Formalizing Constructive Cryptography using CryptHOL

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

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

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

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

***Abstract*—Computer-aided cryptography increases the rigour of cryptographic proofs by mechanizing their verification. Existing tools focus mainly on game-based proofs, and efforts to formalize composable frameworks such as *Universal Composability* have met with limited success. In this paper, we formalize an instance of *Constructive Cryptography*, a generic theory allowing for clean, composable cryptographic security statements. Namely, we extend CryptHOL, a framework for game-based proofs, with an abstract model of *Random Systems* and provide proof rules for their equality and composition. We formalize security as a special kind of system construction in which a complex system is built from simpler ones. As a simple case study, we formalize the construction of an information-theoretically secure channel from a key, a random function, and an insecure channel.**

I. Introduction

A. Problem Context

Since the emergence of *provable security*, cryptographers have proposed various frameworks to tackle the complexity of security proofs. For example, game-based frameworks [7], [32] support the verification of cryptographic schemes and simulation-based frameworks [1], [13], [20], [23] enable the modular reasoning about cryptographic protocols. These frameworks have significantly improved the comprehensibility of security arguments; however, the resulting proofs are still informal, often sketchy, and sometimes even technically incomplete or wrong.

In response to the *crisis of rigor* in cryptography [7], [18], the formal-methods community has developed tools that enable cryptographers to use computers to mechanically check security proofs. Prominent examples are CryptoVerif [10], CertiCrypt [2], EasyCrypt [3], Verypto [9], FCF [30], and CryptHOL [6]. All these tools focus primarily on the game-based paradigm, and the results on formalizing simulation-based proofs are limited to those that study individual protocols, e.g., [12], [17].

The lack of formal-methods tools for simulation-based frameworks is due, by and large, to their complexity. These frameworks are popular since they support the composition of security proofs. However, the level of details in these frameworks surpasses what the formal-methods community can reasonably handle with existing techniques. Formalizing security in a framework such as Universal Composability would require formal notions of algorithms, runtime, and complexity, which would be both challenging and detailed due to the way these foundations are modeled. For example, algorithms are modeled as Turing Machines, which are too concrete for efficient mechanized proof checking.

B. Our Solution

To alleviate the above obstacle and raise the abstraction level of our reasoning, we take a different view of composable security statements and follow the Abstract Cryptography approach [26], [27]. In particular, we extend CryptHOL [6], [24], a framework for formalizing game-based proofs in Isabelle/HOL [29], to support the formalization of security proofs in the Constructive Cryptography framework [26], [27]. Constructive Cryptography, an instance of Abstract Cryptography, supports clean, composable cryptographic security statements in which every cryptographic scheme constructs a *resource* from a set of assumed resources.

First, we introduce new coalgebraic datatypes that model resources and resource transformers as abstract probabilistic input-output systems. Our formalization constitutes an instance of Maurer's theory of Random Systems [28] where the coalgebraic approach makes it amenable for mechanized proof checking. Second, we formalize an algebra of abstract systems built using different composition operators. Third, we define equality of the abstract systems as trace equivalence. This enables us to formalize asymptotic information-theoretic security as a special type of construction that resembles the well-known *ideal-world real-world* paradigm [15]. Finally, as a simple case study, we use our framework to formalize the security of a protocol that constructs a secure channel from a key, a random function, and an insecure channel.

We chose to formalize information-theoretic security so that we can focus on studying composability without the distractions of a computational model. Our coalgebraic approach, trace equivalence, and our proof rules also work in a computational setting, which we leave as future work.

The complete formalization, including our case study, is available online [25]. Appendix A provides entry points to the source files. Besides the fact that CryptHOL uses Isabelle/HOL, our formalization builds on Isabelle/HOL since it supports coalgebraic datatypes that recurse through non-free functors like discrete probability distributions. In principle, we could have carried out our formalization in other proof assistants like Coq or Lean, where constructing the codatatypes probably would require more effort, but dependent types would simplify the formalization of interfaces.

C. Contributions

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

- On top of CryptHOL, we formalize an instantiation of Constructive Cryptography where resources have multiple interfaces. This includes formalizing the composition of multi-interface resources. We demand that outputs must be provided over the same interface as the query. This allows us to reason algebraically about composition.
- We model resources coalgebraically as probabilistic input-output systems. We introduce a notion of probabilistic traces and show that a resource's probabilistic traces precisely capture the behaviour that the adversary can observe. We also present a novel bisimulation-style proof rule that is sound and complete for establishing the trace equivalence of resources. Existing logics like pRHL [5] are incomplete for trace equivalence and simulation-based proofs in particular suffer from this incompleteness.
- We formalize two standard constructions in Constructive Cryptography as case studies: the construction of an authentic channel from a shared random function (idealizing a pseudo-random function applied to a shared secret key), and the one-time pad construction of a secure channel from an authentic channel and another secret key. We use composition theorems to compose these construction steps.

D. Structure

We start by reviewing the essentials of CryptHOL and Constructive Cryptography in Section II. In Section III, we introduce our running example. In Section IV, we describe our formalization of abstract systems and their composition operators. In Section V, we define trace equality of abstract systems and derive the corresponding proof rule. In Section VI, we define information-theoretic security in terms of abstract systems construction and prove the composition theorems and that trace equivalence is the right equality notion. In Section VII, we delve into our case study, illustrating our framework's applicability. Finally, in Sections VIII and IX, we compare with related work and draw conclusions.

II. Background

This section summarizes the background needed for understanding the rest of this paper. More details can be found in the references cited.

A. CryptHOL

With the CryptHOL framework [6], [24], game-based cryptographic proofs can be formalized in higher-order logic (HOL) [16]. The proofs are mechanically checked by the proof assistant Isabelle/HOL [29], which ensures that every proof step is a valid application of HOL's logical inference rules. Games in CryptHOL are expressed as probabilistic functional programs whose semantic is expressed using discrete probabilities and *Generative Probabilistic Values (GPV)*, a denotational domain for probabilistic systems with inputs and outputs. Formally, we write $\mathbb{D}(\alpha)$ for discrete probability distributions over elementary events of type α, and $\mathbb{G}(\alpha, \beta, \gamma)$ for probabilistic input-output (IO) systems that interact with their environment through queries of type β with responses of type γ and, upon termination, return a result of type α. All IO systems terminate with probability one.[1] From the semantics, CryptHOL derives proof rules for program equivalences and typical cryptographic arguments in game-based proofs.

B. Constructive Cryptography

Maurer and Renner's Abstract Cryptography framework [26], [27] proposes a top-down paradigm for developing a theory of cryptography. In the conventional bottom-up approach, the notions of algorithms, complexity, and efficiency are fixed before the security of cryptographic constructs can be defined. In contrast, theorems in Abstract Cryptography can be proved at a (high) level of abstraction without the instantiation of the lower levels. The lower levels inherit these theorems if they satisfy the postulated axioms of the higher level. Each abstraction level can thus focus on specific aspects, such as composability or efficiency.

At the highest level, Abstract Cryptography studies how complex systems can be built from simpler ones. Consider a *Component set* Ω equipped with a parallel composition operator $\|$ and a *Constructor set* Γ equipped with serial composition $\circ$ and parallel composition $|$. A construction $\mathcal{R} \xrightarrow{x} \mathcal{S}$ expresses that the component $\mathcal{S} \in \Omega$ can be built from the component $\mathcal{R} \in \Omega$ using the constructor $x \in \Gamma$. Maurer and Renner show that the following properties are sufficient (and necessary) for *General Composability*:

1) $\mathcal{R} \xrightarrow{id} \mathcal{R}$, where id denotes the identity constructor.
2) If $\mathcal{R} \xrightarrow{x} \mathcal{S}$ and $\mathcal{S} \xrightarrow{y} \mathcal{T}$, then $\mathcal{R} \xrightarrow{x \circ y} \mathcal{T}$.
3) If $\mathcal{R} \xrightarrow{x} \mathcal{S}$, then $\mathcal{R} \| \mathcal{T} \xrightarrow{x|id} \mathcal{S} \| \mathcal{T}$ and $\mathcal{T} \| \mathcal{R} \xrightarrow{id|x} \mathcal{T} \| \mathcal{S}$.

To make the building blocks of a construction more concrete, they introduce an abstract theory of systems. In this theory, every aspect of a component, including the adversary's capabilities and the communication network, is made explicit as a *Resource* system with input-output interfaces. Constructors are modeled as (possibly multiple) *Converter* systems. Attaching a converter x to the interfaces of a resource $\mathcal{R}$ results in a new (more complex) resource $x\mathcal{R}$.

