forwards $m \oplus k$, where $\oplus$ is the xor operator for bit-strings, to its internal interface that is connected to the authenticated channel R_{aut3}. The C_{dec} converter works analogously to decrypt the cipher-text that it fetches from R_{aut3}.

The C_{dhA} and C_{dhB} are more involved. We describe C_{dhA} as a representative of these converters since they are like duals. C_{dhA} is a stateful converter with two external interfaces. The first external interface, which is placed at the top and redirected to R_{r1}'s rest part, stores a half-key x in its state upon receiving a unit query $\circledast$ and puts g^x into the authenticated channel R_{aut1}, where g is the cyclic group's generator. The second external interface, which is attached to C_{dec}, can only get queried after the query to the first interface; otherwise the failure mechanism is triggered. It fetches the half-key g^y, which is sent by the other party, from the authenticated channel R_{aut2} and stores $(g^y)^x$ in its state.

Figure 6.2 depicts the central steps of the security proof. First, corresponding to steps from Figure 6.2a to 6.2e, we prove that the specification R_{r1} securely realizes the key specification R_{key}. Second, corresponding to the dotted area in Figures 6.2e and 6.2f, we show that the specification R_{r2}, i.e., the fused resource that is presented using dashed borders and results from attaching C_{enc} and C_{dec} converters to R_{key} and R_{aut3}, securely realizes the secure channel specification R_{sec}. Finally, justifying the steps from Figure 6.2a to 6.2f, the two constructions are composed using Theorem 8.

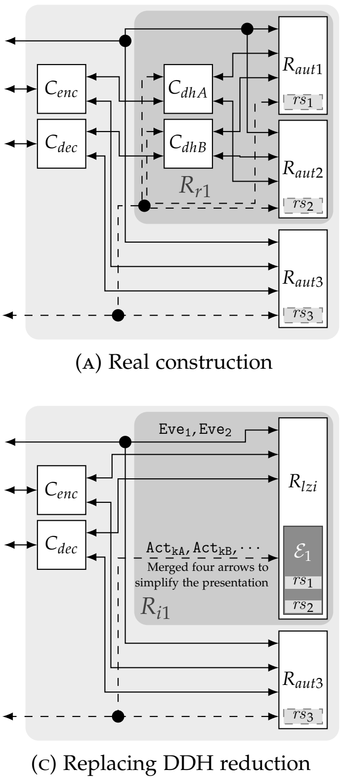
The secure realization of R_{key} using R_{r1} requires a proof with three major steps. First, going from Figure 6.2a to Figure 6.2b, we make our key-exchange protocol lazy. That is, we prove that R_{r1} is trace equivalent to a fused resource R_{lzr} that postpones the half-key samplings until one of the parties requires the key. Note that the aforementioned fused resources provide the same interfaces to the outside world. Let's call C_{dhA} and C_{dhB}'s top interface $\texttt{Act}_{\texttt{kA}}$ and $\texttt{Act}_{\texttt{kB}}$ respectively; furthermore, let's call R_{aut1} and R_{aut2}'s adversary interfaces $\texttt{Eve}_1$ and $\texttt{Eve}_2$. In the fused resource R_{r1}, Diffie-Hellman half-keys x and y are sampled separately upon the queries to $\texttt{Act}_{\texttt{key-a}}$ and $\texttt{Act}_{\texttt{key-b}}$ respectively. However, R_{lzr} samples the pair (x, y) when it receives a query on any of $\texttt{Act}_{\texttt{kA}}$, $\texttt{Act}_{\texttt{kB}}$, $\texttt{Eve}_1$, or $\texttt{Eve}_2$. This proof step utilizes Lemma 2 to reason about the attachment of Diffie-Hellman converters to the authenticated channels and Lemma 3 to move the protocol initiation queries, i.e., $\texttt{Act}_{\texttt{kA}}$ and $\texttt{Act}_{\texttt{kB}}$, to R_{lzr}'s rest part.

Second, corresponding to the backwards steps from Figure 6.2e to Figure 6.2c, we apply a similar procedure to make the ideal specification R_{key} lazy. That is, R_{i1} is trace equivalent to a FRT R_{lzi} that samples the outputs of the adversary and user interfaces at the same time. Therefore, instead of sampling x, y, and z separately upon the queries to $\texttt{Act}_{\texttt{kA}}$, $\texttt{Act}_{\texttt{kB}}$, and either of $\texttt{Eve}_1$ or $\texttt{Eve}_2$ respectively, R_{lzi} samples the triple (x, y, z) whenever any of the aforementioned interfaces are queried. Furthermore, R_{lzi} resembles the combined behavior of C_{sm1} and $\mathcal{E}_{\mathrm{dh}}$ using the rest embedding $\mathcal{E}1$: it keeps track of $\texttt{Eve}_1$ and $\texttt{Eve}_2$'s forward queries and the events received on the rest interfaces, e.g. $\texttt{Act}_{\texttt{kA}}$ and $\texttt{Act}_{\texttt{kB}}$, to trigger appropriate events on its core part. The proof step from Figure 6.2e to Figure 6.2d uses a corollary of Lemma 4 to move $\mathcal{E}_{\mathrm{dh}}$ to R_{mvk}'s core part; and the proof step to Figure 6.2c uses Lemma 2 to reason about the attachment of the resulting specification to C_{sm1}. Note that C_{sm1} is not connected to R_{key}'s core adversary interface that answers all queries with $\circledast$. We use an unconnected arrow to signify the fact that C_{sm1}'s behavior does not depend on the input on that interface.

