Abstract Modeling of System Communication in Constructive Cryptography using CryptHOL

David Basin
Dept. of Computer Science
ETH Zürich, Switzerland
basin@inf.ethz.ch

Andreas Lochbihler
Digital Asset
Zurich, Switzerland
mail@andreas-lochbihler.de

Ueli Maurer
Dept. of Computer Science
ETH Zürich, Switzerland
maurer@inf.ethz.ch

S. Reza Sefidgar
Dept. of Computer Science
ETH Zürich, Switzerland
reza.sefidgar@inf.ethz.ch

Abstract—**Proofs in simulation-based frameworks have the greatest rigor when they are machine checked. But the level of details in these proofs surpasses what the formal-methods community can handle with existing tools. Existing formal results consider streamlined versions of simulation-based frameworks to cope with this complexity. Hence, a central question is how to abstract details from composability results and enable their formal verification.**

In this paper, we focus on the modeling of system communication in composable security statements. Existing formal models consider fixed communication patterns to reduce the complexity of their proofs. However, as we will show, this can affect the reusability of security statements. We propose an abstract approach to modeling system communication in Constructive Cryptography that avoids this problem. 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. As a case study, we formalize the construction of a secure channel using Diffie-Hellman key exchange and a one-time-pad.

I. Introduction

A. Problem Context

Simulation-based frameworks [1], [7], [11], [13], also called composability frameworks, pave the way for modular reasoning about cryptographic constructs independently of their application context. They follow the *Ideal-world—Real-world* [9] paradigm to security proofs, where a distributed protocol's security is defined by comparing its execution, i.e., the real world, to an idealized specification that intrinsically satisfies the desired properties.

Devising rigorous security statements in these frameworks is very challenging. The way that notions of algorithms, runtime, corruptions, and the like are baked into composability theorems in frameworks such as Universal Composability (UC) [7] introduces considerable details into composable security statements. The resulting complexity is beyond what the current scientific review processes can afford. The formal-methods community has not been able to alleviate the situation either. The level of details in simulation-based frameworks surpasses what the existing formal-methods tools can reasonably handle.

Simulation-based frameworks' approach to modeling system 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 [6]. As such, existing formal results such as EasyUC [8] consider a simplified version of these frameworks with restricted communication capabilities between components. However, as we will show in Section III-A, such simplifications affect the reusability of security statements.

Constructive Cryptography (CC) [17], [18], [19] proposes a fundamental shift in how security statements are made and proved. It introduces an abstract approach to composable security arguments that 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. However, the existing CC results [12], [16] do not delve into the details of system communication.

B. Our Solution

We propose an abstract approach to modeling system communication in composable security statements by means of generic ideal functionalities. We follow CC's methodology whereby specifications (and implementations) are *sets of* ideal (or real respectively) functionalities. This is in contrast to UC-style frameworks that define such functionalities as single objects and enables us to model system communication without introducing a centralized adversary machine.

First, we propose a semantic domain called *Fused Resource Templates* (FRT) that abstracts over the system communication patterns. FRTs are suitable for mechanizing composable security proofs, and we formalize all the results presented in this paper using CryptHOL [3], [14], a framework for mechanizing cryptographic proofs in the proof assistant Isabelle/HOL [20]. Second, we show how FRTs constitute an instantiation of the CC framework. We formalize operators that enable us to compose FRTs and thereby define complex FRTs in terms of simpler ones. We then introduce the notion of secure constructions using FRTs and provide composition theorems for such constructions. Finally, we demonstrate the practicality of our approach in a case study. We formalize the construction of a secure channel by composing Diffie-Hellman key exchange with one-time-pad encryption.

Our result is based on Lochbihler et al.'s formalization of CC [16]. To keep this paper self contained and preserve its presentation flow, we recap this work in various parts of the paper. We provide a detailed comparison in Section VI.

C. Contributions

The results presented in this paper contribute to both the cryptography and the formal-methods communities. Our approach to capturing communication patterns increases the abstraction of ideal functionalities and makes composable security arguments amenable to mechanized verification. In detail:

- We propose the first CC instantiation that explicates the details of system communication. The role of system communication models has not been studied in the CC literature, although we show that the reusability and modularity of CC proofs depend on it (cf. Section III-A). We propose an abstract approach to capturing system communication patterns in CC proofs.
- We formalize our approach in a theorem prover and thereby enable the mechanized verification of composable security arguments. By abstracting over system communication patterns, we allow protocol designers to focus on the main security concerns while ensuring the reusability and modularity of security statements. We provide proof rules that enable modular reasoning about concrete (and asymptotic) security statements. We use these rules to formalize Diffie-Hellman key exchange and the one-time-pad that are comparatively simpler and shorter than any existing formalization.

D. Structure

We start with a short recap of simulation-based frameworks and CC in Section II. In Section III, we explain the importance of precise system communication modeling for the modularity and reusability of security proofs. We then present our formalization of FRTs and their operators. In Section IV, we define concrete security in terms of FRT constructions and provide theorems for composing security proofs. In Section V, we present our case study, which demonstrates our approach's applicability. Finally, in Sections VI and VII, we compare with related work and draw conclusions.

All results presented in this paper are formalized using Isabelle/HOL. Appendix A explains the structure of the formalization source, which is available on the Archive of Formal Proofs [15].

II. Preliminaries

This section introduces background that is used in the rest of the paper. We review the composability of security proofs and how it its treatment in simulation-based frameworks and CC. We focus on system communication modeling in composable security arguments. In each subsection, we use the notation that is common in the research community for the topic under discussion. The paper-specific notation, which is indexed in Appendix B, will be presented when formally introduced.

A. Composable Security Arguments

The ideal-world—real-world [9] 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.

B. UC-style Frameworks

In simulation-based frameworks [1], [7], [11], [13], (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 wish 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. When ρ calls $\mathcal{F}$ as a subroutine, two message broadcasts are used to transfer control between ρ and $\mathcal{F}$: the first message carries the "request" details and 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 [6], protocol descriptions in simulation-based frameworks include meta-level and model-specific

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

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 resolve this conundrum. The level of detail 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 by restricting the communication and corruption models [8]. Hence, they are only applicable in a limited range of scenarios.

C. Constructive Cryptography

Constructive Cryptography (CC) [17], [18], [19] is an abstract paradigm for developing a theory of cryptography. Rather than focusing on concrete systems and proving their properties, CC studies 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 are just means for abstracting over the details of similar systems.

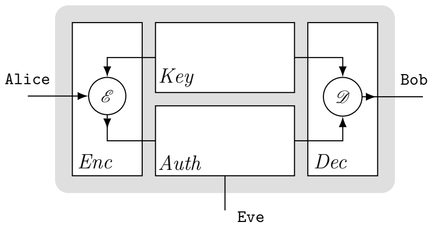
Modular reasoning about systems is based on the concepts of *Resources* and *Converters*. Every aspect of individual systems, including the adversary's capabilities and information leakage is made explicit as a resource with named input/output interfaces. *Specifications* are sets of resources and represent systems that share a common behavior. Parties can change a resource's behavior by attaching converters to their designated interfaces on that resource. It is possible to access multiple resources simultaneously 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 the Cartesian product $[\mathcal{R}, \mathcal{S}] = \{[R, S] \mid R \in \mathcal{R} \wedge S \in \mathcal{S}\}$. 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 the 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 $\pi[\mathcal{R}, \mathcal{T}] \subseteq [\mathcal{S}, \mathcal{T}]$.

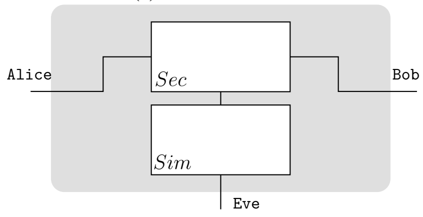
Common security notions are expressed as particular forms of specifications. For example, consider the simulator-based notion of information-theoretic security. Using a simulator σ corresponds to specifications of the form $\sigma\mathcal{S}$, where the simulator is defined in terms of a converter that only attaches to the adversary interfaces. Indistinguishability is analysed by *relaxing* specifications as $\mathcal{R}^\epsilon = \{S \mid R \in \mathcal{R} \wedge \mathfrak{d}(R, S) \leq \epsilon\}$,

[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

Fig. 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 built from the resources and converters in its interior.

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 $\mathtt{A}$ and $\mathtt{B}$ and the adversary $\mathtt{E}$, 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 $\sigma = z^{\mathtt{E}}$, i.e., a converter z attached to interface $\mathtt{E}$, such that $\pi\mathcal{R} \subseteq (\sigma\mathcal{S})^\epsilon$. Computational security is defined analogously using a similar relaxation that 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.

To better understand the above concepts, consider the following example. 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 1a depicts such a scenario in CC. The protocol π 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 1a and 1b.

III. A New CC Semantic Domain

We propose Fused Resource Templates (FRT) as a class of CC specifications that allow us to abstract over the communication patterns in security proofs and is suitable for mechanized verification. To motivate our proposal, we show that compositional security statements should not be tied to a fixed communication pattern, as is the case in the existing CC case studies.

A. 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 II-C. 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.

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 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 II-A, 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 already 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 need 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.

B. Fused Resource Templates

We define FRTs in terms of Lochbihler et al.'s formalization of resources [16]. Each FRT describes a parametrized resource: all instances share a common behavior while each instance exhibits a particular party-activation pattern.

1) A Short Recap of Resources: In CC [12], probabilistic reactive systems are called resources. Consider a bit string that is sampled uniformly from the set of bit strings of length n. One can model such a random bit string as a probabilistic reactive system that gets an integer n as input and outputs n random bits. In general, each system output may depend on its previous inputs and outputs, e.g. a random oracle that answers repeated queries with the same output. It is therefore convenient to model resources by a transition function and an initial state: for a state s and an input x, the transition function $tr(s, x)$ returns a probability distribution over the successor state and the output.