Constructive Cryptography [26], [27] models resources as Random Systems [28] and converters as transformers of random systems. A random system is a family of conditional probabilities: the probability of each output is conditioned on

[1]CryptHOL's semantic domains $\mathbb{D}$ and $\mathbb{G}$ also contain sub-probability distributions and non-terminating IO systems, which are convenient for formalizing probabilistic termination, conditional probabilities and failure events [6]. The formalized versions of this paper's theorems therefore assume probabilistic termination explicitly and we have proved probabilistic termination for all formalized examples. Our presentation omits this technicality for readability.

the current input and all previous input-output pairs. Resources have one interface for every party—e.g., A for Alice and B for Bob—plus the adversary's interface E. Let $x^A\mathcal{R}$ denote that the converter x is attached to the interface A of the resource $\mathcal{R}$. A new resource is constructed by attaching a constructor, which consists of one or more converters, to each interface of an existing resource. For example, $x^A y^B z^E \mathcal{R}$ represents the resource that is constructed from the attachment of converters x, y, and z to $\mathcal{R}$'s interfaces A, B, and E, respectively.

Security is defined using distinguishers following the ideal-world real-world paradigm [15] with a simulator. Let $\mathfrak{d}(\mathcal{R}, \mathcal{S})$ denote the least upper bound on the advantages of all distinguishers in distinguishing $\mathcal{R}$ from $\mathcal{S}$. In the information-theoretic setting, this includes computationally unbounded distinguishers. In a two-party setting, with parties A and B, a protocol with converters x and y *(securely) constructs* the resource $\mathcal{S}$ from the resource $\mathcal{R}$ within ϵ, denoted as $\mathcal{R} \xrightarrow{(x,y)}_{\epsilon} \mathcal{S}$, if the following conditions hold:

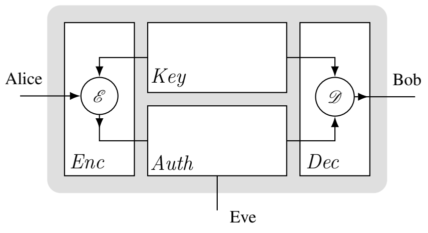
1) $\mathfrak{d}(x^A y^B {}_{\mathcal{S}}{\hookrightarrow}^{E}_{\mathcal{R}} \mathcal{R}, \mathcal{S}) \leq \epsilon$, where the converter ${}_{\mathcal{S}}{\hookrightarrow}_{\mathcal{R}}$ embeds $\mathcal{S}$'s E-interface into $\mathcal{R}$'s.
2) $\exists \sigma.\ \mathfrak{d}(x^A y^B \mathcal{R}, \sigma^E \mathcal{S}) \leq \epsilon$, where σ is the *Simulator*.

The first condition ensures functional correctness, namely that the two resources have the same behaviour when the adversary does not use the additional capabilities that $\mathcal{R}$ exposes on the E interface; they are disabled by the converter ${}_{\mathcal{S}}{\hookrightarrow}_{\mathcal{R}}$, which is determined by the pair of interfaces $\mathcal{R}$ and $\mathcal{S}$. The second condition demands that the simulator can mimick the real-world interface using only the ideal-world interface.

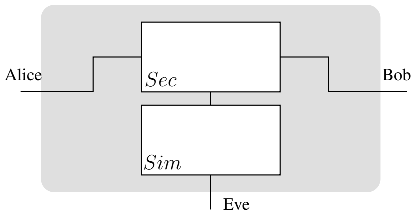
III. Running example

We now introduce our running example: the construction of a secure communication channel. In this section, we consider the case where, using encryption, we construct such a channel from an authentic channel and a key shared between two parties Alice and Bob. In Section VII, we will extend this construction using a message authentication code. Following the constructive cryptography approach, we start with two resources: the authentic communication channel $Auth$ and a shared key Key. Alice then attaches an encryption converter Enc on her side of the channel and Bob the decryption converter Dec on his side. Figure 1a shows the construction. Eve controls the communication network, and here this is formalized via the adversary interface to the channel. Eve can generally look at the contents and delay or drop messages. However, as the channel here is authentic, Eve cannot modify messages or inject new ones. By definition, Eve cannot access the key resource.

We say that an encryption scheme parametrizing the converters Enc and Dec is secure if it securely constructs a secure channel from an authentic channel and a key. A secure channel differs from an authentic channel only in the adversary's interface: The adversary learns just the length of sent messages, but not their content. Recall from the previous section that a secure construction means that—in addition to functional correctness—there exists a simulator Sim that can simulate Eve's interface of the authentic channel using only her



(a) The real world



(b) The ideal world with the simulator

Fig. 1: *Alice*, *Bob*, and *Eve* are the interface names which correspond to the A, B, and E interfaces in the background section's security definition. $\mathscr{E}$ and $\mathscr{D}$ denote the encryption and the decryption functions, respectively. Each gray rectangle represents the resource built from the resources and converters in its interior.

interface of the secure channel. That is, the security proof must exhibit a simulator Sim such that the real world in Fig. 1a is indistinguishable from the ideal world in Fig. 1b.

IV. Formalizing Resources and Converters

Resources and converters are the building blocks of cryptographic systems and constructions, respectively. We now formalize these building blocks (Sections IV-A, IV-B) and how they can be composed to build more complex systems (Section IV-C). The trace equivalence and security notions in Sections V and VI will be based on this novel coalgebraic formalization of resources.

A. Resources

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

In this representation, the internal state s is explicit. This complicates composition arguments that involve distinguishing resources based on their input-output behaviour. We therefore introduce an abstraction that hides the resource's internal state.

Formally, the resource type quantifies existentially over the state type in the pair of the transition function and the initial state:

$$\mathbb{R}(\alpha, \beta) \equiv \exists \sigma.\ (\sigma \Rightarrow \alpha \Rightarrow \mathbb{D}(\beta \times \sigma)) \times \sigma,$$

where α represents the type of inputs, β the type of outputs, and $\mathbb{D}(\gamma)$ probability distributions over γ.

Fortunately, this abstraction can be expressed without existential types, which do not exist in HOL. In the co-algebraic view on reactive systems [31], the transition function *run* : $\mathbb{R}(\alpha, \beta) \Rightarrow \alpha \Rightarrow \mathbb{D}(\beta \times \mathbb{R}(\alpha, \beta))$ for interacting with a resource defines a co-algebra on resources. That is, when we supply a resource with an input, we obtain a probability distribution over an output and a successor resource. In Isabelle/HOL, we formalize this view using the following co-datatype:

$$\textbf{codatatype}\ \mathbb{R}(\alpha, \beta) = \mathit{Resource}\ (\alpha \Rightarrow \mathbb{D}(\beta \times \mathbb{R}(\alpha, \beta))).$$

The codatatype is the final coalgebra for the transition function *run*. Finality means that two resources with the same input-output behaviour are equal. As we will show in Section V, equality corresponds to bisimilarity of the probabilistic transition systems.

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

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

Such probabilistic transition systems with explicit states have already been formalized as part of CryptHOL, where they are used to model cryptographic oracles. So we can reuse CryptHOL's infrastructure for formalising and reasoning about oracles. In particular, CryptHOL's oracle composition operator $+_{\mathbb{O}}$ makes it possible to construct a resource from simpler parts. It interleaves two transition functions $\delta_i : \sigma \Rightarrow \alpha_i \Rightarrow \mathbb{D}(\beta_i \times \sigma)$ (for $i = 1, 2$) operating on a shared state of type σ into one transition function $\delta_1 +_{\mathbb{O}} \delta_2 : \sigma \Rightarrow \alpha_1 + \alpha_2 \Rightarrow \mathbb{D}((\beta_1 + \beta_2) \times \sigma)$, where ":" denotes each term's type and $\alpha + \beta$ denotes the disjoint union of the types α and β.