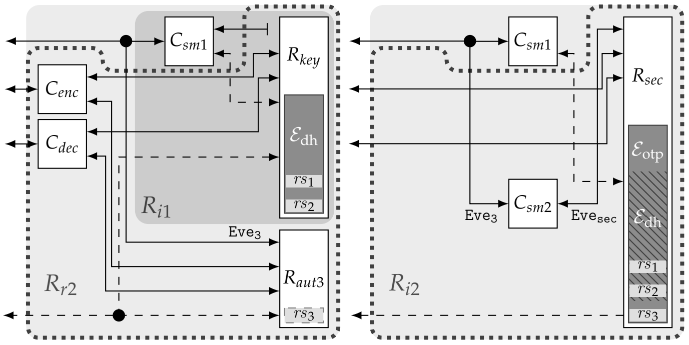
Third, corresponding to the step between Figures 6.2b and 6.2c, we prove an arbitrary distinguisher D's advantage in distinguishing R_{lzr} and R_{lzi} is bounded by D's advantage in the decisional Diffie-Hellman game, in short *ddh*. This proof step is based on a reduction in which R_{lzr} and R_{lzi} query an external oracle to receive the triples $(g^x, g^y, g^{(x*y)})$ and (g^x, g^y, g^z) respectively, which are used to answer all the user and adversary queries.

The secure realization of R_{sec} using R_{r2} requires less effort. This corresponds to the step from Figure 6.2e to Figure 6.2f. We prove that R_{r2} is trace equivalent to the fused resource resulting from attaching C_{sm2} to R_{sec}. The simulator C_{sm2} does not need to communicate with R_{sec}'s rest part. Upon receiving a look query on $\mathtt{Eve}_3$, it queries $\mathtt{Eve}_{\mathtt{sec}}$ with a look query to get the length of R_{sec}'s content and outputs a uniformly sampled bit string with the same length; the output is stored in C_{sm2}'s state to answer future look queries. The rest of the adversary queries to C_{sm2}, e.g. forward or drop queries, are simply forwarded to R_{sec}. The rest embedding $\mathcal{E}_{\mathrm{otp}}$ keeps track of the events that are received on R_{key} and R_{aut3}'s rest interfaces to trigger the corresponding events on R_{sec}'s core. As mentioned earlier, Theorem 8 combines this construction step with the previous ones.

In summary, we obtain the following security results. Let ϵ denote the distinguishing advantage for the decisional Diffie-Hellman game and consider the protocol $C_\pi \equiv C_{\mathbb{1}} \mid ((C_{enc} \odot C_{dhA}) \mid (C_{dec} \odot C_{dhB}))$. Then, the specification $C_\pi \{\!| (aut1 \|_c aut2) \|_c aut3 |\!\}$ ϵ-securely realizes the specification $\{\!| sec |\!\}$ with $(C_{sm1} \mid C_{sm2}) \mid C_{\mathbb{1}}$ as the simulator and $\mathcal{E}_{\mathrm{otp}}(\mathcal{E}_{\mathrm{dh}}(rs_1 \|_r rs_2) \|_r rs_3)$ as the rest embedding.



(A) Real construction
(B) Replacing lazy real key
(C) Replacing DDH reduction
(D) Replacing lazy ideal key



(E) Replacing ideal key
(F) Ideal functinality

FIGURE 6.2: Construction of a secure channel using one-time-pad and Diffie-Hellman key exchange.

7 FINAL WORDS

In this chapter, we discuss how our formalization relates to the existing approaches to composable security arguments and compare it with relevant formalization results. We also draw a conclusion on the thesis.

7.1 DISCUSSION AND RELATED WORK

CC [40] models resources as random systems [43], i.e., families of conditional probability distributions. The trace of a resource is exactly a random system. Conversely, every random system is the trace of some resource. The novel part here is the recursive definition of a trace (Def. 5), which gives rise to an unwinding rule for random systems (Thm. 1). This notion of traces has been inspired by abstract theory of modelling systems coalgebraically. This provides evidence that we have found the right coalgebraic model for CC.

Maurer and Renner [41] already noted that the theory of random systems supports only a single interface but CC needs support for multiple interfaces. Our theory of interfaces and interface types provides this extension. We can safely combine several interfaces into one and rearrange them as needed with wiring converters. The crucial insight here is that an interface type is like a dependent function type: The set of possible responses depends on the concrete input value. This allows us to express that the response will be sent on the same interface as the query.

Our work is the first that formulates requirements for system communication modeling and provides a rigorous solution as an instance of CC. FRTs introduce a formal approach to abstract over the systems communication patterns in a family of systems that exhibit similar behavior. In Section 6.1, we explained the importance of such a solution for the reusability of security proofs. The pen-and-paper presentations of existing CC results [29, 40–42] do not necessitate providing details on the communication model and, by eliding them, they must be understood as just a high-level overview of security proofs and their composition. However, in our approach, security proofs are mathematical objects and leaving out any details would invalidate them.

Comparing to the simulation-based frameworks [1, 16, 27, 35], we consider a less intricate type of system communication. Nevertheless, there are scenarios in which our approach excels. In simulation-based frameworks, the adversary Turing machine controls the communication between all protocol components. The composition theorems in these frameworks fix the corruption model but consider arbitrary adversary machines besides that. This resembles the role that the FRTs' rest part plays in our definition of secure constructions and their composition. We provide composition theorems that consider arbitrary rests; however, we only consider a one-way information flow from the rest to the core part, which corresponds to the non-adversary components in the simulation-based frameworks. But note that we allow multiple rests in our model, e.g. when two FRTs are composed in parallel. This is convenient for modeling scenarios where parties are not corrupted by the same adversary, which would be challenging in simulation-based framework since they use a central adversary machine.

The composition theorems (Thms. 4-6) show that our formalization meets the requirements for general composability [41]. This is an improvement to the state of the art where there are only few results on formally verified composable cryptographic proofs [2]. Existing formal-methods tools mainly focus on game-based proofs, which in general are not composable. There are some results about specific class of protocols' composability in such tools. For example, Blanchet [12] has formalized CryptoVerif's composition theorems to compose a key providing protocol and a protocol that uses this key and utilized them in TLS 1.3's formal verification.

There are few results on formalizing simulation-based proofs. Most of them [14, 24] focus on individual (or class of) protocols and hence they do not offer composition theorems that can be applied in arbitrary context. EasyUC [17], which is the most similar to our work, does support the composition of security proofs. It uses EasyCrypt [5] to formalize a simplified version of the Universal Composability framework with a restricted adversary machine. Such simplifications are essential since the formalization of Universal Composability in its most general form is challenging, even with a state-of-the-art formal-methods tool like EasyCrypt.