Lochbihler et al. [16] hide the internal state of resources to simplify the proofs and definitions. The resource type existentially quantifies over the state type in the pair of the transition function and the initial state:

$$\mathbb{R}(\mathfrak{q}, \mathfrak{r}) = \exists \mathfrak{s}.\ (\mathfrak{s} \Rightarrow \mathfrak{q} \Rightarrow \mathbb{D}(\mathfrak{r} \times \mathfrak{s})) \times \mathfrak{s}, \qquad (1)$$

where $\mathfrak{q}$ represents the type of inputs (queries), $\mathfrak{r}$ the type of outputs (responses), $\times$ the product type, $\Rightarrow$ the function space, and $\mathbb{D}(\mathfrak{a})$ probability distributions over $\mathfrak{a}$. The Isabelle/HOL formalization uses co-inductive datatypes [4], in short co-datatypes, instead of existential types, following the co-algebraic approach to modeling transition systems [21]. Starting from the initial state, the co-algebraic approach unfolds the transition functions into a possibly unbounded probabilistic tree, where the successor state is replaced by the subtree of its behavior. The following co-datatype formalizes such trees in Isabelle/HOL:

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

The constructor *Resource* takes a family of probability distributions as argument, one for each possible query $\mathfrak{q}$. Each distribution contains the responses $\mathfrak{r}$ and the corresponding subtree $\mathbb{R}(\mathfrak{q}, \mathfrak{r})$ that captures the resource's behavior after the query-response interaction. This unfolding of the transition function identifies a state s with its unfolding under tr. This way, we need not represent the internal state of the resource and thus emulate the state hiding in (1).

The identification of states with its unfoldings can be seen in the transition function *run* on resources: $\mathit{run} : \mathbb{R}(\mathfrak{q}, \mathfrak{r}) \Rightarrow \mathfrak{q} \Rightarrow \mathbb{D}(\mathfrak{r} \times \mathbb{R}(\mathfrak{q}, \mathfrak{r}))$, where $e : \mathfrak{t}$ denotes that expression e has type $\mathfrak{t}$. This type is the same as for $\mathfrak{s} \Rightarrow \mathfrak{q} \Rightarrow \mathbb{D}(\mathfrak{r} \times \mathfrak{s})$ except that the state type $\mathfrak{s}$ is replaced by the tree $\mathbb{R}(\mathfrak{q}, \mathfrak{r})$. Formally, the co-datatype definition is the final co-algebra of such transition functions. The final co-algebra identifies resources with bisimilar input-output behaviour. No context can therefore distinguish bisimilar resources and we can thus replace one by the other without having to discharge preconditions.

The existing formalizations [3], [16] introduce two more components that facilitate defining resources with multiple interfaces. First, CryptHOL's oracle composition operator $+_{\mathbb{O}}$ interleaves two transition functions $tr_i : \mathfrak{s} \Rightarrow \mathfrak{q}_i \Rightarrow \mathbb{D}(\mathfrak{r}_i \times \mathfrak{s})$ (for $i = 1, 2$) operating on a shared state of type $\mathfrak{s}$ into one transition function $tr_1 +_{\mathbb{O}} tr_2 : \mathfrak{s} \Rightarrow \mathfrak{q}_1 + \mathfrak{q}_2 \Rightarrow \mathbb{D}((\mathfrak{r}_1 + \mathfrak{r}_2) \times \mathfrak{s})$, where $\mathfrak{q} + \mathfrak{r}$ denotes the disjoint union of the types $\mathfrak{q}$ and $\mathfrak{r}$. Following the CryptHOL terminology, we use the terms "oracle" and "probabilistic transition function" interchangeably in this paper. Second, the function *RES* converts an oracle into a resource, i.e., it unfolds the transition function into a tree and thereby introduces the existential type quantifier and hides the state. $\mathit{RES}(tr, s_0)$ is the co-inductive counterpart of a probabilistic transition system where the state is explicit, represented using the transition function tr and the initial state s_0.

2) 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 member exhibits 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 *Remainder* parts. When an FRT is instantiated, the core part is *fused* with one of the remainder parts and results in a special kind of resource, which we call a *Fused Resource*, that enforces one-way information flow from its remainder 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 $cr = (\!|\mathit{cinit} := \cdots, \mathit{cpoke} := \cdots, \mathit{cfunc} := \cdots|\!)$ with three fields: an initial core state $\mathit{cinit} : \mathfrak{s}$, a probabilistic event handler $\mathit{cpoke} : \mathfrak{s} \Rightarrow \mathfrak{e} \Rightarrow \mathbb{D}(\mathfrak{s})$, and a probabilistic transition function $\mathit{cfunc} : \mathfrak{s} \Rightarrow \mathfrak{q}_c \Rightarrow \mathbb{D}(\mathfrak{r}_c \times \mathfrak{s})$. 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 input-output 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 remainder part communicates with the core part via poke events. They are defined in terms of a record $rm = (\!|\mathit{rinit} := \cdots, \mathit{rfunc} := \cdots|\!)$ that consists of an initial remainder state $\mathit{rinit} : \mathfrak{t}$ and a probabilistic event-augmented transition function $\mathit{rfunc} : \mathfrak{t} \Rightarrow \mathfrak{q}_r \Rightarrow \mathbb{D}((\mathfrak{r}_r \times \mathbb{L}(\mathfrak{e})) \times \mathfrak{t})$, where $\mathbb{L}(\mathfrak{e})$ denotes the type of events list. The *cfunc* field defines the input-output behavior of remainder interfaces as well as the information they leak to the core part: given the current remainder state and an input, it defines a probability distribution on the output, the successor remainder state, and a list of events that the fused resource's core part will be notified about.

Given a $rm = (\!|\mathit{rinit} := ri_{def}, \mathit{rfunc} := rf_{def}|\!)$ and a $cr = (\!|\mathit{cinit} := ci_{def}, \mathit{cpoke} := cp_{def}, \mathit{cfunc} := cf_{def}|\!)$ with the types defined above, the Fusing function $\mathit{fuse}(cr, rm)$ outputs a probabilistic transition function with type $(\mathfrak{s} \times \mathfrak{t}) \Rightarrow \mathfrak{q}_c + \mathfrak{q}_r \Rightarrow \mathbb{D}((\mathfrak{r}_c + \mathfrak{r}_r) \times (\mathfrak{s} \times \mathfrak{t}))$ that enforces a one-way information flow from rm to cr by means of events. Here, $+$ denotes disjoint union and we write *Left* _ and *Right* _ for the injections. Let tr_{fuse} and (s, t) denote the resulting oracle and its internal state respectively. For a given query q, the output of $tr_{\mathit{fuse}}((s, t), q)$ is a probability distribution on the pair of the response r and the successor state (s', t') that is determined in one of two ways:

1) If q is of the form *Left* q', i.e., a query q' to the core part, then the response $r =$ *Left* r' and the core's successor state s' are determined by $cf_{def}(s, x')$ and $t' = t$.
2) If q is of the form *Right* q', i.e., a query q' to the remainder part, then $rf_{def}(t, x')$ determines the response $r =$ *Right* r', the remainder's successor state t', and a list of events es from which we calculate the core's successor state $s' = \mathit{foldl}(cp_{def}, s, es)$.

The Fusing function provides a failure mechanism too. Every probability distribution in this paper 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 III-B1. Let $\mathbf{.}$ denote the operator for accessing record fields. For core and remainder records cr and rm mentioned above, $\mathit{FUSE}(cr, rm) = \mathit{RES}(\mathit{fuse}(cr, rm), (cr\mathbf{.}\mathit{cinit}, rm\mathbf{.}\mathit{rinit}))$ is a fused resource of type $\mathbb{R}(\mathfrak{q}_c + \mathfrak{q}_r, \mathfrak{r}_c + \mathfrak{r}_r)$. We denote the type of such fused resource as $\mathbb{F}(\mathfrak{q}_c, \mathfrak{q}_r, \mathfrak{r}_c, \mathfrak{r}_r)$ to distinguish it from arbitrary resources of the same type.

Core records suffice to describe the behavior that is common to every instantiation of an FRT. Core and remainder records have long type signatures, so let $\boxed{c}$ and $\boxed{r}$ respectively denote the types of cr and rm records mentioned above. Given a core $cr : \boxed{c}$, an FRT $\{\!\!\{cr\}\!\!\} : \boxed{r} \Rightarrow \mathbb{F}(\mathfrak{q}_c, \mathfrak{q}_r, \mathfrak{r}_c, \mathfrak{r}_r)$ is a function from remainder records to fused resources such that

$$\{\!\!\{cr\}\!\!\}(rm) = \mathit{FUSE}(cr, rm). \tag{2}$$

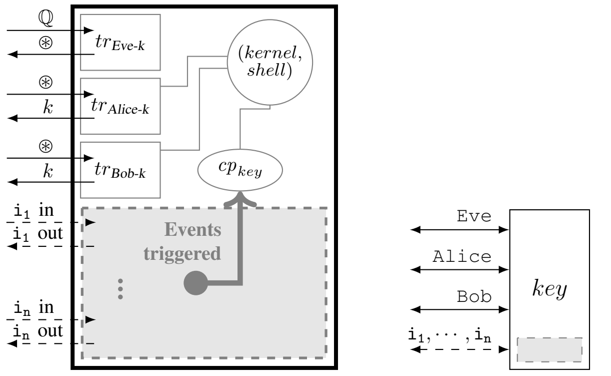
When essential (cf. the end of Sec. III), we specify the type of suitable remainder records using a subscript $\{\!\!\{cr\}\!\!\}_{\boxed{r}}$.

3) A Concrete Example: We now explain how FRTs enable modeling 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 where two parties Alice and Bob execute a protocol to share a symmetric key. The core record $key = (\!|\mathit{cinit} := ci_{key}, \mathit{cpoke} := cp_{key}, \mathit{cfunc} := cf_{key}|\!)$ consists of three fields. 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 $shell \subseteq \{\mathtt{Alice}, \mathtt{Bob}\}$ keeps track of the enabled interfaces, where "disabled" means the interface would return $\perp$ if queried. The initial state ci_{key} is $(\mathit{Void}, \{\})$, 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 $cf_{key} = tr_{\mathit{Eve\text{-}k}} +_{\mathbb{O}} tr_{\mathit{Alice\text{-}k}} +_{\mathbb{O}} tr_{\mathit{Bob\text{-}k}}$ describes the interfaces for the adversary Eve, Alice, and Bob, which are composed using the oracle composition operator. The adversary interface's functionality $tr_{\mathit{Eve\text{-}k}}$ leaks no information to Eve; it is a dummy oracle that does not change the state and answers every input with a unit value $\circledast$. The interface functionality for Alice is defined using $tr_{\mathit{Alice\text{-}k}}$, which outputs a key k upon a unit input $\circledast$ if the state is a pair $(\mathit{Hold}\ k, shell)$ such that $\mathtt{Alice} \in shell$.