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

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

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

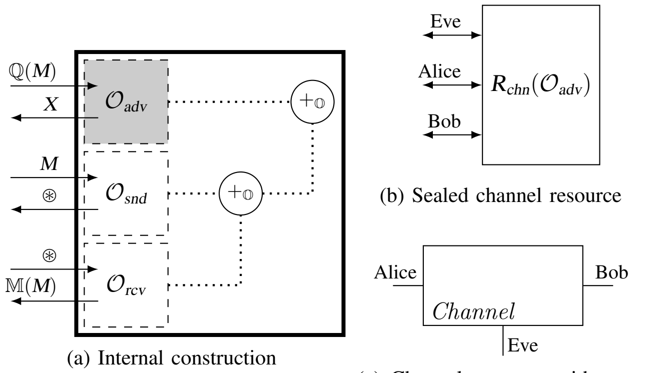
In Figure 2a, we show how $+_{\mathbb{O}}$ and *RES* can be used to formalize the notion of a channel. The parties' interaction with the channel is modeled using three probabilistic transition systems $\mathcal{O}_{snd}$, $\mathcal{O}_{rcv}$, and $\mathcal{O}_{adv}$. The operator $+_{\mathbb{O}}$ composes these oracles into a single transition system. Formally, inputs and outputs are tagged according to their location in the dotted term tree using the injections *Left* and *Right* for disjoint unions. For example, the queries to $\mathcal{O}_{rcv}$ and their corresponding answers are tagged with *Right Right*. In the presentation, we usually omit these tags when they are clear from the context. The *RES* operator, visualized as the bold rectangle, seals the result of the composition and produces a channel resource $R_{chn}(\mathcal{O}_{adv})$ as shown in Fig. 2b. Formally, we write this as follows, where *empty* denotes the initial state:

$$R_{chn}(\mathcal{O}_{adv}) \equiv \mathit{RES}(\mathcal{O}_{adv} +_{\mathbb{O}} \mathcal{O}_{snd} +_{\mathbb{O}} \mathcal{O}_{rcv}, \mathit{empty}).$$

Figure 2c shows the same resource in the notation from Section III. In the remainder of this paper, we use the notation from Fig. 2b, where the different interfaces are combined into one using disjoint unions. In particular, we will not further describe the internal construction of resources, although all our concrete example resources are defined using *RES*.

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

2) A type system for interfaces: Encoding several interfaces into one using tagging has a drawback: The type of resources does not enforce that the response is sent via the same interface as the query. For example, in Fig. 2, a query $\mathbb{Q}(M)$ to $\mathcal{O}_{adv}$ must be responded over $\mathcal{O}_{adv}$'s response port (of type X) and not over $\mathcal{O}_{rcv}$'s. Clearly, resources constructed with *RES* and $+_{\mathbb{O}}$ ensure this property. But unless we look at the internal construction, it is not obvious that this property holds—and we do not want to constantly unfold our definitions. Yet, this property is crucial



(a) Internal construction

(b) Sealed channel resource

(c) Channel resource with notation from Section III

Fig. 2: Formalisation of a communication channel between Alice and Bob.

when reasoning about resources and converters. We therefore introduce a type system for interfaces that allows us to express arbitrary relations between inputs and outputs, i.e. arbitrary non-temporal, non-probabilistic properties of the resource.

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

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

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

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

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

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

$$(\mathcal{I}_1 \oplus \mathcal{I}_2)^B(a) = \begin{cases} \mathcal{I}^B_1(a) & \text{if } a \in \mathcal{I}^A_1 \\ \mathcal{I}^B_2(a) & \text{if } a \in \mathcal{I}^A_2. \end{cases}$$

For example, let $\mathcal{I}_x$ be $\mathcal{O}_x$'s interface type for $x \in \{\text{adv}, \text{snd}, \text{rcv}\}$. Then the channel resource R_{chn} in Fig. 2 respects $\mathcal{I}_{\text{adv}} \oplus \mathcal{I}_{\text{snd}} \oplus \mathcal{I}_{\text{rcv}}$ by construction. Once we have proven respectfulness, Isabelle's reasoning engine can exploit this property without exposing the resources' internal construction.

3) Parallel composition: When building complex systems from simple resources, many resources are typically available at the same time. For example, in Fig. 1a, both a key and an authentic channel are available We model this using parallel composition for resources.

Definition 2 (Parallel composition)**.** *Let $R_1 : \mathbb{R}(\alpha_1, \beta_1)$ and $R_2 : \mathbb{R}(\alpha_2, \beta_2)$ be resources. The parallel composition $R_1 || R_2 : \mathbb{R}(\alpha_1 + \alpha_2, \beta_1 + \beta_2)$ directs queries α_1 to R_1 and α_2 to R_2 and forwards the responses accordingly.*

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

Note that $\boldsymbol{RES}\ (\delta_1 +_{\mathbb{O}} \delta_2, s)$ and $\boldsymbol{RES}\ (\delta_1, s) || \boldsymbol{RES}\ (\delta_2, s)$ are *not* the same. Parallel resource composition ensures that the two resources do not share their state. In particular, probabilistic choices in one resource are independent of those in other resources. So $||$ allows the occurrences of the states s to evolve independently, whereas $+_{\mathbb{O}}$ interleaves δ_1 and δ_2 on the shared state s. Consequently, if correlation or state sharing is required, $+_{\mathbb{O}}$ and $\boldsymbol{RES}$ must be used. Hence, $+_{\mathbb{O}}$ cannot be lifted to the abstract level of resources.

B. Converters

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

A converter has two interfaces: Inputs and outputs are sent over the external interface, and to compute a response, the converter itself may send queries over the internal interface and wait for responses. So a converter is a probabilistic transition system with two layers: On the outer, external layer, it appears like a resource. However, every transition may consist of many internal transitions that drive the interaction with the resource that the converter transforms. This is the second, internal layer. In Fig. 1a, the converters Enc and Dec transform the resources Key and Auth. In our diagrams in the remainder of the paper, the external interface is always on the left and the internal interface on the right of a converter.

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

$$\textbf{codatatype}\ \mathbb{C}(\alpha,\beta,\gamma,\eta) = \textit{Converter}\ (\alpha \Rightarrow \mathbb{G}(\beta \times \mathbb{C}(\alpha,\beta,\gamma,\eta),\gamma,\eta)).$$

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

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

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

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

- Let $C_1 : \mathbb{C}(\alpha_1,\beta_1,\gamma_1,\eta_1)$ and $C_2 : \mathbb{C}(\alpha_2,\beta_2,\gamma_2,\eta_2)$ be two converters. Their parallel composition $C_1 \mid C_2 : \mathbb{C}(\alpha_1{+}\alpha_2,\beta_1{+}\beta_2,\gamma_1{+}\gamma_2,\eta_1{+}\eta_2)$ makes both converters available at the same time, analogous to parallel resource composition. If $\mathcal{I}_1 \vdash C_1 \dashv \mathcal{I}_1'$ and $\mathcal{I}_2 \vdash C_2 \dashv \mathcal{I}_2'$, then $\mathcal{I}_1 \oplus \mathcal{I}_2 \vdash C_1 \mid C_2 \dashv \mathcal{I}_1' \oplus \mathcal{I}_2'$.
- Let $C_1 : \mathbb{C}(\alpha,\beta,\gamma,\eta)$ and $C_2 : \mathbb{C}(\gamma,\eta,\chi,\delta)$ be two converters. Their sequential composition $C_1 \odot C_2 : \mathbb{C}(\alpha,\beta,\chi,\delta)$ uses C_2 to answer C_1's queries on C_1's internal interface. If $\mathcal{I}_1 \vdash C_1 \dashv \mathcal{I}_2$ and $\mathcal{I}_2 \vdash C_2 \dashv \mathcal{I}_3$, then $\mathcal{I}_1 \vdash C_1 \odot C_2 \dashv \mathcal{I}_3$.
- The identity converter $\mathbb{1} : \mathbb{C}(\alpha,\beta,\alpha,\beta)$ simply forwards all queries and responses from the external to the internal interface and vice versa. It is the neutral element for sequential composition: $\mathbb{1} \odot C = C \odot \mathbb{1} = C$. Clearly, $\mathcal{I} \vdash \mathbb{1} \dashv \mathcal{I}$.

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

C. Constructing Systems

We now show how to build complex systems. In the simplest case, we want to attach a converter to a resource (Fig. 3). Let

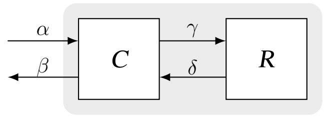


Fig. 3: Attaching a converter to a resource.

$C : \mathbb{C}(\alpha,\beta,\gamma,\delta)$ be a converter and $R : \mathbb{R}(\gamma,\delta)$ be a resource. Attaching C to R creates a new resource $C \rhd R : \mathbb{R}(\alpha,\beta)$, where R responds to C's queries.

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

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

These properties are much more concise than the corresponding equations in CryptHOL, where the internal state is not hidden in the coalgebraic view. For example, the CryptHOL equivalent to (1) reads as follows, where $\widehat{\mathbb{D}}(f)(X)$ applies the function f to the random variable X.

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

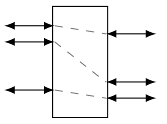
The equation is structurally the same (*inline* corresponds to $\odot$), but the essence is buried under the clutter that explicit state-passing introduces. This demonstrates the gain in abstraction that our formalisation provides over CryptHOL.