We compare EasyUC with our result in two respects. First, EasyUC's restriction on the adversary machine affects the reusability of security proofs, as explained in Section 6.1, since it leads to a fixed communication pattern among the non-adversary machines. Second, formalizing security statements in EasyUC is more difficult. This is a benefit that stems from CC's abstract approach to cryptography; we don't need to delve into concrete details that are intrinsic part of every security statement in UC-style frameworks.

To explain the later difference in more details, the EasyUC formalization that constructs a secure channel using Diffie-Hellman key exchange and a one-time-pad constitutes 18K lines of proofs and definitions, where 12K lines are devoted to composing the concrete security statement alone. However, our formalization [37, 38] required 13.6K lines of lemmas and definitions overall. The case study in Section 5.4 required 2K lines; Diffie-Hellman case study in Section 6.4 constitutes 3.5K lines; and the rest form a library of reusable resource definitions and proof rules that can be used for modelling and reasoning about other security protocols.

7.2 CONCLUSION

We have presented the semantic foundation of our framework and demonstrated that it supports the formalization of composable security statements. Furthermore, we have presented an abstract approach to communication modeling in Constructive Cryptography. We highlight the importance of systems communication patterns on the reusability of security proofs and offer a rigorous approach that allows protocol designers to abstractly capture it. We explain the limitations of existing formalized composability results that do not model systems communications to a full extent. By lifting such limitations, we support the modular formalization of a wider range of scenarios than existing formal-methods tools support. Carrying out further case studies and enhancing automation support is left as future work.

A

FURTHER BACKGROUND

We introduce some additional background that may be necessary for this thesis. First, we introduce higher-order logic and the Isabelle/HOL proof assistant (Section A.1) and review basic concepts and notations of functional programming (Section A.2). Then, in Section A.3, we review the various flavours of game-based proofs in the cryptography literature.

A.1 HIGHER-ORDER LOGIC IN ISABELLE

Higher-order logic (HOL) combines functional programming with logic [44]. This section introduces the logic part and its implementation in the proof assistant Isabelle. Functional programming aspects will be reviewed in the next section.

Terms in higher-order logic are expressed in the simply typed λ-calculus with let-polymorphism [47]. That is, a HOL type is either a type variable (presented using Greek letters $\mathring{a}, \mathring{b}, \ldots$ for type variables) or a type constructor applied to the appropriate number of types as arguments; type constructors are usually written in blackboard bold ($\mathbb{B}, \mathbb{L}, \mathbb{P}, \ldots$) or infix ($+, \times, \Rightarrow, \ldots$). HOL terms are built from variables, constants, function applications, and abstractions. The notation $e : \mathring{t}$ means that the HOL term e has type $\mathring{t}$.

HOL has a simple set-theoretic semantics [31, 32, 47], where types are interpreted as sets and terms denote elements of the set corresponding to their type. In particular, the HOL type $A \Rightarrow B$ of functions from A to B is interpreted as the full function space between A's and B's interpretation. A function need not be given a name: $\lambda x : \mathring{a}.\ t(x)$ denotes the anonymous function that maps every value v of type $\mathring{a}$ to the interpretation of t where x is replaced by v. Typically, one omits the type annotation $: \mathring{a}$ and leave type inference to deduce the most general type. Also, if x does not occur in t, we write _ instead of x, like in the constant function $\lambda\_.\ \mathit{True}$. A HOL proposition is a closed well-typed λ-term of the type $\mathbb{B}$ of the two truth values *True* and *False*.

To prove a proposition, one constructs a derivation using the HOL deduction system. Its proof rules include α-, β- and η-conversion of the λ-calculus, the standard inference rules for implication ($\longrightarrow$) and equality ($=$), and the axioms of excluded middle, of function extensionality, of choice, and of the existence of an infinite type. In short, HOL is a classical logic of total functions with the axiom of choice. Everything else, e.g., quantifiers and induction schemas, can be expressed as HOL terms and derived from these first principles. For example, the universal quantifier $\forall x.\ P(x)$ is defined similarly to Church's simple theory of types [18] by a constant[1] $\mathit{All} : (\mathring{a} \Rightarrow \mathbb{B}) \Rightarrow \mathbb{B}$ given by $\mathit{All} \equiv \lambda P.\ P = (\lambda\_.\ \mathit{True})$ and $\forall x.\ P(x)$ is syntactic sugar for $\mathit{All}\ (\lambda x.\ P(x))$. Defining equations are indicated by $\equiv$ and defined constants are written in *sans-serif*. In theorems, all variables are implicitly universally quantified.

Clearly, mechanical checks are only meaningful if the axioms and proof rules are consistent, which is the case for HOL [31, 32, 47]. Moreover, when a user adds more axioms, e.g., a theory of the real numbers, he himself has to ensure that a model still exists. To avoid the danger of inconsistency, all concepts in Isabelle/HOL are introduced definitionally. That is, one first defines a new constant in terms of existing HOL terms (for example, the universal quantifier is expressed in terms of equality of functions) and then derives the desired properties of the new constant as HOL theorems. Importantly, when sufficiently many properties have been formally derived from the construction (e.g., introduction and elimination rules for *All*), one no longer needs the internal construction for reasoning. Similarly, a new type is introduced by identifying a suitable, non-empty subset of an existing HOL type. These definition principles are conservative, i.e., by making definitions, one cannot prove more than what was possible before. This ensures the overall consistency of the formalization.

The proof assistant Isabelle/HOL is an implementation of HOL written in Standard ML. Its so-called trusted kernel implements the simply typed λ-calculus and the HOL proof rules and checks that definitions adhere to the principles mentioned above [31, 33]. Isabelle accepts a HOL term as a theorem only if the theorem has been constructed using the kernel's proof rules. This design ensures a small trusted code base: only implementation errors in the kernel (a few thousand lines of well-tested and scrutinized code) could result in erroneous proofs being accepted.

To alleviate the user from the burden of constructing proofs from primitive derivations, Isabelle/HOL comes with several *proof engines* that compile high-level proof steps down to primitive inferences. Thus, users can write

1 All function symbols in HOL are called constants as function application is primitive.

their proofs at a high level of abstraction and call the appropriate proof engine, possibly with some hints.