(a) Internal construction (b) Abstract representation

Fig. 2: Formalizing the ideal specification for keys. The Remainder part is shown as a gray area that accepts various instantiations. Remainder interfaces are shown using dashed arrows. To simplify the diagrams, we often merge multiple interfaces into a single arrow like $\mathtt{i_1}, \cdots, \mathtt{i_n}$ above.

Queries to $tr_{\mathit{Alice\text{-}k}}$ will invoke the failure mechanism if the state does not satisfy the aforementioned conditions. Bob's interface functionality $tr_{\mathit{Bob\text{-}k}}$ is the same as Alice's interface except that it checks for $\mathtt{Bob} \in shell$ upon receiving queries.

Figure 2 depicts the ideal specification for keys. $\{\!\!\{key\}\!\!\}$ represents all fused resources that accept queries on $\mathtt{Alice}$ and $\mathtt{Bob}$ interfaces after one or more queries to the $\mathtt{i_1}, \cdots, \mathtt{i_n}$ 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 important that their corresponding interface has been enabled during one of the event triggers.

C. 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 II-C. 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. Our formalization reuses the converters and resources from Lochbihler et al. [16].

1) A Short Recap of Compound Resources: Converters are probabilistic reactive systems that internally interact with other reactive systems. A converter has two types (or classes) of interfaces. First, the *external interfaces* are used to receive inputs and send outputs. Second, computing an output may invoke many interactions on the *internal interfaces*. A converter appears like a resource when one of its external interfaces is queried, however, such a query may invoke other converters and resources prior to its response. A converter's internal interface is formalized as a CryptHOL [3] *Generative Probabilistic Value* (GPV) $\mathbb{G}(\mathfrak{a}, \mathfrak{q}, \mathfrak{r})$: a GPV produces answers of type $\mathfrak{a}$ by

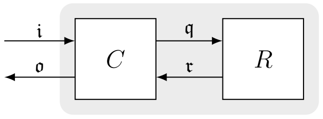


Fig. 3: Attaching a converter to a resource.

interacting with arbitrary reactive systems that accept queries of type $\mathfrak{q}$ and produce responses of type $\mathfrak{r}$. Formally,

$$\textbf{codatatype } \mathbb{G}(\mathfrak{a},\mathfrak{q},\mathfrak{r}) = \textit{GPV } \mathbb{D}(\mathfrak{a} + (\mathfrak{q} \times (\mathfrak{r} \Rightarrow \mathbb{G}(\mathfrak{a},\mathfrak{q},\mathfrak{r})))).$$

Like for resources, the co-datatype unfolds a transition function $tr : \mathfrak{t} \Rightarrow \mathbb{D}(\mathfrak{a} + (\mathfrak{q} \times (\mathfrak{r} \Rightarrow \mathfrak{t})))$ into a probabilistic tree starting with some initial state $t_0 : \mathfrak{t}$. The difference is that a GPV does not wait for an input; it consists of a distribution over answers and pairs of a query and a family of subtrees that capture the reaction to the response.

A converter then is modeled using an initial state $s_0 : \mathfrak{s}$ and a transition function $tr : \mathfrak{s} \Rightarrow \mathfrak{i} \Rightarrow \mathbb{G}(\mathfrak{o} \times \mathfrak{s}, \mathfrak{q}, \mathfrak{r})$ that describes a GPV on the output and the new state of the converter given its current state and an input. Similar to resources, Lochbihler et al. [16] provide an operator $\textit{CNV}(tr, s_0)$ that hides a converter's internal state and represents it as a value of the following co-datatype:

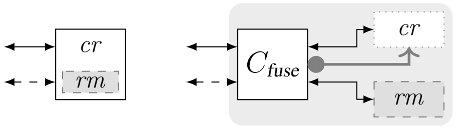
$$\textbf{codatatype } \mathbb{C}(\mathfrak{i},\mathfrak{o},\mathfrak{q},\mathfrak{r}) = \textit{Converter } (\mathfrak{i} \Rightarrow \mathbb{G}(\mathfrak{o} \times \mathbb{C}(\mathfrak{i},\mathfrak{o},\mathfrak{q},\mathfrak{r}), \mathfrak{q}, \mathfrak{r})).$$

As shown in Figure 3, attaching $C : \mathbb{C}(\mathfrak{i},\mathfrak{o},\mathfrak{q},\mathfrak{r})$ to a resource $R : \mathbb{R}(\mathfrak{q},\mathfrak{r})$, which means R responds to C's internal queries, creates a new resource $C \rhd R : \mathbb{R}(\mathfrak{i},\mathfrak{o})$, which is represented as the surrounding gray rectangle.

Lochbihler et al. model multiple interfaces $i_1, \cdots, i_n$, which answer queries of types $\mathfrak{q}_1, \cdots, \mathfrak{q}_n$ with responses of types $\mathfrak{r}_1, \cdots, \mathfrak{r}_n$, using disjoint unions as a single interface with inputs of type $\mathfrak{q}_1 + \cdots + \mathfrak{q}_n$ and outputs of type $\mathfrak{r}_1 + \cdots + \mathfrak{r}_n$. So interfaces are identified by their position in the sum type. The attachment of (multiple) converters to a subset of (simultaneously available) resources' interfaces are defined in terms of the following operators:

- Let $R_1 : \mathbb{R}(\mathfrak{q}_1,\mathfrak{r}_1)$ and $R_2 : \mathbb{R}(\mathfrak{q}_2,\mathfrak{r}_2)$ be resources. Their parallel composition $R_1 \,||\, R_2 : \mathbb{R}(\mathfrak{q}_1 + \mathfrak{q}_2, \mathfrak{r}_1 + \mathfrak{r}_2)$ directs queries $\mathfrak{q}_1$ to R_1 and $\mathfrak{q}_2$ to R_2 and forwards the responses accordingly.
- Let $C_1 : \mathbb{C}(\mathfrak{i}_1,\mathfrak{o}_1,\mathfrak{q}_1,\mathfrak{r}_1)$ and $C_2 : \mathbb{C}(\mathfrak{i}_2,\mathfrak{o}_2,\mathfrak{q}_2,\mathfrak{r}_2)$ be two converters. Their parallel composition $C_1 \mid C_2 : \mathbb{C}(\mathfrak{i}_1 + \mathfrak{i}_2, \mathfrak{o}_1 + \mathfrak{o}_2, \mathfrak{q}_1 + \mathfrak{q}_2, \mathfrak{r}_1 + \mathfrak{r}_2)$ makes both converters available at the same time, analogous to parallel resource composition.
- Let $C_1 : \mathbb{C}(\mathfrak{i},\mathfrak{o},\mathfrak{m},\mathfrak{n})$ and $C_2 : \mathbb{C}(\mathfrak{m},\mathfrak{n},\mathfrak{q},\mathfrak{r})$ be two converters. Their sequential composition $C_1 \odot C_2 : \mathbb{C}(\mathfrak{i},\mathfrak{o},\mathfrak{q},\mathfrak{r})$ uses C_2 to answer C_1's queries on C_1's internal interface.
- The identity converter $\mathbb{1} : \mathbb{C}(\mathfrak{q},\mathfrak{r},\mathfrak{q},\mathfrak{r})$ simply forwards all queries and responses from the external to the internal interface and vice versa.

We provide an example to clarify how the above operators are used. Consider resource R_1 with one interface $\mathtt{x}$, resource R_2 with two interfaces $\mathtt{y}$ and $\mathtt{z}$, and a converter C with an external interface $\mathtt{e}$ and two internal interfaces $\mathtt{a}$ and $\mathtt{b}$. The compound resource $((\mathbb{1} \mid C) \odot C_W) \rhd (R_1 \,||\, R_2)$ results from attaching $\mathtt{a}$ and $\mathtt{b}$ to $\mathtt{z}$ and $\mathtt{x}$ respectively, where C_W is defined in terms of special converters called *wirings* [16] that transpose $\mathtt{x} + (\mathtt{y} + \mathtt{z})$ to $\mathtt{y} + (\mathtt{z} + \mathtt{x})$[4]. To avoid boilerplate notation and operators, we often describe converter attachments using interface names similar to Section II-C. Moreover, we will drop the distinction between the terms "converter" and "protocol" from now on since multiple converter and resource attachments are encoded as a single attachment written in terms of composition operators.



(a) (b)

Fig. 4: Fusing core and remainder resources. The two representations of fused resources in 4a and 4b are the same.

2) Compound Fused Resources and FRTs: The aforementioned operators do not preserve the structure of fused resources. For instance, given the fused resource $F : \mathbb{F}(\mathfrak{q}_c, \mathfrak{q}_r, \mathfrak{r}_c, \mathfrak{r}_r)$ and the converter $C : \mathbb{C}(\mathfrak{q}_r + \mathfrak{q}_c, \mathfrak{r}_r + \mathfrak{r}_c, \mathfrak{q}_c + \mathfrak{q}_r, \mathfrak{r}_c + \mathfrak{r}_r)$ 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 IV-A, 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.