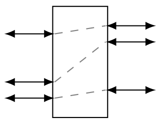
Wiring: The order of converter queries does not always correspond to the resource interfaces, which happens when resources' and converters' interfaces are composed in a different order. For example, $\mathcal{I}_1 \oplus (\mathcal{I}_2 \oplus \mathcal{I}_3)$ is not the same as $(\mathcal{I}_1 \oplus \mathcal{I}_2) \oplus \mathcal{I}_3$ because disjoint union is not associative in HOL. We therefore introduce three embedding converters, called *Wiring Converters*, that rearrange interfaces into the desired order, as shown in Fig. 4.[2]

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

By composing the identity and wiring converters sequentially and in parallel, we can express arbitrary attachments of converters and resources. More precisely, we can express that a converter should be attached only to a subset of the interfaces of a resource, and we can arbitrarily reassociate and permute this subset since every permutation can be expressed as a sequence of transpositions of adjacent positions. Figure 5a shows an

[2]In Fig. 4, the arrow's closeness indicates the association of the disjoint union, i.e., the order of interface composition. Except for Figs. 4 and 5, we omit this detail in the diagrams. Of course, we take care of reassociations in our formalisation.


(a) *lassocr*


(b) *rassocl*

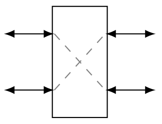
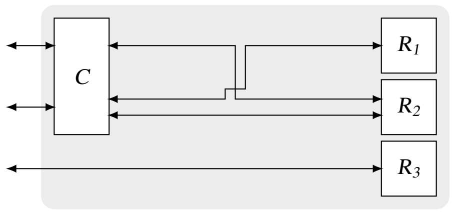
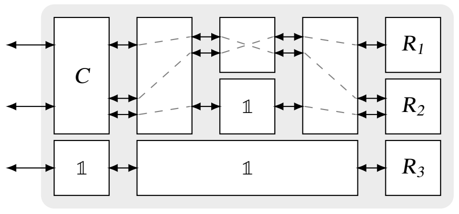

(c) *swap*

Fig. 4: Wiring converters. The arrows' closeness signify the order of interface composition.



(a) Abstract representation



(b) Detailed construction

Fig. 5: The result of attachment is the new resource shown with dark-gray.

example where a converter C with two external interfaces and three internal interfaces (associated to the right) is attached to two resources R_1 and R_2. Another resource R_3 is available, but not involved in the attachment. Figure 5b visualizes how such a diagram is formalized using the wiring converters:

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

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

V. Equivalence of Resources

To reason about the construction of cryptographic systems, we need an equivalence notion on these systems, i.e., resources. In this paper, we introduce three equivalences on resources: bisimilarity, trace equivalence, and indistinguishability (with negligible advantage). Bisimilarity is stronger than trace equivalence (Cor. 1) and trace equivalence is stronger than indistinguishability (Thm. 2). Indistinguishability is used in the security definition in Section VI. Yet, indistinguishability is unwieldy in proofs because negligibility requires asymptotic reasoning. In this section, we therefore introduce the two stronger notions first. They are useful as many steps in security proofs actually establish one of the stronger equivalences; our case study in Section VII gives some examples. Stronger equivalences simplify the reasoning in two respects. First, their proof rules are simpler: Trace equivalence does not need asymptotics, only relational reasoning about distributions on states (Thm. 1), and for bisimilarity, relations between individual states suffice. Second, when two systems satisfy a stronger equivalence, we may replace one with the other in more contexts, so we must check fewer conditions on the context to justify the replacement.

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

Definition 4 (Bisimilarity). *Consider two resources R_1 and R_2 of type $\mathbb{R}(\alpha, \beta)$ that respect the interface type $\mathcal{I}$. Without loss of generality, assume that $R_i = \mathit{RES}\ (\delta_i, s_i)$ for $i \in \{1, 2\}$.*[3] *A relation X between the state spaces σ_1 and σ_2 is a bisimulation relation if and only if*

1) *X relates the two initial states s_1 and s_2, and*
2) *whenever $(s_1', s_2') \in X$ and $a \in \mathcal{I}^A$, there is a joint probability distribution $d : \mathbb{D}(\beta \times \sigma_1 \times \sigma_2)$ with marginal distributions $\delta_1(s_1', a)$ and $\delta_2(s_2', a)$ whose support is contained in $\mathcal{I}^B(a) \times X$.*

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

Bisimilarity is well-suited for proving resource equivalence since it is compositional and has elegant connections with relational parametricity (see Basin et al. [6]). For example, probabilistic relational Hoare logic [4], [5] as used in EasyCrypt [2] establishes bisimilarity. In fact, if the interface type $\mathcal{I}$ allows all queries and responses (i.e., $\mathcal{I}(a) = \mathit{UNIV}$ for all a where *UNIV* denotes the set of all elements of a type), then bisimilarity coincides with logical equality in HOL as we model resources as a codatatype.

Unfortunately, bisimilarity sometimes is too strong. For example, consider two resources that accept $\circledast$ as input and respond with an element from the set $\{a, b, c\}$. The diagrams in Fig. 6 show the behaviour of the two resources as probabilistic transition systems. Every edge is labelled with the response and the probability that the edge is taken during an interaction. Starting in the state at the top, both systems respond with a to the first query and with b and c with probability $1/2$ each to the second query. Thus, no distinguisher can distinguish the two systems through interaction. However, the system on the left decides already in the first interaction whether it will respond with b or c in the second interaction. In contrast, the system on the right makes this choice only during the second interaction. The two systems are therefore not bisimilar: For if X was a bisimulation relation, it would have to relate s_1 and s_1' each with t_1. However, this is impossible because t_1 can

[3]Every resource R can be expressed as *RES* (δ, s) by choosing $\delta = \mathit{run}$ and $s = R$.

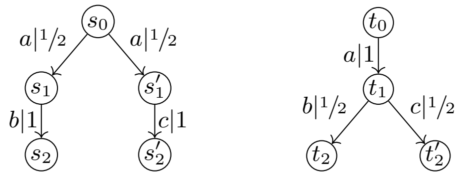


Fig. 6: Two probabilistic transition systems that are trace equivalent, but not bisimilar.

output both b and c whereas s_1 and s'_1 can each only produce one of them.

In a non-deterministic setting, when considering outputs without probabilities, this is the standard example that separates bisimilarity from the coarser notion of trace equivalence. Intuitively, the problem in Fig. 6 is that the states s_1 and s'_1 encode hidden non-determinism that will be unveiled later. An observer of the system on the left does not yet know whether the system is in s_1 or s_2. It knows only a *distribution* over those states, where each state is assigned with the likelihood that the system is in that state. This distribution is given by the transition probabilities of the system conditioned on the past interactions. Here, this conditional distribution is uniform over s_1 and s'_1. These conditional distributions are the probabilistic equivalent of the well-known powerset construction for non-deterministic automata. If we ignore the probabilities, then the support of the conditional distribution denotes the set of states reachable under the given input.

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

Traces and trace equivalence and their unwinding characteristics have previously been studied in the coalgebraic approach to modelling systems [19], [21], [22], [33]. Unfortunately, none of these results can be applied directly to resources in our setting because they need either a ccpo structure or a decomposition of the coalgebra's functor F into a functor G and a monad T such that $F(X) = G(T(X))$ or $F(X) = T(G(X))$. For resources, we have $F(R) = \alpha \Rightarrow \mathbb{D}(\beta \times R)$ with $T = \mathbb{D}$ as the monad, but no ccpo structure and no suitable decomposition. Abstractly, this is because the summaries, i.e., conditional distributions over states, are not explicit in our representation. Fortunately, a distribution $\mathbb{D}(\beta \times R)$ over pairs can be equivalently represented as two parts: the marginal distribution over the first component $\mathbb{D}(\beta)$ and a family of conditional distributions over the second component (the summaries), indexed by the support of the first component: $\beta \Rightarrow \mathbb{D}(R)$. Thus, we can think of resources as a coalgebra of the functor $F'(R) = \alpha \Rightarrow (\mathbb{D}(\beta) \times (\beta \Rightarrow \mathbb{D}(R)))$, which we can decompose as $G(\mathbb{D}(R))$ with $G(Y) = \alpha \Rightarrow (\mathbb{D}(\beta) \times (\beta \Rightarrow Y))$. To our knowledge, this functor has not yet been considered in the literature. In the remainder of this section, we apply the abstract theory to resources and define traces and trace equivalence.

The trace of a resource R is a function from past input-output pairs to a family of sub-probability distributions over outputs, indexed by inputs, i.e.,

$$\mathit{trace} : \mathbb{R}(\alpha, \beta) \Rightarrow \mathbb{L}(\alpha \times \beta) \Rightarrow \alpha \Rightarrow \mathbb{D}(\beta),$$

where $\mathbb{L}(\alpha \times \beta)$ denotes the type of lists of pairs $\alpha \times \beta$.