Isabelle/HOL comes with a library of formalized mathematics that follows these principles. For example, the rationals are constructed from a pair of integers (numerator and denominator) and the reals as Dedekind cuts of rationals. Probability theory, analysis, and many algebraic concepts have been formalized in this way.

A.2 FUNCTIONAL PROGRAMMING IN HOL

Analogous to the proof engines, Isabelle's definitional packages alleviate the user from the low-level details of the definitional principles. They take a specification of the new type or constant, internally construct the definition from existing concepts, and derive the user specification as theorems from the definition. This way, users can work in HOL like in a functional programming language with algebraic datatypes and recursive functions, which we explain in this section.

An algebraic datatype is a disjoint union of (combinations of) HOL types where the embedding into the union is made explicit using constructors. For example, the natural numbers $\mathbb{N}$ form an algebraic datatype as they are the disjoint union of the singleton set $\{0\}$ and the successors of all natural numbers $\{\, n+1 \mid n \in \mathbb{N} \,\}$, i.e., the constructors are $0 : \mathbb{N}$ and $\mathit{Suc} : \mathbb{N} \Rightarrow \mathbb{N}$. Algebraic datatypes can be polymorphic. For example, the type of pairs $\mathring{a} \times \mathring{b}$ has only one polymorphic constructor $(\_,\_) : \mathring{a} \Rightarrow \mathring{b} \Rightarrow \mathring{a} \times \mathring{b}$. Datatype values are analysed using *case* expressions. For example, $\mathit{case}\ x\ \mathit{of}\ \mathit{Suc}(y) \Rightarrow (y, \mathit{True}) \mid 0 \Rightarrow (0, \mathit{False})$ analyses $x : \mathbb{N}$ and returns the predecessor y of x and a Boolean to indicate whether x is greater than 0. In Isabelle, the command **datatype** [13] introduces algebraic datatypes according to definitional principles. It also derives an induction schema, e.g., $P(0) \longrightarrow (\forall x.\ P(x) \longrightarrow P(\mathit{Suc}(x))) \longrightarrow (\forall x.\ P(x))$ for $\mathbb{N}$.

It is common to curry functions with several arguments, for example, a function f with n arguments has type $\tau_1 \Rightarrow \tau_2 \Rightarrow \ldots \Rightarrow \tau_n \Rightarrow \tau$, where the function type constructor $\Rightarrow$ associates to the right. So f takes its arguments individually rather than as a single tuple of type $\tau_1 \times \tau_2 \times \ldots \tau_n \Rightarrow \tau$. Function application is written as usual using parentheses as in $f(t_1, t_2, \ldots, t_n)$. Currying has the advantage that functions can be applied partially: if too few arguments are supplied, the result is a specialized function that waits for the missing arguments, e.g., $f(t_1, t_2) = \lambda x_3 \ldots x_n.\ f(t_1, t_2, x_3, x_4, \ldots, x_n)$, when $x_3, \ldots, x_n$ are not free in t_1 and t_2. Tuples nevertheless occur as arguments when the components form a conceptual entity, e.g., a two-part ciphertext (β, ζ). To avoid ambiguities with partial applications, we do not merge the function application parentheses with the tuple constructor: $g((\beta, \zeta))$ applies the unary function g to the tuple (β, ζ) and $h(\beta, \zeta)$ is a two-argument function h applied to β and ζ.

A.2.1 *Other Types and Operations*

The singleton type $\mathbb{1}$ has only one element $\circledast$. Pairs (type $\mathring{a} \times \mathring{b}$) come with two projection functions π_1 and π_2. Tuples are identified with pairs nested to the right, i.e., (a, b, c) is identical to $(a, (b, c))$ and $\mathring{a} \times \mathring{b} \times \mathring{c}$ to $\mathring{a} \times (\mathring{b} \times \mathring{c})$. Dually, the sum type $\mathring{a} + \mathring{b}$ models the disjoint union of the types $\mathring{a}$ and $\mathring{b}$ with the injections $\mathit{Left} :: \mathring{a} \Rightarrow \mathring{a} + \mathring{b}$ and $\mathit{Right} :: \mathring{b} \Rightarrow \mathring{a} + \mathring{b}$. The predicates $\mathit{is\text{-}Left}(x)$ and $\mathit{is\text{-}Right}(x)$ check whether x is of the form $\mathit{Left}(\_)$ or $\mathit{Right}(\_)$.

Sets (type $\mathbb{P}(\mathring{a})$) are isomorphic to predicates (type $\mathring{a} \Rightarrow \mathbb{B}$) via the bijections membership $\in$ and set comprehension $\{\, x \mid \_ \,\}$; the empty set is $\{\,\}$. Binary relations are sets of pairs and written in infix, for example, $x\ R\ y$ denotes $(x, y) \in R$.

The datatype $\mathbb{M}(\mathring{a}) = \mathit{None} \mid \mathit{Some}\ \mathring{a}$ adjoins a new element None to $\mathring{a}$ while all existing values in $\mathring{a}$ are prefixed by Some. Maps (partial functions) are modelled as functions of type $\mathring{a} \Rightarrow \mathbb{M}(\mathring{b})$, where None represents undefinedness and $f(x) = \mathit{Some}(y)$ means that f maps x to y. The empty map $\varnothing \equiv (\lambda\_.\ \mathit{None})$ is undefined everywhere. Map update is defined as $f(a \mapsto b) \equiv (\lambda x.\ \mathit{if}\ x = a\ \mathit{then}\ \mathit{Some}(b)\ \mathit{else}\ f(x))$.

A.3 GAME-BASED CRYPTOGRAPHIC PROOFS

Game hopping was originally proposed as a technique for structuring cryptographic security proofs and taming their complexity [21, 52]. Over time, the level of formality has increased; Kilian and Rogaway [30] put forth the idea of games as programs written in a semi-formal language. Bellare and Rogaway [9] suggested that the games be expressed in a probabilistic programming language and the proofs consist of applications of pre-defined program transformations until the security claim is obvious. In their model, a game consists of three phases: initialisation, running the adversary with access to the oracles, and finalization. Halevi [25] picked up this idea and envisioned an interactive proof checker that uses static program analysis to apply simple game transformations specified by the user and to verify their correctness; game hops with complicated probabilistic or algebraic reasoning are to be proven by the user on paper and checked by human reviewers instead of the tool.