The core and remainder parts can be understood as resources of their own: A core part cr is like an oracle with two interfaces $cr\textbf{.}\textit{cpoke} +_{\mathbb{O}} cr\textbf{.}\textit{cfunc}$, where $cr\textbf{.}\textit{cpoke}$ conceptually responds to every event with $\circledast$. Similarly, $rm\textbf{.}\textit{rfunc}$ is the oracle version of the remainder part rm. Both can be converted into resources using *RES*. Under this view, we can compose core and remainder parts in parallel (notations $cr_1 \|_c cr_2$ and $rm_2 \|_r rm_2$ respectively) using the parallel composition of resources. Similarly, we can attach a converter to a core and remainder (notation $\rhd_c$ and $\rhd_r$). Moreover, as Figure 4 depicts, the fusing of cr and rm can be expressed by attaching a suitable fuse converter $C_{\textit{fuse}}$ to the parallel composition of the core and remainder resources.

The next four lemmas show how parallel composition and converter attachment can be pushed through the fuse function. The lemmas refer to the core records $cr_i : \boxed{c}_i$, the remainder records $rm_i : \boxed{r}_i$, and the fused resources $F_i = \textit{FUSE}(cr_i, rm_i)$ for $i \in \{1, 2\}$. The first lemma explains the simultaneous access to fused resources. and is depicted in Figure 5.

Lemma 1. *For $F_i : \mathbb{F}(\mathfrak{q}_c^i, \mathfrak{q}_r^i, \mathfrak{r}_c^i, \mathfrak{r}_r^i)$, consider the fused resource $F : \mathbb{F}(\mathfrak{q}_c^1 + \mathfrak{q}_c^2, \mathfrak{q}_r^1 + \mathfrak{q}_r^2, \mathfrak{r}_c^1 + \mathfrak{r}_c^2, \mathfrak{r}_r^1 + \mathfrak{r}_r^2)$ defined as $F =$*

[4]The overloaded $+$ operator on interface names is only used to indicate that disjoint unions are not associative and wirings enable interface reordering.

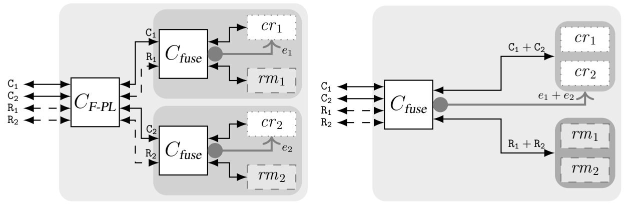


Fig. 5: Parallel composition of fused resources. The two resources in 5a and 5b are equal. The rounded rectangle surrounding cr_1 and cr_2 represents $cr_1 \|_c cr_2$ and the gray area around rm_1 and rm_2 represents $rm_1 \|_r rm_2$.

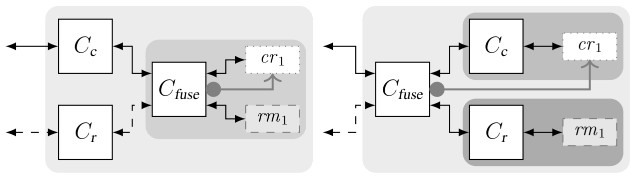


Fig. 6: Attaching converters to core and remainder interfaces. The two resources in 6a and 6b are equal. The rounded rectangle surrounding C_c and cr_1 represents cr_2 and the gray area around C_r and rm_1 represents rm_2.

FUSE$(cr_1 \|_c cr_2, rm_1 \|_r rm_2)$. *Then* $F = C_{\textit{F-PL}} \rhd (F_1 \parallel F_2)$, *where* $C_{\textit{F-PL}}$ *is the wiring that groups the core and remainder interfaces of* F_1 *and* F_2 *together.*

Lemma 2. *For* $F_1 : \mathbb{F}(\mathfrak{q}_c, \mathfrak{q}_r, \mathfrak{r}_c, \mathfrak{r}_r)$ *and* $F_2 : \mathbb{F}(\mathfrak{i}_c, \mathfrak{i}_r, \mathfrak{o}_c, \mathfrak{o}_r)$, *consider the converters* $C_c : \mathbb{C}(\mathfrak{i}_c, \mathfrak{o}_c, \mathfrak{q}_c, \mathfrak{r}_c)$ *and* $C_r : \mathbb{C}(\mathfrak{i}_r, \mathfrak{o}_r, \mathfrak{q}_r, \mathfrak{r}_r)$. *Then* $F_2 = (C_c \mid C_r) \rhd F_1$ *if* $cr_2 = C_c \rhd_c cr_1$ *and* $rm_2 = C_r \rhd_r rm_1$, *where* $\rhd_c$ *attaches a converter to the* cfunc *oracle of a core and* $\rhd_r$ *does so for the* rfunc *oracle of a remainder part.*

Figure 6 depicts what is described in Lemma 2. In subfigure 6a, the converters C_c and C_r attach to the core and remainder interfaces of the fused resource $F_1 =$ *FUSE*(cr_1, rm_1). In subfigure 6b, the two converters C_c and C_r directly attach to the core cr_1 and remainder rm_1, and the two resulting parts are fused only thereafter using $C_{\textit{fuse}}$; note that $C_{\textit{fuse}}$ sends the events from the remainder rm_1 directly to cr_1, bypassing the attached converters. Lemma 2 states that the two are equivalent. So we can push converters of the core or remainder interfaces through the fusing. In particular, we can set C_c or C_r to the identity converter $\mathbb{1}$ if we have just one converter for the remainder or core interface, respectively.

The next two lemmas determine if a converter's simultaneous attachment to core and remainder 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 remainder to core interfaces of fused resources.

Lemma 3. *For* $F_1 : \mathbb{F}(\mathfrak{i} + \mathfrak{q}_c, \mathfrak{q}_r, \mathfrak{o} + \mathfrak{r}_c, \mathfrak{r}_r)$, *let* `I` *refer to the left-most (w.l.o.g.) core interface. Assume that* `I`*'s* $\mathfrak{o}$ *responses do not depend on* $\mathfrak{q}_c$ *queries, and let* cr_2 *and* rm_2 *be the records that result from removing* `I`*'s functionality from* cr_1 *and adding it to* rm_1. *Then* $(F_2 : \mathbb{F}(\mathfrak{q}_c, \mathfrak{i} + \mathfrak{q}_\mathfrak{r}, \mathfrak{r}_c, \mathfrak{o} + \mathfrak{r}_r)) = C_{\textit{F-CR}} \rhd F_1$, *where* $C_{\textit{F-CR}}$ *is the wiring converter that reorders the interface positions accordingly.*

Lemma 4. *For* $F_1 : \mathbb{F}(\mathfrak{q}_c, \mathfrak{i} + \mathfrak{q}_r, \mathfrak{r}_c, \mathfrak{o} + \mathfrak{r}_r)$, *let* `I` *refer to the left-most (w.l.o.g.) remainder interface. Assume that* $\mathfrak{r}_r$ *responses do not depend on* `I`*'s* $\mathfrak{i}$ *queries, and let* rm_2 *and* cr_2 *be the records that result from removing* `I`*'s functionality from* rm_1 *and adding it to* cr_1. *Then* $(F_2 : \mathbb{F}(\mathfrak{i} + \mathfrak{q}_c, \mathfrak{q}_r, \mathfrak{o} + \mathfrak{r}_c, \mathfrak{r}_r)) = C_{\textit{F-RC}} \rhd F_1$, *where* $C_{\textit{F-RC}}$ *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 $C \lBrace cr \rBrace_{\boxed{T}}$ for the FRT that consists of the fused resources $C \rhd \lBrace cr \rBrace(rm)$ for $rm \in \boxed{T}$. This notation is analogous to Section II-C where $\pi\mathcal{R}$ represented the attachment of the protocol π to the specification $\mathcal{R}$.

For Lemma 1, the resulting FRT $\lBrace cr_1 \|_c cr_2 \rBrace_{\boxed{T}_1 \times \boxed{T}_2}$ is indexed over $\boxed{T}_1 \times \boxed{T}_2 = \{rm_1 \|_r rm_2 \mid rs_1 \in \boxed{T}_1 \wedge rm_2 \in \boxed{T}_2\}$. This set does not include all remainder 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 remainder records in $\boxed{T}_1 \times \boxed{T}_2$ can be split into two independent components, one for each of the fixed remainder records rm_1 and rm_2; however such a decomposition is not possible in general.

IV. Secure Constructions with FRTs

As explained in Section II-C, 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

$$\forall R \in \mathcal{R}.\ \exists S \in \mathcal{S}.\ \mathfrak{d}(\pi R, \sigma S) \leq \epsilon. \tag{3}$$

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 $R \in \mathcal{R}$. Hence, $\mathcal{R}$ 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 remainder record hides the details of the party activation patterns.

First, we specialize (3) to the case where $\mathcal{R}$ and $\mathcal{S}$ are FRTs. We only define the information-theoretic security here; the computational security notion is defined analogously.

Definition 1 (Information-theoretic concrete security)**.** *Let* $\lBrace \textit{real} \rBrace_{\boxed{T}_R}$ *and* $\lBrace \textit{ideal} \rBrace_{\boxed{T}_I}$ *be FRTs and let* π *be the protocol, i.e., a converter that attaches to the core user interface of* $\lBrace \textit{real} \rBrace$. *Then* $\pi\lBrace \textit{real} \rBrace$ *is an information-theoretically* ϵ*-secure realization of the ideal specification* $\lBrace \textit{ideal} \rBrace$ *if there exist a simulator*

σ that attaches to the adversary interface of core and remainder, and a remainder embedding function $f : \boxed{T}_R \Rightarrow \boxed{T}_I$ such that

$$\mathfrak{d}(\pi \rhd \{\!|\mathit{real}|\!\}(rm), \sigma \rhd \{\!|\mathit{ideal}|\!\}(f(rm))) \leq \epsilon \qquad (4)$$

for all $rm \in \boxed{T}_R$.

Note that the remainder embedding f in (4) skolemizes the existential $\exists S \in \mathcal{S}$ in (3). This works thanks to the shape of FRTs, namely both $\mathcal{R}$ and $\mathcal{S}$ are parametrized by the remainder records.

We extend this notion to asymptotic security by introducing a security parameter η and requiring that ϵ is negligible in η. As is customary in CryptHOL [3], 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.

A. Proving a Protocol Secure

Proving a protocol π secure boils down to establishing a bound on the advantage of a distinguisher. This is typically done as a sequence of game transformations using the existing tool set from CryptHOL [3], [14], similar to Lochbihler et al.'s formalization of CC [16]. We now explain the common pattern for FRTs.