In the example resource on the left of Fig. 6, the trace of the top state probabilistically combines the traces of its two successor states. When a system interacts with this resource, it knows after the first interaction only that the resource is in state s_1 with probability $1/2$ and in s_2 with the same probability. The system will learn whether it is s_1 or s_2 only during the second interaction.

This example shows that we must support reasoning about probability distributions on the states. This entails that we define traces for distributions over resources.[4] The trace of a single resource R then is the special case for the one-point distribution $\mathit{dirac}(R)$. Formally, if $p : \mathbb{D}(\mathbb{R}(\alpha, \beta))$ is a distribution of resources, let $\overline{\mathit{run}}(p, a)$ denote the weighted combination of running the resources from p with input a, i.e.,

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

We next define the traces by recursion over the list of past input-output pairs:

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

[4]If we ignore probabilities and consider non-deterministic systems, we can directly define the traces for a state, without first going to sets of states. We now show that this does not work for probabilistic systems. In a non-deterministic system, the corresponding trace function returns the set of possible responses (rather than their distribution) for every input given the previous input-output pairs, i.e., $\mathit{trace} : \mathbb{R}(\alpha, \beta) \Rightarrow \mathbb{L}(\alpha \times \beta) \Rightarrow \alpha \Rightarrow \mathbb{P}(\beta)$ where $\mathbb{P}(\beta)$ denotes the powerset of β. This function *can* be defined recursively over the list of input-output pairs by taking the union over the possible successor states: $\mathit{trace}(s, (x, y) \cdot l, a) = \bigcup_{(y, s') \in \delta(s, x)} \mathit{trace}(s', l, a)$.

This definition only works because conditioning on the output y distributes over unions: $\{s \mid (s, y) \in \bigcup \mathfrak{A}\} = \bigcup_{A \in \mathfrak{A}} \{s \mid (s, y) \in A\}$ holds for all y and $\mathfrak{A}$. For probability distributions, the corresponding identity, taking the weighted combination instead of the union, does not hold. Therefore, our generalisation to distributions over states (or resources) is necessary. (The pointwise non-deterministic definition can be interpreted in probabilities too, but this leads to the wrong definition of trace equivalence.) Categorically speaking, the powerset functor $\mathbb{P}$ is additive, but the distribution functor $\mathbb{D}$ is not [14].

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

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

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

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

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

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

1) *X relates $\mathsf{dirac}(R_1)$ to $\mathsf{dirac}(R_2)$, and*
2) *Whenever $(p, q) \in X$ and $a \in \mathcal{I}^A$, then $\overline{\mathsf{run}}(p, a)$ and $\overline{\mathsf{run}}(q, a)$ have the same marginal distribution on outputs and $(\mathsf{run}(p, a)\upharpoonright_b, \mathsf{run}(q, a)\upharpoonright_b) \in X$ for all $(b, \_) \in \mathit{support}(\overline{\mathsf{run}}(q, a))$.*

Corollary 1. *Bisimilar resources are trace equivalent.*

For example, Thm. 1 allows us to prove trace equivalence of the two resources in Fig. 6 in a bisimulation style. We pick as the relation X the following four pairs of probability distributions, identifying a state with the resource with that initial state. The notation $[x_1|p_1, x_2|p_x, \ldots, x_n|p_n]$ denotes the probability distribution where the elementary event x_i has probability p_i.

- $([s_0|1], [t_0|1])$
- $([s_1|1/2, s_1'|1/2], [t_1|1])$
- $([s_2|1], [t_2|1])$
- $([s_2'|1], [t_2'|1])$

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

VI. Secure Constructions

Trace equivalence expresses that two systems are identical from an observer's point of view. This equivalence notion is still too strong for cryptographic constructions as there is in general a small probability that the two systems are not the same, for example, if the adversary guesses a secret. In this section, we define a coarser equivalence notion based on distinguishers and the ideal-real paradigm.

A. Distinguishers

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

$$\textbf{type-synonym } \mathbb{A}(\alpha, \beta) = \mathbb{G}(\mathbb{B}, \alpha, \beta).$$

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

Moreover, a distinguisher $D : \mathbb{A}(\alpha, \beta)$ can absorb a converter $C : \mathbb{C}(\alpha, \beta, \gamma, \eta)$. The result $D \circledast C : \mathbb{A}(\gamma, \eta)$ is again a distinguisher, but for resources of type $\mathbb{R}(\gamma, \eta)$. The absorption operator $(\circledast)$ corresponds to CryptHOL's inline operator. Connection and absorption satisfy two important identities:

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

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

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

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

Theorem 2 (Characterisation of trace equivalence)**.** *Let R_1 and R_2 be two resources respecting $\mathcal{I}$. Then the following are equivalent:*

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

B. Security model

We now define the security of a construction following the ideal-real paradigm. The ideal resource defines the secure "functionality", and we would like to show that a real resource realizes the same functionality, although it may differ with small probability. We focus on information-theoretic security as we do not restrict the class of distinguishers. Computational security could be defined similarly except that we would need to formalize a computational model, which we leave as future work.

Formally, we consider three interfaces: the user interface with type $\mathcal{I}_{\mathrm{U}}$ to the actual functionality and the adversary's interfaces with types $\mathcal{I}_{\mathrm{I}}$ and $\mathcal{I}_{\mathrm{R}}$ to the ideal and real resources, respectively. The ideal resource $R_{\mathrm{I}} : \mathbb{R}(\alpha + \gamma, \beta + \eta)$ provides the ideal interface and the user interface: $\mathcal{I}_{\mathrm{I}} \oplus \mathcal{I}_{\mathrm{U}} \vdash R_{\mathrm{I}}$. The real resource $R_{\mathrm{R}} : \mathbb{R}(\chi + \gamma, \delta + \eta)$ provides the real interface and the user interface: $\mathcal{I}_{\mathrm{R}} \oplus \mathcal{I}_{\mathrm{U}} \vdash R_{\mathrm{R}}$. Technically, everything depends on a security parameter η, which we leave implicit in our presentation, but which is explicit in the formalization. An advantage is negligible if, as a function of the security parameter, it approaches 0 faster than any inverse polynomial.

In the running example, the two users Alice and Bob have their own interface. So in this setting, we combine Alice's interface $\mathcal{I}_{\text{snd}}$ and Bob's $\mathcal{I}_{\text{rcv}}$ into a single channel user interface $\mathcal{I}_{\mathrm{U}} = \mathcal{I}_{\text{snd}} \oplus \mathcal{I}_{\text{rcv}}$. Similarly, when the channel resource is combined with a key resource like in Fig. 1a, parallel resource composition leads to a combined interface that is the disjoint union of disjoint unions of the adversary and user interfaces. We therefore rearrange the interfaces using wiring converters such that they obey the format "adversary interface $\oplus$ user interface"; Theorem 5 below shows the construction.

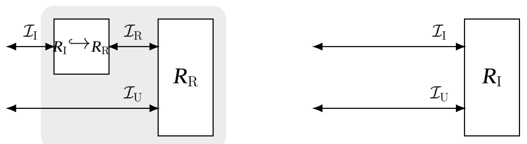
Clearly, the real resource should provide the functionality that the ideal resource promises. In general, this functionality may involve the adversary. For example, the adversary must explicitly forward the messages in our channels because the adversary controls the communication network. We therefore define when the real resource functionally realizes the ideal resource. Since the adversary interface may differ between the ideal and real resource, we assume that there is an embedding of the ideal adversary interface into the real adversary interface. We write ${}_{R_{\mathrm{I}}}\!\hookrightarrow_{R_{\mathrm{R}}}$ for the corresponding embedding converter.

In the running example, e.g., the embedding transforms the responses that return the channel's content: While the authentic channel reveals the entire message, the secure channel exposes only the length. The embedding function applies the length function to such a response of the authentic channel.

Definition 7 (Functional realisation)**.** *Let the resources R_{I} and R_{R} be given with the above interfaces and let* bound *be a natural number or infinity. We say that R_{R} functionally realises R_{I} up to* bound *interactions iff all distinguishers $D : \mathbb{A}(\alpha + \gamma, \beta + \eta)$ that respect the interface type $\mathcal{I}_{\mathrm{I}} \oplus \mathcal{I}_{\mathrm{U}}$ and that make at most* bound *queries have negligible advantage to distinguish between R_{I} and $({}_{R_{\mathrm{I}}}\!\hookrightarrow_{R_{\mathrm{R}}} \mid \mathbb{1}) \rhd R_{\mathrm{R}}$, i.e., the real resource with the ideal adversary interface.*

Figure 7 illustrates the definition: All distinguishers subject to the restrictions must have negligible advantage to distinguish the grey resource in Fig. 7a from the one in Fig. 7b. In the remainder of this section, we visualize indistinguishability in this way by showing two resources with the same interfaces.