In contrast to Halevi's proposal, Shoup [49] objected to being restricted to a fixed toolbox of syntactic program manipulations. He considered games to be a convenient notation for probability distributions rather than formal syntactic objects. This lowers the bar for cryptographers, as they can give free reign to their creativity. So, his proofs mix different types of reasoning, including syntactic program transformations, conventional reasoning about conditional probabilities, and algebraic arguments.

Bellare, Rogaway, and Shoup [9, 49] agree that extending the notation, i.e., programming language, with problem-specific conventions is essential, as otherwise, the definitions and proofs become unreadable and hard to check.

A.3.1 *CryptHOL's Approach to Game-based Proofs*

In CryptHOL, a game is just a (discrete sub-)probability distribution expressed in CryptHOL's notation, but the notation has a well-defined formal meaning. Thus, CryptHOL combines the best of two worlds: it achieve the extensibility and flexibility that Shoup calls for because a game is just a distribution rather than a program written in a fixed programming language. At the same time, like envisioned by Bellare and Rogaway, it can express program transformations, prove them correct, and apply them to the syntactic objects as the notation is formal. Instead of constructing sequences of games by applying game transformations like in Halevi's vision, CryptHOL users themselves explicitly specify all the intermediate games and then Isabelle/HOL checks that the given justifications for the transformations are correct. This yields declarative proofs, which are in general easier to understand and maintain as experience has shown in other domains [51].

B GUIDE TO SOURCE THEORY FILES

In what follows, we provide a guide for the reader to navigate the formalization source. For brevity, in the body of this thesis, we only provided informal text and used pictures to present the main lemmas and theorems. However, all the definitions and lemmas are formalized and verified in the Isabelle/HOL proof assistant. The theory files are split in two entries on the Archive of Formal Proofs (AFP). The first entry [37] formalizes the contents of Chapters 3, 4, and 5. The second entry [38] includes the formal results of Chapter 6 and Section 4.2.1.

B.1 AFP ENTRY I

The root directory contains many theory files and an **Examples** folder that stores the formalization of the case study in Section 5.4. Each theory file contains the lemmas and definitions corresponding to its name:

- **Resource.thy** formalizes resources, their parallel composition, and the notion of interface respecting resources (Section 3.1).
- **Converter.thy** formalizes converters, their sequential and parallel composition, the notion of interface respecting converters, and the attachment of converters to resources (Sections 3.2, 3.3).
- **Wiring.thy** formalizes the wiring converters (Section 3.3).
- **Random_System.thy** formalizes the notion of trace and trace equivalence, where the propositions `trace'_eqI_sim` and `trace_callee_-complete` prove the two directions of Theorem 1.
- **Distinguisher.thy** formalizes the notion of distinguishers, where lemmas `connect_cong_trace` and `distinguish_trace_eq` formalize the two directions of Theorem 3.
- **Constructive_Cryptography.thy** formalizes the notion of secure realization (Def. 7) using the locale `constructive_security`. The composability theorems 4–7 are formalized by the theorems `construc-tive_security_trivial`, `composability`, `parallel_constructive_-security`, and `lifting` respectively.

- **Converter_Rewrite.thy** formalizes the notion of equivalence of resources and converters subject to assumptions on the context given by interface types. The thesis does not delve into this technicality.

The case study has four main theory files, which are stored in folder **Secure_Channel** listed inside the examples folder.

- **System_Construction.thy** contains the generic formalization of channels, encryption schemas, and message authentication schemas.
- **One_Time_Pad.thy** formalizes the first part of our case study, where we instantiate a one-time-pad encryption scheme and use it to construct a secure channel from a key and an authentic channel (lemma `one_time_pad`).
- **Message_Authentication_Code.thy** constitutes the formal construction of an authentic channel from a random function and an insecure channel. Lemmas `trace_eq_lazy`, `game_difference`, and `trace_eq_sim` constitute the three reduction steps and the lemma `secure_mac` states the constructive security of our simple MAC.
- **Secure_Channel.thy** stores the final composition lemma `mac_otp` that states the constructive security of the aforementioned construction's composition.

B.2 AFP ENTRY II

The root directory consists of many theory files, the **Specifications** folder that contains the ideal specifications for keys and channels, and the **Constructions** folder that stores the formalization of Section 6.4's case study. The theory files in the root directory contain the lemmas and definitions that correspond to their names:

- **Fold_Spmf.thy** and **Goodies.thy** formalize the probabilistic fold function and a series of helper lemmas.
- **More_CC.thy** and **State_Isomorphism.thy** formalize our extensions to the theory of resources and converters. The lemmas and definitions in this theory file are not specific to fused resources only.
- **Observe_Failure.thy** formalizes the indistinguishability of resources when a bottom element $\perp$ is added to every distribution's sample space to model failures.

- `Fused_Resource.thy` formalizes fused resources, their trace equivalence, and various operators on core and rest records. In particular, the proposition `trace'_eq_simI_upto` proves the Theorem 2 for fused resources.
- `Construction_Utility.thy` formalizes common building blocks for defining compound fused resources. In particular, the propositions `parallel_oracle_fuse` and `attach_parallel_fuse'` prove the Lemmas 1 and 2; and the propositions `fuse_ishift_core_to_rest` and `move_simulator_interface` prove the Lemmas 3 and 4 respectively.
- `Concrete_Security.thy` formalizes the notion of information-theoretic concrete security. In particular, the propositions `constructive_security_obsf_trivial`, `parallel_constructive_security_obsf`, `constructive_security_obsf_composability`, and `constructive_security_obsf_lifting_usr` prove the four claims of Theorem 8.
- `Asymptotic_Security.thy` extends the above to the asymptotic notion of security.

The **Specifications** folder provides the ideal specifications for keys and channels. We explained the details of **Key.thy** in Section 6.4.

The case study has three main theory files, which are stored in the folder **Constructions**.

- `One_Time_Pad.thy` formalizes the construction of a secure channel from a key and an authenticated channel using one-time-pad.
- `Diffie_Hellman.thy` constitutes the formal construction of a key from two authenticated channels using the Diffie-Hellman key exchange.
- `DH_OTP.thy` stores the final lemma that states the security of the aforementioned constructions' composition.