Lochbihler et al. [16] have already formalized the distinguishability bound $\mathfrak{d}(R, S)$ on two concrete resources R and S using explicit distinguishers. Formally, a distinguisher $D : \mathbb{G}(\mathbb{B}, \mathfrak{q}, \mathfrak{r})$ is a GPV that returns a Boolean $\mathbb{B}$ after having interacted with a resource through queries $\mathfrak{q}$ and responses $\mathfrak{r}$. We write $D \blacktriangleright R : \mathbb{D}(\mathbb{B})$ for the resulting probability distribution over Booleans $\mathbb{B}$. D's advantage $\mathit{adv}(D, R, S)$ of distinguishing R and S is then given by

$$\mathit{adv}(D, R, S) = \left| Pr_{(D \blacktriangleright R)}[\mathit{True}] - Pr_{(D \blacktriangleright S)}[\mathit{True}] \right|,$$

where $Pr_p[x]$ denotes the probability that the distribution p assigns to the elementary event x. Then $\mathfrak{d}(R, S) \leq \epsilon$ iff $\mathit{adv}(D, R, R') \leq \epsilon$ for all distinguishers D. Here, D ranges over all distinguishers in the information-theoretic setting and over computationally-bounded distinguishers in the computational setting.

The proof of (4) must work for all $rm \in \boxed{T}_R$. It should therefore concentrate on the cores *real* and *ideal* of the fused resources $\{\!|\mathit{real}|\!\}(rm)$ and $\{\!|\mathit{ideal}|\!\}(f(rm))$. 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 $\pi \rhd \{\!|\mathit{real}|\!\}(rm)$, Lemma 2 shows that the converter π can be attached directly to *real* rather than the fused resource, because π attaches only to the core interface. This transformed core *real′* yields a transformed FRT $\{\!|\mathit{real}'|\!\}_{\boxed{T}_R}$.

On the ideal side, however, we cannot directly use Lemma 2 to push the simulator σ through *fuse* because σ attaches to the adversary core and remainder interfaces. This way, the simulator can create a flow of information from the core part *ideal* to the remainder part $f(rm)$, which cannot happen in a fused resource. We now exploit that the simulator must work for all rm in the same way, in particular for the remainder record that does not provide any interfaces at all. Accordingly, σ uses only the interfaces that f adds on top of rm's. By a similar argument, $f(rm)$ answers queries on these additional interfaces without consulting rm itself. Moreover, f may translate poke events from rm to poke events for the ideal core and this translation is independent of rm's state.[5] So we can move these additional interfaces to *ideal* by Lemma 4. Then 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 $\{\!|\mathit{ideal}'|\!\}_{\boxed{T}_R}$.

In summary, it suffices to prove

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

for all D. Now the fused resources $\{\!|\mathit{real}'|\!\}(rm)$ and $\{\!|\mathit{ideal}'|\!\}(rm)$ use the same remainder record rm. Since the core and remainder 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

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

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 the literature [16].

B. Trace Equivalence Up-To

When proving indistinguishability between resources, we frequently must transform the resource into a particular shape, e.g., to prepare for the application of a reduction argument. Lochbihler et al. [16] found that these steps can often be justified by trace equivalence. They gave the following bisimulation-style proof rule for establishing trace equivalence of resources. We now provide a new up-to proof rule for trace equivalence and explain how it simplifies such proofs.

Let $\mathit{dirac}(x)$ be the one-point distribution on x, and $\overline{\mathit{run}}(p, a)$ denote the weighted combination of running a resource from the distribution p of resources with query a, and $p\rceil_x$ condition a distribution p over pairs $\mathfrak{a} \times \mathfrak{b}$ on the elementary event $x : \mathfrak{a}$.

Theorem 1 ([16, Thm. 1])**.** *Two resources R and S are trace equivalent iff there exists a relation X between distributions of resources such that*

1) *X relates $\mathsf{dirac}(R)$ to $\mathsf{dirac}(S)$; and*
2) *Whenever $(p, q) \in X$, then for all queries a, $\overline{\mathsf{run}}(p, a)$ and $\overline{\mathsf{run}}(q, a)$ have the same marginal distribution on*

[5]This argument can be made precise in the computational setting: The skolemization f in (4) of the existential $\exists S \in \mathcal{S}$ in (3) ensures that f chooses the remainder records in $\{\!|\mathit{ideal}|\!\}$ uniformly. Uniformity can be captured by f being relationally parametric in the queries and responses of rm. So we can decompose $f(rm)$ into two parts with disjoint state: (i) rm and (ii) additional independent interfaces and a translator for poke events coming from rm.

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. We have not formalized the above parametricity argument in Isabelle/HOL because CryptHOL uses HOL's function space, which does not have a computational interpretation.

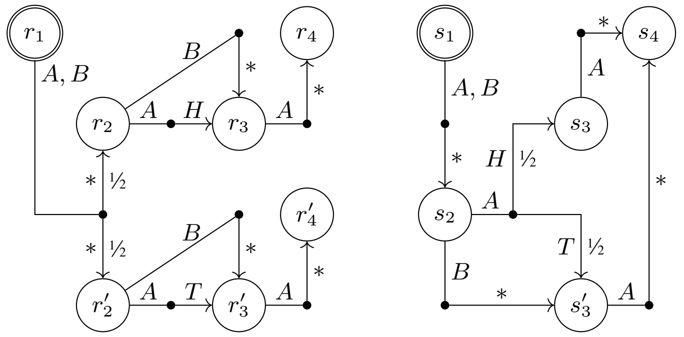


Fig. 7: Two trace-equivalent resources that respond with a fair coin flip (H or T) if the second query is A.

row	r_1	r_2	r_2'	r_3	r_3'	r_4	r_4'	s_1	s_2	s_3	s_3'	s_4	justification
#1	1							1					initialization
#2		½	½						1				#1: $A, B \bullet *$
#3				1						1			#2: $A \bullet H$
#4					1						1		#2: $A \bullet T$
#5				½	½						1		#2: $B \bullet *$
#6						1						1	#3: $A \bullet *$
#7							1					1	#4: $A \bullet *$
#8						½	½					1	#5: $A \bullet *$
#9													invalid query
#10				1							1		strengthening

TABLE I: The smallest relation X (rows #1 to #9 only) to prove the resources in Fig. 7 trace-equivalent using Thm. 1. Every row describes two subprobability distributions, one over $r_1, \ldots, r_4'$ and one over $s_1, \ldots, s_4$, and X relates the former to the latter. Empty cells contain the probability mass 0.

responses and $(\overline{run}(p, a)\upharpoonright_b, \overline{run}(q, a)\upharpoonright_b) \in X$ for each possible response b to the query a.

The distributions in the relation X capture the so-far unobserved probabilistic choices of the resources. For example, suppose that one resource R eagerly samples a random value at the start whereas the other resource S delays the probabilistic choice until it is actually needed. So there is a time interval when R has already sampled the value, but not S, i.e., the chosen value has not yet been observed through the interactions with R. During this time, the relation X conserves this unobserved randomness in a distribution on S's side. When S finally samples the value and uses it to compute a response, this conserved randomness is used in the proof to justify the indistinguishability of the responses.

Figure 7 shows two such resources R (left) and S (right) with initial states r_1 and s_1, respectively. The arrows denote the transitions upon a query (A or B; before the dot) and the responses (H, T, or $*$; after the dot) along with the probability mass of the response; a probability mass of 1 is omitted. Both resources accept the query sequences A, A, A and A, B, A and B, A, A and B, B, A. When the second query is A, they respond with a fair coin flip (H and T for heads and tails). All other queries are merely acknowledged using the response $*$. The resource S flips the coin lazily upon the second query whereas R eagerly flips the coin upon the first query and remembers the outcome in the states r_2 and r_2'. The two resources are thus *not* bisimilar as no single state in R can mimick s_2's behavior of responding to an A query with H or L uniformly at random.

A suitable (and minimal) relation X for Thm. 1 can be found as follows:

1) Initialize X to relate the one-point distributions $\mathit{dirac}(r_1)$ and $\mathit{dirac}(s_1)$ for the initial states.
2) For each pair $(p, q) \in X$ and every possible query a and response b, add $(\overline{run}(p, a)\upharpoonright_b, \overline{run}(q, a)\upharpoonright_b)$ to X. Repeat this step until no new pairs are added to X.

Table I illustrates this process for the resources in Fig. 7, where we use the initial states to refer to the resources. The notation #i: $a \bullet b$ in the last column means that this Row is added by Step 2 for query a and response b due to X already containing the Row #i. Row #1 represents the initialization Step 1. Row #2 originates from Row #1 through the queries A or B with response $*$. The uniform distribution over r_2 and r_2' in #2 conserves the eager coin flip's unobserved randomness, so that the next A query's marginal distribution on responses is uniform over H and T, as required by Thm. 1(2). Conditioning on each response of this query exposes the conserved randomness and leads to the Rows #3 and #4. The query B in Row #2 produces only a single response $*$, which does not expose the randomness. Accordingly, #5 contains the uniform distribution over r_3 and r_3'. In Rows #3 to #5, a query A yields the response $*$, which adds Rows #6 to #8. The query B is invalid in states r_3, r_3', s_3, and s_3', i.e., Rows #3 to #5; technically, such a query results in a failure, which we represent by the empty subprobability distribution in #9. No queries are valid in Rows #6 to #9 and X already contains Row #9 for such failures. So we have found a minimal relation X for Thm. 1, given by Rows #1 to #9. We discuss Row #10 later in this section.

As the B query in Row #2 shows, conserved randomness may not be exposed in some interactions with a resource. 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 (Row #5). In practice, 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 resulting up-to trace equivalence proof rule is similar to Sangiorgi's bisimulation up-to rules [22], except that our rule is about trace equivalence. The closure $[\![X]\!]$ of a relation X of distributions is defined inductively as follows, where $p \ggg f$ denotes the p-weighted combination of a family

f of distributions, i.e., $Pr_{(p \gg= f)}[x] = \sum_y Pr_p[y] \cdot Pr_{f(y)}[x]$.