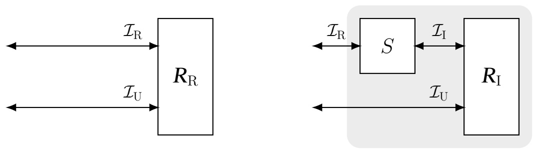
Providing the functionality alone is not enough: The real resource typically provides a richer adversary interface than the ideal resource. For a secure realisation, we demand that a


$\mathcal{I}_{\mathrm{I}}$
${}_{R_{\mathrm{I}}}\!\hookrightarrow_{R_{\mathrm{R}}}$
$\mathcal{I}_{\mathrm{R}}$
$\mathcal{I}_{\mathrm{U}}$
R_{R}
$\mathcal{I}_{\mathrm{I}}$
$\mathcal{I}_{\mathrm{U}}$
R_{I}

(a) Real resource with ideal interface

(b) Ideal resource

Fig. 7: Functional realisation of a resource.


$\mathcal{I}_{\mathrm{R}}$
$\mathcal{I}_{\mathrm{U}}$
R_{R}
$\mathcal{I}_{\mathrm{R}}$
S
$\mathcal{I}_{\mathrm{I}}$
$\mathcal{I}_{\mathrm{U}}$
R_{I}

(a) Real resource

(b) Ideal resource with simulator

Fig. 8: Secure realization of an ideal resource.

simulator S can mimick the real interface based only on the ideal. Figure 8 illustrates the indistinguishability condition.

Definition 8 (Secure realisation)**.** *Let the resource R_{R} functionally realise R_{I} up to* bound *interactions, with the interfaces as in Def. 7. We say that R_{R} securely realises R_{I} up to* bound *interactions iff there exists a simulator, i.e., a converter $S : \mathbb{C}(\chi, \delta, \alpha, \beta)$ with $\mathcal{I}_{\mathrm{R}} \vdash S \dashv \mathcal{I}_{\mathrm{I}}$ between the two adversary interfaces such that all distinguishers $D : \mathbb{A}(\chi + \gamma, \delta + \eta)$ that respect the interface $\mathcal{I}_{\mathrm{R}} \oplus \mathcal{I}_{\mathrm{U}}$ and that make at most* bound *queries have negligible advantage to distinguish between R_{R} and $(S \mid \mathbb{1}) \rhd R_{\mathrm{I}}$, i.e., the ideal resource with the simulator attached only to the adversary interface.*

Theorem 3 (Reflexivity)**.** *Every resource securely realises itself up to any bound with simulator $\mathbb{1}$ and embedding converter $\mathbb{1}$.*

C. Composability

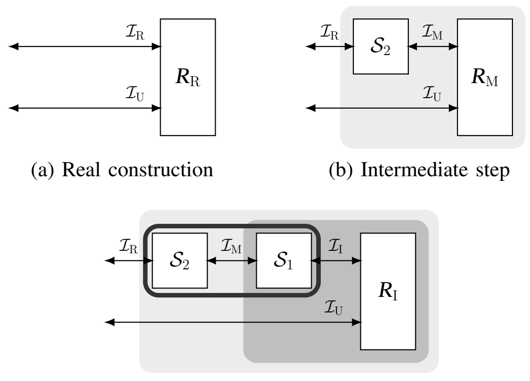
We now show that secure realisation composes. That is, secure realisation is closed under three kinds of composition:

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

We will illustrate each of these in the case study in Section VII. (Functional realisation has analogous composition theorems, which we do not present.)

Concatenation stacks secure realisations on top of each other. Suppose that R_1 securely realises R_2 and R_2 itself securely realises R_3. Then, R_1 securely realises R_3. So, secure realisation is a transitive relation. The next theorem captures this property and the associated construction is visualized in Fig. 9.

Theorem 4 (Composability)**.** *Let $\mathcal{I}_{\mathrm{I}} \oplus \mathcal{I}_{\mathrm{U}} \vdash R_{\mathrm{I}}$ and $\mathcal{I}_{\mathrm{M}} \oplus \mathcal{I}_{\mathrm{U}} \vdash R_{\mathrm{M}}$ and $\mathcal{I}_{\mathrm{R}} \oplus \mathcal{I}_{\mathrm{U}} \vdash R_{\mathrm{R}}$ such that R_{M} securely realises R_{I} up to* bound$_1$ *interactions and R_{R} securely realises R_{M} up to* bound$_2$

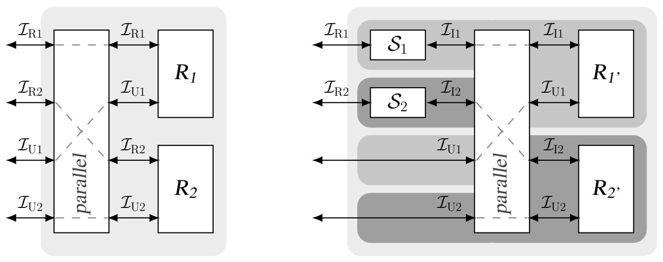


$\mathcal{I}_{\mathrm{R}}$
$\mathcal{I}_{\mathrm{U}}$
R_{R}
$\mathcal{S}_2$
$\mathcal{I}_{\mathrm{M}}$
R_{M}
$\mathcal{S}_1$
$\mathcal{I}_{\mathrm{I}}$
R_{I}

(a) Real construction (b) Intermediate step

(c) Ideal construction

Fig. 9: Composability of secure realizations. The real resource R_{R} is constructed from the middle resource R_{M} using converter S_2. R_{M} is constructed from the ideal resource R_{R} using converter S_1. Thefore, R_{R} can be directly constructed from R_{I} using the sequential composition of of S_1 and S_2, depicted using bold black rectangle.



$\mathcal{I}_{\mathrm{R1}}$
$\mathcal{I}_{\mathrm{R2}}$
$\mathcal{I}_{\mathrm{U1}}$
$\mathcal{I}_{\mathrm{U2}}$
parallel
R_1
R_2
$\mathcal{S}_1$
$\mathcal{S}_2$
$\mathcal{I}_{\mathrm{I1}}$
$\mathcal{I}_{\mathrm{I2}}$
$R_{1'}$
$R_{2'}$

(a) Real Construction (b) Ideal construction

Fig. 10: Parallel composition of secure realizations.

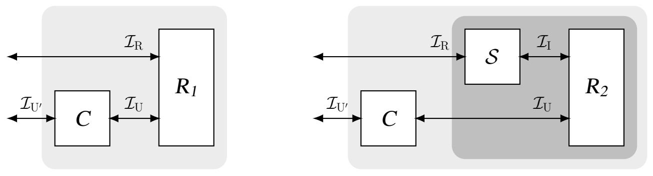
interactions. Let S_1 and S_2 be the respective simulators and bound_S *be a bound on the number of queries that S_2 makes during one invocation. If* $\mathsf{bound}_2 * \mathsf{max}(\mathsf{bound}_S, 1) \leq \mathsf{bound}_1$*, then* R_{R} *securely realises* R_{I} *up to* bound_1 *interactions with simulator* $S_2 \odot S_1$.

Moreover, secure realisation is closed under parallel composition (Fig. 10).

Theorem 5 (Parallel composition)**.** *Let R_1 securely realise R'_1 up to* bound_1 *interactions and let R_2 securely realise R'_2 up to* bound_2 *interactions. Let* $\mathsf{parallel}$ *denote the converter* $\mathsf{lassocr} \odot (\mathbb{1} \mid (\mathsf{rassocl} \odot (\mathsf{swap} \mid \mathbb{1}) \odot \mathsf{lassocr})) \odot \mathsf{rassocl}$*. Then, the parallel composition* $\mathsf{parallel} \odot (R_1 \mid R_2)$ *securely realises* $\mathsf{parallel} \odot (R'_1 | R'_2)$ *up to* $\mathsf{min}(\mathsf{bound}_1, \mathsf{bound}_2)$ *interactions. The new simulator is the parallel composition of the old simulators.*

Together with Thm. 3, Thm. 5 yields that secure realisation is preserved under parallel composition contexts, i.e., if we add an arbitrary resource in parallel to a real and an ideal resource, then the resulting real resource securely realises the resulting ideal resource.

Third, secure realisation is preserved when we attach a converter to the user interfaces (Fig. 11). There is no corresponding theorem for attaching a converter to the adversary interface. Changes to the adversary interface must be expressed as a composition of secure realisations (Thm. 4).