BIBLIOGRAPHY

[1] Michael Backes, Birgit Pfitzmann, and Michael Waidner. "A General Composition Theorem for Secure Reactive Systems". In: *Theory of Cryptography (TCC 2003), Proceedings*. Vol. 2951. LNCS. Springer, 2004, pp. 336–354. DOI: `10/d4t8ws`.

[2] Manuel Barbosa, Gilles Barthe, Karthikeyan Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. "Sok: Computer-aided Cryptography". In: *Symposium on Security and Privacy (S&P 2021), Proceedings*. IEEE Computer Society, 2021, pp. 123–141.

[3] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. "Formal Certification of Code-based Cryptographic Proofs". In: *Principles of Programming Languages (POPL 2009), Proceedings*. Vol. 44. ACM, 2009, pp. 90–101. DOI: `10/cf2khq`.

[4] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. "Probabilistic Relational Hoare Logics for Computer-aided Security Proofs". In: *Mathematics of Program Construction (MPC 2012), Proceedings*. Vol. 7342. LNCS. Springer, 2012, pp. 1–6. DOI: `10/ggpqkz`.

[5] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. "Computer-aided Security Proofs for the Working Cryptographer". In: *Advances in Cryptology (CRYPTO 2011), Proceedings*. Vol. 6841. LNCS. Springer, 2011, pp. 71–90. DOI: `10/cnpgbz`.

[6] Gilles Barthe, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. "Coupling Proofs Are Probabilistic Product Programs". In: *Principles of Programming Languages (POPL 2017), Proceedings*. ACM, 2017, pp. 161–174. DOI: `10/gjrsnd`.

[7] David Basin, Andreas Lochbihler, Uelil Maurer, and S. Reza Sefidgar. "Abstract Modeling of Systems Communication in Constructive Cryptography Using CryptHOL". In: *Computer Security Foundations Symposium (CSF 2021), Proceedings*. IEEE Computer Society, 2021.

[8] David Basin, Andreas Lochbihler, and S. Reza Sefidgar. "CryptHOL: Game-based Proofs in Higher-order Logic". In: *Journal of Cryptology* 33.2 (2020), pp. 494–566. DOI: `10/ggpqf5`.

[9] Mihir Bellare and Phillip Rogaway. "The Security of Triple Encryption and a Framework for Code-based Game-playing Proofs". In: *Advances in Cryptology (EUROCRYPT 2006), Proceedings*. Vol. 4004. LNCS. Springer, 2006, pp. 409–426. DOI: 10/fdnq34.

[10] Matthias Berg. "Formal Verification of Cryptographic Security Proofs". PhD thesis. Saarland University, 2013. DOI: 10/gjrsng.

[11] Bruno Blanchet. "A Computationally Sound Mechanized Prover for Security Protocols". In: *Security and Privacy (S&P 2006), Proceedings*. Vol. 5. IEEE Computer Society, 2006, pp. 140–154. DOI: 10/fjqxcq.

[12] Bruno Blanchet. "Composition Theorems for Cryptoverif and Application to TLS 1.3". In: *Computer Security Foundations (CSF 2018), Proceedings*. IEEE Computer Society, 2018, pp. 16–30. DOI: 10/gjrsnh.

[13] Jasmin Christian Blanchette, Johannes Hölzl, Andreas Lochbihler, Lorenz Panny, Andrei Popescu, and Dmitriy Traytel. "Truly Modular (co)datatypes for Isabelle/HOL". In: *Interactive Theorem Proving (ITP 2014), Proceedings*. Vol. 8558. LNCS. Springer, 2014, pp. 93–110. DOI: 10/gjrsnj.

[14] David Butler, David Aspinall, and Adrià Gascón. "How to Simulate It in Isabelle: Towards Formal Proof for Secure Multi-party Computation". In: *Interactive Theorem Proving (ITP 2017), Proceedings*. Vol. 10499. LNCS. Springer, 2017, pp. 114–130. DOI: 10/gjrsnk.

[15] Jan Camenisch, Robert R. Enderlein, Stephan Krenn, Ralf Küsters, and Daniel Rausch. "Universal Composition with Responsive Environments". In: *Advances in Cryptology (ASIACRYPT 2016), Proceedings, Part II*. Vol. 10032. LNCS. Springer, 2016, pp. 807–840. DOI: 10/gjrsnm.

[16] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *Foundations of Computer Science (FOCS 2001), Proceedings*. IEEE Computer Society, 2001, pp. 136–145. DOI: 10/dh6zkt.

[17] Ran Canetti, Alley Stoughton, and Mayank Varia. "EasyUC: Using EasyCrypt to Mechanize Proofs of Universally Composable Security". In: *Computer Security Foundations Symposium, (CSF 2019), Proceedings*. IEEE Computer Society, 2019, pp. 167–183. DOI: 10/gjrsnn.

[18] Alonzo Church. "A Formulation of the Simple Theory of Types". In: *The Journal of Symbolic Logic* 5.2 (1940), pp. 56–68. DOI: 10/br4892.

[19] Dion Coumans and Bart Jacobs. “Scalars, Monads, and Categories”. In: *Quantum Physics and Linguistics: A Compositional, Diagrammatic Discourse*. Ed. by Chris Heunen, Mehrnoosh Sadrzadeh, and Edward Grefenstette. Oxford University Press, 2013, pp. 184–216. DOI: 10/gjrsnp.

[20] Whitfield Diffie and Martin E. Hellman. “New Directions in Cryptography”. In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654. DOI: 10/cmtvwm.

[21] Shafi Goldwasser and Silvio Micali. “Probabilistic Encryption”. In: *Journal of Computer and System Sciences* 28.2 (1984), pp. 270–299. DOI: 10/c67w9s.

[22] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. “Knowledge complexity of interactive proof systems”. In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208. DOI: 10/cqx3km.

[23] Michael J. C. Gordon and Andrew M. Pitts. “The HOL Logic and System”. In: *Towards Verified Systems*. Ed. by Jonathan Bowen. Vol. 2. Real-Time Safety Critical Systems. Elsevier, 1994, pp. 49–70. DOI: 10/gjrsnq.