1) Whenever $(p, q) \in X$, then $(p, q) \in [\![X]\!]$.
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 up-to)**.** *Two resources R and S are trace equivalent iff there exists a relation X between distributions of resources such that*

1) *X relates $\mathsf{dirac}(R)$ to $\mathsf{dirac}(S)$.*
2) *Whenever $(p, q) \in X$, then for all queries a, $\overline{\mathsf{run}}(p, a)$ and $\overline{\mathsf{run}}(q, a)$ have the same marginal distribution on responses and $(\overline{\mathsf{run}}(p, a)\upharpoonright_b, \overline{\mathsf{run}}(q, a)\upharpoonright_b) \in [\![X]\!]$ for each possible response b to the query a.*

For a trace equivalence proof, this theorem can significantly reduce the number of cases in the relation X. For the resources in Fig. 7, Row #8 in Table I is a weighted combination of Rows #6 and #7. Similarly, Row #9 is the 0-weight scaling of any other Row and thus redundant too. So these two Rows are redundant because the closure operator $[\![X]\!]$ adds them for free.

Moreover, the closure operator $[\![\_]\!]$ allows us to simplify some cases by strengthening them. In Table I, Row #5 requires us to prove that the uniform distribution over resources r_3 and r_3' is trace equivalent to s_3'. Yet, using our knowledge of the system, the stronger statement holds that the resources r_3 and r_3' each are trace equivalent to s_3', as expressed by Rows #4 and #10. So we can deliberately add Row #10 to our relation X. Row #5 then is a weighted combination of Rows #4 and #10 and therefore obsolete by the up-to closure. Row #10 simplifies the proof compared to Row #5 in two ways. First, reasoning about the one-point distribution $\mathit{dirac}(r_3)$ is simpler than about the uniform distribution over r_3 and r_3'. Second, applying Step 2 to Row #10 directly yields Row #7 for query A and response $*$. So Row #8 is never added to X in the first place, i.e., no redundancy argument is needed.

In summary, Rows #1 to #4, #6, #7, and either #10 or #5 suffice for proving trace equivalence using Thm. 2. In this artificial example, the savings are modest because Rows #7 to #9 do not incur further pairs in X and no queries are valid in those Rows. The savings are much higher in our case studies though.

C. Composability

We now show that our security definition 1 yields composable security statements. In CC, a protocol π typically uses multiple resources $R_1, \ldots, R_n$ to create a resource $\pi \rhd (R_1 \| \ldots \| R_n)$, which should be hard to distinguish from a simulated ideal resource $\sigma \rhd S$. 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.

We first give composability theorems for concrete security statements of individual resources and then use them to establish composability for FRTs. As before, we focus on the information-theoretic setting; the computational setting is analogous by restricting the class of distinguishers to computationally bounded ones.

Definition 2 (Concrete security for individual resources)**.** *A resource R ϵ-securely realizes a resource S if there is a converter σ that attaches to S's adversary interfaces such that*

$$\mathsf{adv}(D, R, \sigma \rhd S) \leq \epsilon$$

for all distinguishers D.

Clearly, if $\pi\{\!|\mathit{real}|\!\}_{\boxed{T}_R}$ ϵ-securely realizes the FRT $\{\!|\mathit{ideal}|\!\}_{\boxed{T}_I}$ (Def. 1), then $\pi \rhd \{\!|\mathit{real}|\!\}(rm)$ ϵ-securely realizes the fused resource $\{\!|\mathit{ideal}|\!\}(f(rm))$ in the sense of Def. 2 for all $rm \in \boxed{T}_R$ as witnessed by the simulator σ.

Secure realization between individual instances is compositional. The theorem below generalizes the asymptotic security statements from [16] to the concrete setting.

Theorem 3 (Composability for individual resources)**.**

1) *Every resource R 0-securely realizes itself.*
2) *If R_1 ϵ_1-securely realizes S_1 as witnessed by σ_1 and R_2 ϵ_2-securely realizes S_2 as witnessed by σ_2, then $R_1 \| R_2$ $(\epsilon_1 + \epsilon_2)$-securely realizes $S_1 \| S_2$ as witnessed by $\sigma_1 | \sigma_2$.*
3) *If R ϵ-securely realizes S as witnessed by σ and S ϵ'-securely realizes T as witnessed by τ, then R $(\epsilon + \epsilon')$-securely realizes T as witnessed by $\tau \odot \sigma$.*
4) *If R ϵ-securely realizes S as witnessed by σ and C is a converter that attaches to R's user interface, then $C \rhd R$ ϵ-securely realizes $C \rhd S$ as witnessed by σ.*

These composability results suffice to show that secure realization between FRTs is also compositional. As a FRT $\{\!|cr|\!\}_{\boxed{T}}$ is a family of fused resources $\{\!|cr|\!\}(rm)$ for $rm \in \boxed{T}$, we obtain composability by lifting Thm. 3 pointwise to specifications. Note that the simulator and the remainder embeddings remain independent of the chosen fused resources, as required by Def. 1.

Theorem 4 (Composability for FRTs)**.**

1) *The identity protocol $\mathbb{1}$ applied to a FRT $\{\!|cr|\!\}_{\boxed{T}}$ 0-securely realizes the FRT $\{\!|cr|\!\}_{\boxed{T}'}$ for $\boxed{T}' \supseteq \boxed{T}$ with the simulator being the identity converter and the remainder embedding being the identity function.*
2) *Let $\pi_i\{\!|\mathit{real}_i|\!\}_{\boxed{T}_{R_i}}$ ϵ_i-securely realizes the FRT $\{\!|\mathit{ideal}_i|\!\}_{\boxed{T}_{I_i}}$ with simulator σ_i and remainder embedding f_i for $i = 1, 2$. The parallel composition*

$$(\pi_1 \mid \pi_2)\{\!|\mathit{real}_1 \|_c \mathit{real}_2|\!\}_{\boxed{T}_{R_1} \times \boxed{T}_{R_2}}$$

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

$$\{\!|\mathit{ideal}_1 \|_c \mathit{ideal}_2|\!\}_{\boxed{T}_{I_1} \times \boxed{T}_{I_2}}$$

with simulator $\sigma_1|\sigma_2$ and remainder embedding $f(rm_1, rm_2) = (f_1(rm_1), f_2(rm_2))$.

3) *Let $\pi\{\!\{\mathsf{real}\}\!\}_{\boxed{T}_R}$ ϵ-securely realize the FRT $\{\!\{\mathsf{middle}\}\!\}_{\boxed{T}_M}$ with simulator σ and remainder embedding f and let $\rho\{\!\{\mathsf{middle}\}\!\}_{\boxed{T}'_M}$ ϵ'-securely realize $\{\!\{\mathsf{ideal}\}\!\}_{\boxed{T}_I}$ with simulator τ and remainder embedding g such that $\boxed{T}_M \subseteq \boxed{T}'_M$. Then, $\pi\{\!\{\mathsf{real}\}\!\}_{\boxed{T}_R}$ $(\epsilon+\epsilon')$-securely realizes $\{\!\{\mathsf{ideal}\}\!\}_{\boxed{T}_I}$ with simulator $\tau \odot \sigma$ and remainder embedding $g \circ f$.*
4) *If $\pi\{\!\{\mathsf{real}\}\!\}_{\boxed{T}_R}$ ϵ-securely realizes the FRT $\{\!\{\mathsf{ideal}\}\!\}_{\boxed{T}_I}$ and C is a converter that attaches to the core user interface, then $C\pi\{\!\{\mathsf{real}\}\!\}_{\boxed{T}_R}$ ϵ-securely realize $C\{\!\{\mathsf{ideal}\}\!\}_{\boxed{T}_I}$ with the same simulator and remainder embedding.*

V. Case Studies

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 [15]. We start by describing the proof's main building blocks and steps in Section V-A. We then delve into the details of each step in Sections V-B to V-D.

We apply the following conventions in diagrams in Figure 8. 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 remainder record rm_1 in Figure 1a. Converters' attachment to fused Resources is represented using arrows, where solid and dashed arrows represent the attachment to core and remainder 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 "remainder" from top to bottom. For simplicity, we merge multiple remainder 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.

A. Proof's Building Blocks and High-level Overview

Our example construction combines two cryptographic primitives in Figure 8a. First, the 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 the 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 III-B3. 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 ciphertext 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 remainder 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^x, which is sent by the other party, from the authenticated channel R_{aut2} and stores $(g^x)^y$ in its state.

The proof has three central steps depicted in Fig. 8:

1) Corresponding to the dotted area in Figures 8e and 8f, we show in Section V-B 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}.
2) Corresponding to the steps from Figure 8a to 8e, we prove in Section V-C that the specification R_{r1} securely realizes the key specification R_{key}.
3) Justifying the steps from Figure 8a to 8f, the two constructions are combined using the composition theorems in Section V-D.

B. Proving the One-time-pad Construction Secure

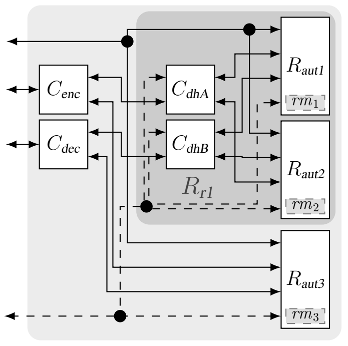
We begin with the secure realization of R_{sec} using R_{r2}. We want to show that the dotted area in Figure 8e securely realizes the dotted area in Figure 8f. We achieve this by proving the trace equivalence of the aforementioned constructions, which implies a 0-secure realization.

We prove that R_{r2} is trace equivalent to the fused resource resulting from attaching C_{sim2} to R_{sec}. The simulator C_{sim2} does not need to communicate with R_{sec}'s remainder 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 of the same length; the output is stored in C_{sim2}'s state to answer future look queries. The remainder of the adversary queries to C_{sim2}, e.g. forward or drop queries, are simply forwarded to R_{sec}. The remainder embedding $\mathcal{E}_{\mathrm{otp}}$ keeps track of the events that are received on R_{key} and R_{aut3}'s remainder interfaces to trigger the corresponding events on R_{sec}'s core.