$\mathcal{I}_{\mathrm{R}}$
$\mathcal{I}_{\mathrm{U'}}$
$\mathcal{I}_{\mathrm{U}}$
C
R_1
$\mathcal{S}$
$\mathcal{I}_{\mathrm{I}}$
R_2

(a) Real Construction (b) Ideal construction

Fig. 11: Attaching a converter to the user interface

Theorem 6 (Secure attachment)**.** *Let R_1 securely realise R_2 up to* bound *interactions. Let C be a converter on the user interface that performs at most* bound_C *many interactions in a single invocation. Then,* $(\mathbb{1} \mid C) \rhd R_1$ *securely realises* $(\mathbb{1} \mid C) \rhd R_2$ *up to* bound' *interactions if* $\mathsf{bound}' * \mathsf{max}(\mathsf{bound}_C, 1) \leq \mathsf{bound}$ *with the same simulator and embedding converter.*

VII. Wrapping-up the Running Example

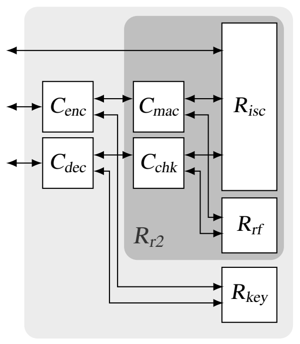
We now use our framework to formalize the construction of a secure channel from a key, a random function, and an insecure channel. For brevity, we only provide a high-level summary and use pictures to present the main proof steps. Similar to the Section IV, the visualisations are a direct translation of the formal text, i.e. they have corresponding syntax and semantics, and are thus meaningful. The readers can refer to the online source [25] for more details.

In each subfigure in the Figure 12, resources are depicted on the right, and the attachment of abstract systems $\rhd$ is presented using arrows. Moreover, the resources or converters that are in the same column are composed in parallel using $\|$ or $|$ respectively. Following the lemmas about the sequential composition $\odot$ of converters in the Section IV-C, the arrow connecting two converters, e.g. the arrow between $\mathcal{S}_2$ and $\mathcal{S}_1$, can be understood as either the attach operation or the sequential composition of converters.

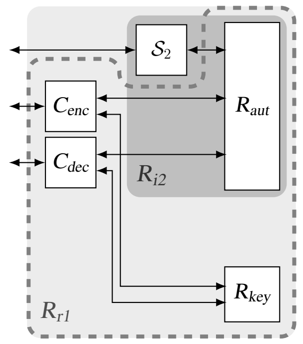
We use two cryptographic schemes in our construction. First, an encryption scheme with converters C_{enc} and C_{dec} that is used to construct a secure channel from an authentic channel and a key. Second, a message authentication code (MAC) scheme with converters C_{mac} and C_{chk} that is used to construct an authentic channel from an insecure channel and a random function.[5] We have formalized these schemes in a generic way that admits multiple instantiations. We instantiate the encryption scheme with a one-time-pad, and we use the random function's mapping of each message as the message authentication code.

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

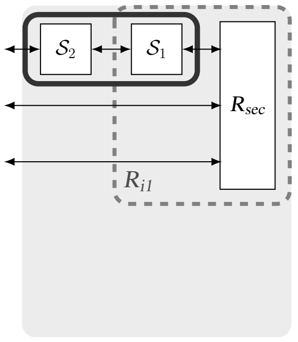
[5]It is possible to construct these resources differently. For instance, an authentic channel can be constructed from a key, an unpredictable function, and an insecure channel.



(a) Construction using an insecure channel



(b) Replacing an authentic channel



(c) Replacing a secure channel

Fig. 12: Composing two secure constructions using the composition theorem.

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

In the first step of the proof, we use the notion of trace equivalence (Def. 5) to prove that the encryption scheme, instantiated with one-time pad, securely constructs a secure channel from a key and an authentic channel. That is, we show that a distinguisher, subject to the side conditions of our security definition in the Section VI, cannot distinguish between resources R_{r1} and R_{i1}, the areas with dashed lines in the Figure 12. R_{r1} is the resource resulting from attaching C_{enc} and C_{dec} to R_{aut} and R_{key}; and R_{i1} is the resource resulting from attaching $\mathcal{S}_1$ to R_{sec}. For the one-time-pad encryption, the simulator $\mathcal{S}_1$ generates random bit-strings of a given length obtained from querying the secure channel R_{sec}.

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

In the next step, we prove the security of our simple MAC scheme by showing that it securely constructs an authentic channel from a random function and an insecure channel. This construction is presented using dark gray areas in Figure 12. Let R_{r2} denote the resource resulting from attaching C_{mac} and C_{chk} to R_{isc} and R_{rf}, and let R_{i2} be the resource resulting from attaching the simulator $\mathcal{S}_2$ to R_{aut}. We prove the indistinguishability of R_{r2} and R_{i2} in three intermediate steps. (i) We show the trace equivalence of R_{r2} with its lazy variant. The main difference between the standard (eager) and the lazy systems is the way in which they treat MACs: in a lazy-R_{r2}, messages are inserted into the channel without generating a MAC and the receiver does not check MACs by default; however, in a scenario where an adversary looks into the channel's content, a MAC is generated on the fly and the receiver is triggered to check its correctness later. This preparatory step aligns the samplings in the real system with those in the ideal system with the simulator, where the simulator generates a fake MAC on the fly. (ii) We define a restricted variant of lazy-R_{r2} with a special random function that avoids generating the authentication codes queried by the adversary, and we show that any distinguisher has negligible advantage for distinguishing the restricted and non-restricted instances of lazy-R_{r2}. Here, the requirement for the notion of indistinguishability with negligible advantage becomes clear: neither the bisimulation nor the trace-equivalence notions are sufficient for this step since the two systems are actually not equal. In the final intermediate step, (iii) we show that the restricted-lazy-R_{r2} and R_{i2} are trace equivalent. Hence, using the triangular inequality, we can combine the three steps and show that R_{r2} and R_{i2} are indistinguishable.

In the final proof step, we just instantiate our composition theorem, i.e. Theorem 4 with the two aforementioned constructions. Figure 12 depicts how the composition theorem combines the two constructions.

VIII. Discussion and Related Work

In this section, we discuss how our formalisation relates to Constructive Cryptography and to the existing approaches to computer-aided cryptography.

Constructive cryptography [26] models resources as random systems [28], i.e., families of conditional probability distributions. The trace of a resource is exactly a random system. Conversely, every random system is the trace of some resource. The novel part here is the recursive definition of a trace (Def. 5), which gives rise to an unwinding rule for random systems (Thm. 1). This notion of traces has been inspired by abstract theory of modelling systems coalgebraically. This provides evidence that we have found the right coalgebraic model for Constructive Cryptography.

Maurer and Renner [27] already noted that the theory of random systems supports only a single interface but constructive cryptography needs support for multiple interfaces. Our theory of interfaces and interface types provides this extension. We can safely combine several interfaces into one and rearrange them as needed with wiring converters. The crucial insight here is that an interface type is like a dependent function type: The set of possible responses depends on the concrete input value. This allows us to express that the response will be sent on the same interface as the query.

Our proof that trace equivalence captures perfect indistinguishability (Thm. 2) shows that bisimulations are necessarily incomplete for proving indistinguishability. This incompleteness is practically relevant, as our case study shows. Typical cryptographic arguments like switching between lazy and eager sampling cannot be handled by bisimulations. This restriction is particularly relevant for simulation-based proofs, because the switch happens in a context that is controlled by an unknown adversary. This insight applies not only to Constructive Cryptography, but also to other approaches to computer-aided cryptography and relational reasoning about probabilistic programs in general.

Let us expand on this last point with several examples. EasyCrypt [3] and CertiCrypt [2] develop a separate theory for code motion. Our result shows that there is no way around that: probabilistic relational Hoare logic (pRHL) [5] establishes bisimilarity and therefore cannot justify code motion in a compositional way. Recently, Barthe et al. [4] wondered whether their extension of pRHL is complete for hoisting random assignment out of loops. Our result gives a negative answer: they establish bisimilarity, which is too strong.