[24] Helene Haagh, Aleksandr Karbyshev, Sabine Oechsner, Bas Spitters, and Pierre-Yves Strub. “Computer-aided Proofs for Multiparty Computation with Active Security”. In: *Computer Security Foundations (CSF 2018), Proceedings*. IEEE Computer Society, 2018, pp. 119–131. DOI: 10/gjrsnr.

[25] Shai Halevi. “A Plausible Approach to Computer-aided Cryptographic Proofs”. In: *IACR Cryptology ePrint Archive* (2005). https://eprint.iacr.org/2005/181, p. 181.

[26] Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. “Generic Trace Semantics Via Coinduction”. In: *Logical Methods in Computer Science* 3.4 (2007), pp. 1–36. DOI: 10/cj83zq.

[27] Dennis Hofheinz and Victor Shoup. “GNUC: A New Universal Composability Framework”. In: *Journal of Cryptology* 28.3 (2015), pp. 423–508. DOI: 10/f7m5s5.

[28] Bart Jacobs, Alexandra Silva, and Ana Sokolova. “Trace Semantics Via Determinization”. In: *Journal of Computer and System Sciences* 81.5 (2015), pp. 859–879. DOI: 10/gjrsns.

[29] Daniel Jost and Ueli Maurer. "Overcoming Impossibility Results in Composable Security Using Interval-wise Guarantees". In: *Advances in Cryptology (CRYPTO 2020), Proceedings, Part I*. Vol. 12170. LNCS. Springer, 2020, pp. 33–62. DOI: `10/gjrsnt`.

[30] Joe Kilian and Phillip Rogaway. "How to Protect DES against Exhaustive Key Search (an Analysis of DESX)". In: *Journal of Cryptology* 14.1 (2001), pp. 17–35. DOI: `10/djzxkm`.

[31] Ondrej Kuncar and Andrei Popescu. "A Consistent Foundation for Isabelle/HOL". In: *Interactive Theorem Proving (ITP 2015), Proceedings*. Vol. 9236. LNCS. Springer, 2015, pp. 234–252. DOI: `10/gjrsnv`.

[32] Ondrej Kuncar and Andrei Popescu. "Comprehending Isabelle/HOL's Consistency". In: *European Symposium on Programming (ESOP 2017), Proceedings*. Vol. 10201. LNCS. Springer, 2017, pp. 724–749. DOI: `10/gjrsnw`.

[33] Ondrej Kuncar and Andrei Popescu. "Safety and Conservativity of Definitions in HOL and Isabelle/HOL". In: *Principles of Programming Languages (POPL 2017), Proceedings*. Vol. 2. ACM, 2017, pp. 1–26. DOI: `10/gc7r3x`.

[34] Alexander Kurz, Stefan Milius, Dirk Pattinson, and Lutz Schröder. "Simplified Coalgebraic Trace Equivalence". In: *Software, Services, and Systems*. Ed. by Rocco De Nicola. LNCS. Springer, 2015, pp. 75–90. DOI: `10/gjrsnx`.

[35] Ralf Küsters and Max Tuengerthal. "The IITM Model: A Simple and Expressive Model for Universal Composability". In: *Journal of Cryptology* 2013.4 (2013), pp. 1461–1584. DOI: `10/gjrsnz`.

[36] Andreas Lochbihler. "Probabilistic Functions and Cryptographic Oracles in Higher Order Logic". In: *European Symposium on Programming (ESOP 2016), Proceedings*. Vol. 9632. LNCS. Springer, 2016, pp. 503–531. DOI: `10/gjrsn4`.

[37] Andreas Lochbihler and S. Reza Sefidgar. "Constructive Cryptography in HOL". In: *Archive of Formal Proofs* (2018). `https://isa-afp.org/entries/Constructive_Cryptography.html`, Formal proof development.

[38] Andreas Lochbihler and S. Reza Sefidgar. "Constructive Cryptography in HOL: the Communication Modeling Aspect". In: *Archive of Formal Proofs* (2021). `https://isa-afp.org/entries/Constructive_Cryptography_CM.html`, Formal proof development.

[39] Andreas Lochbihler, S. Reza Sefidgar, David Basin, and Ueli Maurer. "Formalizing Constructive Cryptography Using CryptHOL". In: *Computer Security Foundations Symposium (CSF 2019), Proceedings*. IEEE Computer Society, 2019, pp. 152–166. DOI: `10/gjrsn5`.

[40] Ueli Maurer. "Constructive Cryptography - A New Paradigm for Security Definitions and Proofs". In: *Theory of Security and Applications - Joint Workshop (TOSCA 2011), Revised Selected Papers*. Vol. 6993. LNCS. Springer, 2011, pp. 33–56. DOI: `10/fx6d5w`.

[41] Ueli Maurer and Renato Renner. "Abstract Cryptography". In: *Innovations in Computer Science (ICS 2010), Proceedings*. Tsinghua University Press, 2011, pp. 1–21.

[42] Ueli Maurer and Renato Renner. "From Indifferentiability to Constructive Cryptography (and Back)". In: *Theory of Cryptography Conference (TCC 2016), Proceedings, Part I*. Vol. 9985. LNCS. Springer, 2016, pp. 3–24. DOI: `10/gjrsn6`.

[43] Ueli M. Maurer. "Indistinguishability of Random Systems". In: *Advances in Cryptology (EUROCRYPT 2002), Proceedings*. Vol. 2332. LNCS. Springer, 2002, pp. 110–132. DOI: `10/fc9465`.

[44] Tobias Nipkow and Gerwin Klein. *Concrete Semantics - with Isabelle/HOL*. Springer, 2014. DOI: `10/gjmn46`.

[45] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle - A Proof Assistant for Higher-Order Logic*. Vol. 2283. LNCS. Springer, 2002. DOI: `10/ccdhts`.

[46] Adam Petcher and Greg Morrisett. "The Foundational Cryptography Framework". In: *Principles of Security and Trust (POST 2015), Proceedings*. Vol. 9036. LNCS. Springer, 2015, pp. 53–72. DOI: `10/gjrsn7`.