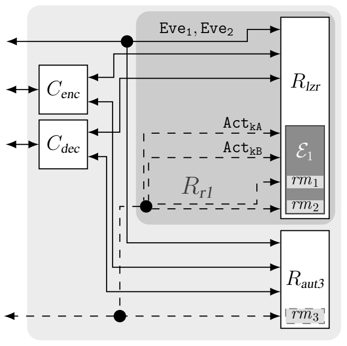
C. Proving the Diffie-Hellman Construction Secure

The secure realization of R_{key} using R_{r1} requires a proof with three major steps. These steps essentially capture the reduction to the Decisional Diffie-Hellman (DDH) game that is represented in Figures 8a to 8e.

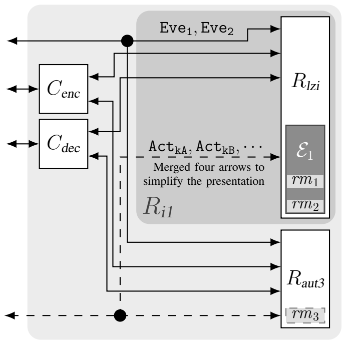
First, going from Figure 8a to Figure 8b, we make our key-exchange protocol lazy. That is, we prove that R_{r1} is trace equivalent to a fused resource R_{lzr} that postpones the half-key samplings until one of the parties requires the key. Note that the aforementioned fused resources provide the same interfaces to



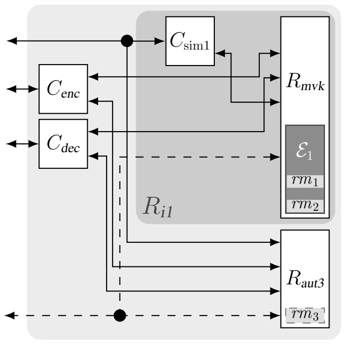
(a) Real construction


$\mathtt{Eve}_1, \mathtt{Eve}_2$
C_{enc}
C_{dec}
R_{lzr}
$\mathtt{Act}_{\mathtt{kA}}$
$\mathtt{Act}_{\mathtt{kB}}$
$\mathcal{E}_1$
rm_1
rm_2
R_{r1}
R_{aut3}
rm_3

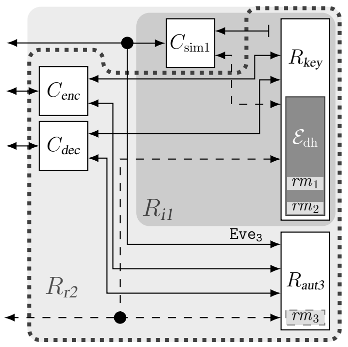
(b) Replacing lazy real key


$\mathtt{Eve}_1, \mathtt{Eve}_2$
C_{enc}
C_{dec}
R_{lzi}
$\mathtt{Act}_{\mathtt{kA}}, \mathtt{Act}_{\mathtt{kB}}, \cdots$
Merged four arrows to simplify the presentation
$\mathcal{E}_1$
rm_1
rm_2
R_{i1}
R_{aut3}
rm_3

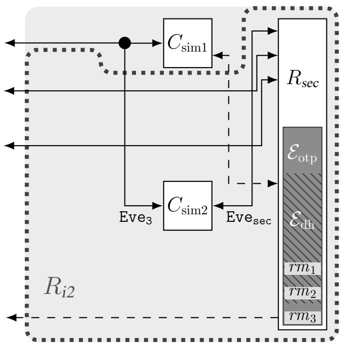
(c) Replacing DDH reduction



(d) Replacing lazy ideal key



(e) Replacing ideal key



(f) Ideal functionality

Fig. 8: Composing two security statements.

the outside world. We call C_{dhA} and C_{dhB}'s top interface $\mathtt{Act}_{\mathtt{kA}}$ and $\mathtt{Act}_{\mathtt{kB}}$ respectively; furthermore, we call R_{aut1} and R_{aut2}'s adversary interfaces $\mathtt{Eve}_1$ and $\mathtt{Eve}_2$. In the fused resource R_{r1}, the Diffie-Hellman half-keys x and y are sampled separately upon the queries to $\mathtt{Act}_{\mathtt{kA}}$ and $\mathtt{Act}_{\mathtt{kB}}$ respectively. However, R_{lzr} samples the pair (x, y) when it receives a query on any of $\mathtt{Act}_{\mathtt{kA}}$, $\mathtt{Act}_{\mathtt{kB}}$, $\mathtt{Eve}_1$, or $\mathtt{Eve}_2$. 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., $\mathtt{Act}_{\mathtt{kA}}$ and $\mathtt{Act}_{\mathtt{kB}}$, to R_{lzr}'s remainder part.

Second, corresponding to the backwards steps from Figure 8e to Figure 8c, 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 $\mathtt{Act}_{\mathtt{kA}}$, $\mathtt{Act}_{\mathtt{kB}}$, and either of $\mathtt{Eve}_1$ or $\mathtt{Eve}_2$ 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_{sim1} and $\mathcal{E}_{\text{dh}}$ using the remainder embedding $\mathcal{E}1$: it keeps track of $\mathtt{Eve}_1$ and $\mathtt{Eve}_2$'s forward queries and the events received on the remainder interfaces, e.g. $\mathtt{Act}_{\mathtt{kA}}$ and $\mathtt{Act}_{\mathtt{kB}}$, to trigger appropriate events on its core part. The proof step from Figure 8e to Figure 8d uses a corollary of Lemma 4 to move $\mathcal{E}_{\text{dh}}$ to R_{mvk}'s core part; and the proof step to Figure 8c uses Lemma 2 to reason about the attachment of the resulting specification to C_{sim1}. Note that C_{sim1} is not connected to R_{key}'s core adversary interface that answers all queries with $\circledast$. We use an unconnected arrow to signify that C_{sim1}'s behavior does not depend on the input on that interface.

Third, corresponding to the step between Figures 8b and 8c, 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.

D. Putting It All Together

We use Theorem 4 to compose the security statements in the previous sections. This enables us to replace the dotted areas in their surrounding context in Figures 8e and 8f.

In summary, we obtain the following security results. Let ϵ denote the distinguishing advantage for the decisional Diffie-Hellman game and consider the protocol $\pi = \mathbb{1}|((C_{\text{enc}} \odot C_{dhA})| (C_{\text{dec}} \odot C_{dhB}))$. Then, $\pi\{\!|(aut1\|_c aut2)\|_c aut3|\!\}$ ϵ-securely realizes the specification $\{\!|sec|\!\}$ with the simulator $(C_{sim1}|C_{sim2})|\mathbb{1}$ and the remainder embedding $\mathcal{E}_{\text{otp}}(\mathcal{E}_{\text{dh}}(rm_1\|_r rm_2)\|_r rm_3)$.

As is standard, the above results depend on the correctness of the Isabelle infrastructure. Namely the correctness of Isabelle's small kernel and the ML compiler that produces its binary. Furthermore, one should examine all the definitions to ensure they match with the security claim.

VI. Discussion and Related Work

In this section, we discuss how our formalization relates to the existing approaches to composable security arguments and compare it with relevant formalization results.

Our result is an instance of CC. As mentioned in Section II-C, CC is a theory that enables the modular reasoning about system classes, i.e., the shared behavior of similar systems. FRTs introduce a formal approach to abstract over the system communication patterns in a family of systems that exhibit similar behavior.

Our work is the first that formulates requirements for system communication modeling and provides a rigorous solution as an instance of CC. In Section III-A, we explained the importance of such a solution for the reusability of security proofs. The pen-and-paper presentations of existing CC results [12], [17], [18], [19] 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.

In comparison to the simulation-based frameworks [1], [7], [11], [13], 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' remainder part plays in our definition of secure constructions and their composition. We provide composition theorems that consider arbitrary remainders; however, we only consider a one-way information flow from the remainder to the core part, which corresponds to the non-adversary components in the simulation-based frameworks. But note that we allow multiple remainders 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 frameworks since they use a central adversary machine.

We build on an existing formalization of CC in CryptHOL [16], which we refer to as CCHOL in this section. We reuse large parts of CCHOL's formalization of resources, converters, distinguishers, and various composition operators that were briefly recapped in the paper. Overall, our formalization required 9.7K lines of lemmas and definitions, where the case study constitutes 3.5k lines and the remainder form a library of reusable resource definitions and proof rules that can be used for modelling and reasoning about other security protocols.

We compare our result with CCHOL in four respects. First, the limitations mentioned in Section III-A apply to CCHOL proofs since it focuses on individual resources rather than specifications. For example, CCHOL's one-time-pad case study cannot be composed with a Diffie-Hellman key exchange like in Section V because CCHOL's notion of ideal key has a fixed communication pattern and does not allow for the activations that are needed to exchange the half-keys. Second, trace-equivalence proofs in our result are shorter and less complicated than those in CCHOL using the up-to proof rule for trace equivalence (Thm. 2). Third, CCHOL only comes with an asymptotic security definition whereas we provide both concrete and asymptotic statements and composition theorems. Finally, CCHOL's asymptotic security notion is too strong: in addition to $\mathfrak{d}(\pi \rhd R, \sigma \rhd S)$ being negligible in the security parameter, CCHOL requires a notion of functional correctness, which guarantees that the real resource provides the distinguisher with at least the same capabilities as the ideal resource. This hinders composability even further. For example, a secure channel does not securely realize an authenticated channel because the adversary cannot learn the contents of the channel. So one cannot use a secure channel when only an authenticated channel is needed.

There are few results on formalizing simulation-based proofs. Most of them [5], [10] focus on individual (or classes of) protocols and hence they do not offer composition theorems that can be applied in arbitrary context. EasyUC [8], which is the most similar to our work, does support the composition of security proofs. It uses EasyCrypt [2] 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 III-A, since it leads to a fixed communication pattern among the non-adversary machines. Second, formalizing security statements in EasyUC is more difficult. This stems from CC's abstract approach to cryptography, where we do not need to delve into concrete details that are intrinsic part of every security statement in UC-style frameworks. For instance, 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.