To make this precise, let $\vDash c_1 \sim c_2 : \Phi \Rightarrow \Psi$ denote that if we start the two probabilistic programs c_1 and c_2 in the initial memories m_1 and m_2, respectively, such that $(m_1, m_2) \in \Phi$, then the probability distributions d_1 and d_2 over the final memories are related in the lifted relaton *lift*(Ψ).[6] Here, two distributions d_1 and d_2 are in the lifted relation *lift*(R) iff there is a joint distribution d with support contained in R such that d_1 and d_2 are the two marginal distributions of d. (Note that bisimilarity in Def. 4 uses a special case of lifting for the bisimulation relation X.) Following traditional relational Hoare logic [8], pRHL consists of a set of compositional, syntax-directed rules for judgements $\vdash c_1 \sim c_2 : \Phi \Rightarrow \Psi$ that establish $\vDash c_1 \sim c_2 : \Phi \Rightarrow \Psi$. For example, the pRHL rule for sequential composition $c;\ c'$ is as follows:

$$\frac{\vdash c_1 \sim c_2 : \Phi \Rightarrow \Xi \qquad \vdash c_1' \sim c_2' : \Xi \Rightarrow \Psi}{\vdash c_1;\ c_1' \sim c_2;\ c_2' : \Phi \Rightarrow \Psi}.$$

Here, Ξ relates the intermediate memories, which are the outputs of running c_1 and c_2 that become the inputs for c_1' and c_2'. Rules of this shape cannot justify cryptographic arguments based on code motion like lazy sampling where $c_1;\ c_1'$ eagerly samples a value in c_1, whereas $c_2;\ c_2'$ does so lazily in c_2'. This is because Ξ relates individual intermediate memories instead of conditional distributions on intermediate memories. In fact, the shape of pRHL judgements cannot accomodate relations between conditional distributions due to the lifting operator *lift*. In practice, pRHL's rules are therefore extended with other proof principles like equational reasoning for code motion.

The same argument applies to CryptHOL's relational proof rules, which Basin et al. [6] derive from relational parametricity, and to FCF's rules [30]: They establish bisimilarity, not trace equivalence. In contrast, our proof rule (Thm. 1) establishes trace equivalence and is therefore complete. In our case study, we use this rule to hoist the random assignment (the one-time-pad key) out of the key resource and into the simulator.

The composition theorems (Thms. 3-5) show that our formalisation meets the requirements for general composability (Section II-B). This is an improvement to the state of the art where there are only few results on formally verified composable cryptographic proofs. Existing formal-methods tools mainly focus on game-based proofs, which in general are not composable. There are some results about specific class of protocols' composability in such tools. For example, Blanchet [11] has formalized CryptoVerif's composition theorems to compose a key providing protocol and a protocol that uses this key and utilized them in TLS 1.3's formal verification. Furthermore, there are few results on formalized simulation-based proofs [12], [17]. They do not formalize (or report on) any composability statement, yet the individual protocols that are formalized in these works should be composable in the context of a simulation-based frameworks such as Universal Composability.

IX. Conclusion and Future Work

We have presented the semantic foundation of our framework and demonstrated that it supports the formalization of composable cryptographic security statements. We work in an asymptotic information-theoretic security setting that makes the general composability statements more concise. Nevertheless, our approach would also work in a concrete security setting. Computational security statements and more complex case-studies are left as future work.

Acknowledgments

We thank Christoph Sprenger, Esfandiar Mohammadi, Christian Badertscher, and Fabio Banfi for their helpful discussions and feedback.

[6]Results and outputs of the program are written to special memory locations and therefore need no special treatment.

REFERENCES

[1] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *Theory of Cryptography (TCC 2004), Proceedings*, volume 2951 of *LNCS*, pages 336–354. Springer, 2004.

[2] G. Barthe, B. Grégoire, and S. Z. Béguelin. Formal certification of code-based cryptographic proofs. In *Principles of Programming Languages (POPL 2009), Proceedings*, pages 90–101. ACM, 2009.

[3] 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.

[4] G. Barthe, B. Grégoire, J. Hsu, and P.-Y. Strub. Coupling proofs are probabilistic product programs. In *Principles of Programming Languages (POPL 2017), Proceedings*, pages 161–174. ACM, 2017.

[5] G. Barthe, B. Grégoire, and S. Zanella Béguelin. Probabilistic relational hoare logics for computer-aided security proofs. In *Mathematics of Program Construction (MPC 2012), Proceedings*, volume 7342 of *LNCS*, pages 1–6. Springer, 2012.

[6] D. A. Basin, A. Lochbihler, and S. R. Sefidgar. CryptHOL: Game-based proofs in higher-order logic. *IACR Cryptology ePrint Archive*, 2017:753, 2017.

[7] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology (EUROCRYPT 2006), Proceedings*, volume 4004 of *LNCS*, pages 409–426. Springer, 2006.

[8] N. Benton. Simple relational correctness proofs for static analyses and program transformations. In *POPL 2004*, pages 14–25, New York, NY, USA, 2004. ACM.

[9] M. Berg. *Formal Verification of Cryptographic Security Proofs*. PhD thesis, Saarland University, 2013.

[10] B. Blanchet. A computationally sound mechanized prover for security protocols. In *Security and Privacy (S&P 2006), Proceedings*, pages 140–154. IEEE Computer Society, 2006.

[11] B. Blanchet. Composition theorems for cryptoverif and application to TLS 1.3. In *Computer Security Foundations (CSF 2018), Proceedings*, pages 16–30. IEEE Computer Society, 2018.

[12] 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.

[13] 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.

[14] D. Coumans and B. Jacobs. Scalars, monads, and categories. In *Quantum Physics and Linguistics: A Compositional, Diagrammatic Discourse*, pages 184–216. Oxford University Press, 2013.

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

[16] M. J. C. Gordon and A. M. Pitts. Chapter 3 - the hol logic and system. In *Towards Verified Systems*, volume 2 of *Real-Time Safety Critical Systems*, pages 49–70. Elsevier, 1994.

[17] 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.

[18] S. Halevi. A plausible approach to computer-aided cryptographic proofs. *IACR Cryptology ePrint Archive*, 2005:181, 2005.

[19] I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4):1–36, 2007.

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

[21] B. Jacobs, A. Silva, and A. Sokolova. Trace semantics via determinization. *Journal of Computer and System Sciences*, 81(5):859–879, 2015.

[22] A. Kurz, S. Milius, D. Pattinson, and L. Schröder. Simplified coalgebraic trace equivalence. In *Software, Services, and Systems*, volume 8950 of *LNCS*, pages 75–90. Springer, 2015.

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

[24] 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.

[25] A. Lochbihler and S. R. Sefidgar. Constructive Cryptography using CryptHOL. *Archive of Formal Proofs*, 2018. http://isa-afp.org/entries/Constructive_Cryptography.html.

[26] 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.

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

[28] U. M. Maurer. Indistinguishability of random systems. In *Advances in Cryptology (EUROCRYPT 2002), Proceedings*, volume 2332 of *LNCS*, pages 110–132. Springer, 2002.

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

[30] A. Petcher and G. Morrisett. The foundational cryptography framework. In *Principles of Security and Trust (POST 2015), Proceedings*, volume 9036 of *LNCS*, pages 53–72. Springer, 2015.

[31] J. J. M. M. Rutten. Universal coalgebra: A theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[32] V. Shoup. Sequences of games: A tool for taming complexity in security proofs. *IACR Cryptology ePrint Archive*, 2004:332, 2004.

[33] A. Silva, F. Bonchi, M. M. Bonsangue, and J. J. M. M. Rutten. Generalizing the powerset construction, coalgebraically. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010), Proceedings*, volume 8 of *LIPIcs*, pages 272–283. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

APPENDIX

A. Guide to the source theory files

For brevity, in the body of this paper, we only provided informal text and used pictures to present the main lemmas and theorems. However, all the definitions and lemmas are formalized and verified in the Isabelle/HOL proof assistant. In what follows, we provide a guide for the reader to navigate the source theories, which are available online [25].

The root directory contains many theory files and an **`Examples`** folder that stores the case study's formalization. Each theory file contains the lemmas and definitions which correspond to its name:

- **`Resource.thy`** formalizes resources, their parallel composition, and the notion of interface respecting resources (Section IV-A).
- **`Converter.thy`** formalizes converters, their sequential and parallel composition, the notion of interface respecting converters, and the attachment of converters to resources (Sections IV-B, IV-C).
- **`Wiring.thy`** formalizes the wiring converters (Section IV-C).
- **`Random_System.thy`** formalizes the notion of trace and trace equivalence, where the propositions `trace'_eqI_sim` and `trace_callee_complete` prove the two directions of Theorem 1.
- **`Distinguisher.thy`** formalizes the notion of distinguishers, where lemmas `connect_cong_trace` and `distinguish_trace_eq` formalize the two directons of Theorem 2.
- **`Constructive_Cryptography.thy`** formalizes the notion of secure realisation (Def. 8) using the locale `constructive_security`. The

composability theorems 3–6 are formalized by the theorems `constructive_security_trivial`, `composability`, `parallel_constructive_-security`, and `lifting` respectively.
- **`Converter_Rewrite.thy`** formalizes the notion of equivalence of resources and converters subject to assumptions on the context given by interface types. The paper does not delve into this technicality.

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

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