[47] Andrew M. Pitts. "The HOL Logic". In: *Introduction to HOL a Theorem Proving Environment for Higher Order Logic*. Ed. by Michael J. C. Gordon and Tom F. Melham. Cambridge University Press, 1993, pp. 191–232.

[48] Jan J. M. M. Rutten. "Universal Coalgebra: A Theory of Systems". In: *Theoretical Computer Science* 249.1 (2000), pp. 3–80. DOI: `10/fqrjpn`.

[49] Victor Shoup. "Sequences of Games: A Tool for Taming Complexity in Security Proofs". In: *IACR Cryptology ePrint Archive* (2004). `https://eprint.iacr.org/2004/332`, p. 332.

[50] Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. “Generalizing the Powerset Construction, Coalgebraically”. In: *Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010), Proceedings*. Vol. 8. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010, pp. 272–283. DOI: 10/gjrsn8.

[51] Freek Wiedijk. “A Synthesis of the Procedural and Declarative Styles of Interactive Theorem Proving”. In: *Logical Methods in Computer Science* 8 (2012). DOI: 10/gfkhmv.

[52] Andrew C. Yao. “Theory and Application of Trapdoor Functions”. In: *Foundations of Computer Science (FOCS 1982), Proceedings*. IEEE Computer Society, 1982, pp. 80–91. DOI: 10/dtk2wv.

CURRICULUM VITAE

PERSONAL DATA

Name	Seyed Reza Sefidgar
Address	Paul-Feyerabend-Hof 5, 8049 Zurich
Email	`reza.sefidgar@inf.ethz.ch`

EDUCATION

Oct 2013 – Feb 2016	M.Sc. Informatik Technische Universität München (TUM) Munich, Germany
Oct 2008 – Mar 2012	B.Sc. Computer Science University of Tehran Tehran, Iran
Oct 2006 – Sep 2008	Bachelor's Candidate in Material Science University of Tehran *(switched to Computer Science)*

WORK EXPERIENCE

Aug 2014 – Jul 2015	Student Researcher and Developer *FML institute at TUM* Munich, Germany
Apr 2012 – Sep 2013	Solution Architecht *Ministry of Economic Affairs and Finance* Tehran, Iran
Oct 2009 – Dec 2011	Full Stack Developer *0&1 Corporation* Tehran, Iran
2006 – 2009	Freelance Developer

TEACHING EXPERIENCE

2017 – 2021	Teaching Assistant Functional Programming and Formal Methods ETH Zurich, Zurich, Switzerland
2016 – 2020	Teaching Assistant Introduction to Programming ETH Zurich, Zurich, Switzerland
2009 – 2011	Teaching Assistant Fundamentals of Computer Science University of Tehran, Tehran, Iran
2011	Teaching Assistant Principles of Computer Systems University of Tehran, Tehran, Iran

PUBLICATIONS

ARTICLES IN PEER-REVIEWED JOURNALS:

- David Basin, Andreas Lochbihler, and S. Reza Sefidgar. "CryptHOL: Game-based Proofs in Higher-order Logic". In: *Journal of Cryptology* 33.2 (2020), pp. 494–566. DOI: `10/ggpqf5`.
- Peter Lammich and S. Reza Sefidgar. "Formalizing Network Flow Algorithms: A Refinement Approach in Isabelle/HOL". In: *Journal of Automated Reasoning* 62.2 (2019), pp. 261–280. DOI: `10/gjrsn3`.

CONFERENCE CONTRIBUTIONS:

- David Basin, Andreas Lochbihler, Uelil Maurer, and S. Reza Sefidgar. "Abstract Modeling of Systems Communication in Constructive Cryptography Using CryptHOL". In: *Computer Security Foundations Symposium (CSF 2021), Proceedings*. IEEE Computer Society, 2021.
- Andreas Lochbihler, S. Reza Sefidgar, David Basin, and Ueli Maurer. "Formalizing Constructive Cryptography Using CryptHOL". In: *Computer Security Foundations Symposium (CSF 2019), Proceedings*. IEEE Computer Society, 2019, pp. 152–166. DOI: `10/gjrsn5`.
- Peter Lammich and S. Reza Sefidgar. "Formalizing the Edmonds-Karp algorithm". In: LNCS 9807 (2016), pp. 219–234. DOI: `10/gjrsn2`.

ARCHIVE OF FORMAL PROOFS:

- Andreas Lochbihler and S. Reza Sefidgar. "Constructive Cryptography in HOL: the Communication Modeling Aspect". In: *Archive of Formal Proofs* (2021). `https://isa-afp.org/entries/Constructive_Cryptography_CM.html`, Formal proof development.
- Andreas Lochbihler and S. Reza Sefidgar. "Constructive Cryptography in HOL". In: *Archive of Formal Proofs* (2018). `https://isa-afp.org/entries/Constructive_Cryptography.html`, Formal proof development.

- Peter Lammich and S. Reza Sefidgar. “Flow Networks and the Min-Cut-Max-Flow Theorem”. In: *Archive of Formal Proofs* (2017). `https://isa-afp.org/entries/Flow_Networks.html`, Formal proof development.
- Peter Lammich and S. Reza Sefidgar. “Formalizing Push-Relabel Algorithms”. In: *Archive of Formal Proofs* (2017). `https://isa-afp.org/entries/Prpu_Maxflow.html`, Formal proof development.
- Andreas Lochbihler, S. Reza Sefidgar, and Bhargav Bhatt. “Game-based cryptography in HOL”. In: *Archive of Formal Proofs* (2017). `https://isa-afp.org/entries/Game_Based_Crypto.html`, Formal proof development.
- Peter Lammich and S. Reza Sefidgar. “Formalizing the Edmonds-Karp Algorithm (proof document)”. In: *Archive of Formal Proofs* (2016). `https://isa-afp.org/entries/EdmondsKarp_Maxflow.html`, Formal proof development.

FURTHER TECHNICAL REPORTS:

- Andreas Lochbihler and S. Reza Sefidgar. “A tutorial introduction to CryptHOL”. In: *IACR Cryptology ePrint Archive* (2018). `https://eprint.iacr.org/2018/941`, p. 941.