VII. Conclusion and Future Work

We have presented an abstract approach to communication modeling in Constructive Cryptography that is suitable for mechanized verification. We highlight the importance of system 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, which do not model system communications to the 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.

Acknowledgements

We thank the anonymous reviewers for their suggestions on the presentation.

References

[1] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *Theory of Cryptography (TCC 2003), Proceedings*, volume 2950 of *LNCS*, pages 335–354. Springer, 2003.

[2] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology (CRYPTO 2011), Proceedings*, volume 6841 of *LNCS*, pages 71–90. Springer, 2011.

[3] D. A. Basin, A. Lochbihler, and S. R. Sefidgar. CryptHOL: Game-based proofs in higher-order logic. *Journal of Cryptology*, 33(2):494–566, 2020.

[4] J. C. Blanchette, J. Hölzl, A. Lochbihler, L. Panny, A. Popescu, and D. Traytel. Truly modular (co)datatypes for Isabelle/HOL. In G. Klein and R. Gamboa, editors, *Interactive Theorem Proving (ITP 2014), Proceedings*, volume 8558 of *LNCS (LNAI)*, pages 93–110. Springer, 2014.

[5] D. Butler, D. Aspinall, and A. Gascón. How to simulate it in isabelle: Towards formal proof for secure multi-party computation. In *Interactive Theorem Proving (ITP 2017), Proceedings*, volume 10499 of *LNCS*, pages 114–130. Springer, 2017.

[6] J. Camenisch, R. R. Enderlein, S. Krenn, R. Küsters, and D. Rausch. Universal composition with responsive environments. In *Advances in Cryptology (ASIACRYPT 2016), Proceedings, Part II*, volume 10032 of *LNCS*, pages 807–840, 2016.

[7] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science (FOCS 2001), Proceedings*, pages 136–145. IEEE Computer Society, 2001.

[8] R. Canetti, A. Stoughton, and M. Varia. EasyUC: Using EasyCrypt to mechanize proofs of universally composable security. In *Computer Security Foundations Symposium, (CSF 2019), Proceedings*, pages 167–183. IEEE, 2019.

[9] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[10] H. Haagh, A. Karbyshev, S. Oechsner, B. Spitters, and P.-Y. Strub. Computer-aided proofs for multiparty computation with active security. In *Computer Security Foundations (CSF 2018), Proceedings*, pages 119–131. IEEE Computer Society, 2018.

[11] D. Hofheinz and V. Shoup. GNUC: A new universal composability framework. *Journal of Cryptology*, 28(3):423–508, 2015.

[12] D. Jost and U. Maurer. Overcoming impossibility results in composable security using interval-wise guarantees. In *Advances in Cryptology (CRYPTO 2020), Proceedings, Part I*, volume 12170 of *LNCS*, pages 33–62. Springer, 2020.

[13] R. Küsters and M. Tuengerthal. The IITM model: A simple and expressive model for universal composability. *IACR Cryptology ePrint Archive*, 2013:25, 2013.

[14] A. Lochbihler. Probabilistic functions and cryptographic oracles in higher order logic. In *European Symposium on Programming (ESOP 2016), Proceedings*, volume 9632 of *LNCS*, pages 503–531. Springer, 2016.

[15] A. Lochbihler and S. R. Sefidgar. Constructive cryptography in HOL: the communication modeling aspect. *Archive of Formal Proofs*, 2021. https://isa-afp.org/entries/Constructive_Cryptography_CM.html, Formal proof development.

[16] A. Lochbihler, S. R. Sefidgar, D. Basin, and U. Maurer. Formalizing constructive cryptography using CryptHOL. In *Computer Security Foundations Symposium (CSF 2019), Proceedings*, pages 152–166. IEEE, 2019.

[17] U. Maurer. Constructive cryptography - A new paradigm for security definitions and proofs. In *Theory of Security and Applications - Joint Workshop (TOSCA 2011), Revised Selected Papers*, volume 6993 of *LNCS*, pages 33–56. Springer, 2011.

[18] U. Maurer and R. Renner. Abstract cryptography. In *Innovations in Computer Science (ICS 2010), Proceedings*, pages 1–21. Tsinghua University Press, 2011.

[19] U. Maurer and R. Renner. From indifferentiability to constructive cryptography (and back). In *Theory of Cryptography Conference (TCC 2016), Proceedings, Part I*, volume 9985 of *LNCS*, pages 3–24. Springer, 2016.

[20] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[21] J. J. M. M. Rutten. Automata and coinduction (an exercise in coalgebra). In D. Sangiorgi and R. de Simone, editors, *Concurrency theory (CONCUR 1998), Proceedings*, volume 1466 of *LNCS*, pages 194–218. Springer, 1998.

[22] D. Sangiorgi. Beyond bisimulation: The "up-to" techniques. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, editors, *Formal Methods for Components and Objects (FMCO 2005), Proceedings*, volume 4111 of *LNCS*, pages 161–171. Springer, 2006.

Appendix

A. Guide to the Source Theory Files

All the definitions and lemmas in this paper are formalized and verified in the Isabelle/HOL proof assistant. In what follows, we provide a guide for the reader to navigate the source theories, which are available online [15].

The root directory consists of multiple theory files, the **Specifications** folder that contains the ideal specifications for keys and channels, and the **Constructions** folder that stores our case study's formalization. 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 auxiliary lemmas that are used in the other theory files.
- **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 remainder 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 3.
- **Asymptotic_Security.thy** extends the above to asymptotic notion of security.

The **Specifications** folder provides the ideal specifications for keys and channels. We explained the details of **Key.thy** in Section III-B3.

Our 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 encryption.
- **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.

B. Notation Index

In Table II, we provide a list of all symbols that are used in the paper. For each symbol, we provide a short explanation and a reference to the section in which that symbol is introduced for the first time.

Symbol	**Intro-duced**	**Meaning**
$e : \mathfrak{t}$	III-B1	Expression e has type $\mathfrak{t}$.
$\mathfrak{a}, \mathfrak{b}, \mathfrak{c}, \ldots$	III-B1	Type variables.
$\mathfrak{e}$	III-B2	Type variable for events.
$\mathfrak{i}$	III-C1	Type variable for inputs.
$\mathfrak{o}$	III-C1	Type variable for outputs.
$\mathfrak{q}$	III-B1	Type variable for queries.
$\mathfrak{r}$	III-B1	Type variable for responses.
$\mathfrak{s}, \mathfrak{t}$	III-B1	Type variable for states.
$\mathbb{A}, \mathbb{B}, \mathbb{C}, \ldots$	III-B1	Type constructors.
$\Rightarrow, \times, +$	III-B1	Infix type constructors for function space, pairs, and disjoint union.
$\mathbb{C}(\mathfrak{i}, \mathfrak{o}, \mathfrak{q}, \mathfrak{r})$	III-C1	Type of converters that take inputs of type $\mathfrak{i}$ and produce outputs of type $\mathfrak{o}$ by interacting with an oracle through queries of type $\mathfrak{q}$ and responses of type $\mathfrak{r}$.
$\mathbb{D}(\mathfrak{a})$	III-B1	Type of probability distributions over elementary events of type $\mathfrak{a}$.
$\mathbb{G}(\mathfrak{a}, \mathfrak{q}, \mathfrak{r})$	III-C1	Type of generative probabilistic values that produce answers of type $\mathfrak{a}$ by interacting with an oracle through queries of type $\mathfrak{q}$ and responses of type $\mathfrak{r}$.
$\mathbb{R}(\mathfrak{q}, \mathfrak{r})$	III-B1	Type of resources that accepts queries of type $\mathfrak{q}$ and produces responses of type $\mathfrak{r}$.
R	III-B1	A resource.
$\mathcal{R}, \mathcal{S}$	II-C	A specification, i.e., a set of resources.
C	III-C1	A converter.
C_{fuse}	III-C2	The *fuse* function as a converter.
D	IV-A	A distinguisher.
F	III-B2	A fused resource.
tr	III-B1	A probabilistic transition function, i.e., an oracle.
cr	III-C2	Short name for FRTs' core.
rm	III-C2	Short name for FRTs' remainder.
fuse, *FUSE*	III-B2	The fuse function on core-remainder parts and on FRTs.
$\{\!\!\mid name \mid\!\!\}$	III-B2	An FRT defined in terms of $name$ core.
$\boxed{c}$	III-B2	Place holder for cores' type.
$\boxed{r}$	III-B2	Place holder for remainder's type.
$+_{\mathbb{O}}$	III-B1	CryptHOL's plus-oracle operator.
$\parallel$	III-C1	Parallel composition of resources.
$\odot, \mid$	III-C1	Sequential and parallel composition of converters.
$\rhd$	III-C1	Attaching a converter to a resource.
$\blacktriangleright$	IV-A	Connecting a distinguisher to a resource.
$\rhd_c, \parallel_c$	III-C2	Attachment and parallel composition of cores.
$\rhd_r, \parallel_r$	III-C2	Attachment and parallel composition of remainders.
$\mathit{adv}(D, R_1, R_2)$	IV-B	Advantage of the distinguisher D to distinguish between R_1 and R_2.
$\mathit{dirac}(x)$	IV-B	One-point distribution on x.
$\gg\!=$	IV-B	Weighted combination of a family of probability distributions.
$\overline{\mathit{run}}(p, a)$	IV-B	The weighted combination of running a resource from the distribution p of resources with query a.
$p\upharpoonright_x$	IV-B	Conditioning of the distribution p over pairs $\mathfrak{a} \times \mathfrak{b}$ on the elementary event $x : \mathfrak{a}$.
X	IV-B	Relation between subdistributions of resource for the trace equivalence proof rule.
$[\![\_]\!]$	IV-B	Up-to closure operator for the trace equivalence up-to proof rule.
comm	III-B1	Isabelle/HOL specific commands.
NAME	III-B1	Definitions like *RES*, *CNV*, and *FUSE*.
`name`	III-B3	Artificial entities like interface names.

TABLE II: Notation index.