Abstract Modeling of Systems Communication in Constructive Cryptography using CryptHOL

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

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

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

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

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

In this paper, we focus on the modeling of systems communication in composable security statements. The existing formal models consider fixed communication patterns to reduce the complexity of their proofs. However, as we will show, this can affect the reusability of security statements. We propose an abstract approach to modeling systems communication in Constructive Cryptography that avoids this problem. Our approach is suitable for mechanized verification and we use CryptHOL, a framework for developing mechanized cryptography proofs, to implement it in the Isabelle/HOL theorem prover. As a case study, we formalize the construction of a secure channel using the Diffie-Hellman key exchange and a one-time-pad.

Note to reviewers: All results presented in this paper are formalized using Isabelle/HOL. We provide this formalization as a supplementary material, which is available online [1], and explain its structure in Appendix A.

I. INTRODUCTION

A. Problem context

Simulation-based frameworks [2], [7], [11], [13], also called composability frameworks, pave the way for modular reasoning about cryptographic constructs independently of their application context. They follow the Ideal-world—Real-world [9] paradigm to security proofs, where a distributed protocol’s security is defined by comparing its execution, i.e., the real world, to an idealized specification that intrinsically satisfies the desired properties.

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

Simulation-based frameworks’ approach to modeling systems communication is one of the main reasons for the complexity of their resulting proofs. In these frameworks, components communicate via network tapes and a central scheduler called the adversary Turing machine. This concrete approach increases the complexity of ideal specifications and makes the security arguments intricate [6]. As such, the existing formal results such as EasyUC [8] consider a simplified version of the simulation-based frameworks with restricted communication capabilities between components. However, as we will show in Section III-A, such simplifications affect the reusability of security statements.

Constructive Cryptography (CC) [16], [17], [18] proposes a fundamental shift in how security statements are made and proved. It introduces an abstract approach to composable security arguments that allows one to focus on a particular aspect of security proofs without being distracted by other details. This makes composable security statements manageable for protocol designers. However, the existing CC results [12], [15] do not delve into the details of systems communication.

B. Our solution

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

First, we propose a semantic domain called Fused Resource Templates (FRT) that abstracts over the systems communication patterns. FRTs are suitable for mechanizing composable security proofs, and we formalize all the results presented in this paper using CryptHOL [4], [14], a framework for mechanizing cryptographic proofs in the proof assistant Isabelle/HOL [19].

Second, we show how FRTs constitute an instantiation of the CC framework. We formalize operators that enable us to compose FRTs and thereby define complex FRTs in terms of simpler ones. We then introduce the notion of secure constructions using FRTs and provide composition theorems for such constructions. Finally, we demonstrate the practicality of our approach in a case study. We formalize the construction
of a secure channel by composing the Diffie-Hellman key exchange with one-time-pad encryption.

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

C. Contributions

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

- We propose the first CC instantiation that explicates the details of systems communication. The role of systems communication models has not been studied in the CC literature, although we show that reusability and modularity of CC proofs depends on it (cf. Section III-A). We propose an abstract approach to capturing systems communication patterns in CC proofs.
- We formalize our approach in a theorem prover and provide theorems for security proof compositions. In Section V, we present our case study to demonstrate our approach’s applicability. Finally, in Sections VI and VII we compare with related work and draw conclusions.

D. Structure

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

II. Preliminaries

This section introduces background that is used in the rest of the paper. We review the compositability of security proofs and how it is treated in simulation-based frameworks and CC. We focus on system communication modeling in composable security arguments. In each subsection, we use the notation that is common among the research community for the topic under discussion.

A. Composable security arguments

The ideal-world—real-world [8] approach to security enables modular reasoning about security statements. In this approach, a protocol’s security is defined by comparing its execution, i.e., the real world, to an idealized specification that satisfies the desired properties by definition: a protocol $\pi$ realizes (or implements) an idealized functionality $F$ if there exists a simulator that can simulate $\pi$’s behavior (in an adversarial environment) by interacting with $F$. The so-called Composability Theorems extend the above idea to an arbitrary application context. Consider a protocol $\rho|F$, where the $|$ operator denotes protocol composition, that realizes an ideal functionality $G$; then the protocol $\rho|\pi$ also realizes $G$ if the protocol $\pi$ realizes the ideal functionality $F$.

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

B. UC-style frameworks

In simulation-based frameworks [2, 7, 11, 13], (sub-) protocols are modeled using interactive Turing machines with network tapes and unique identifiers. At each point of a protocol’s execution, only one of these machines is active and the others wait for new inputs. A special Turing machine, called the adversary, schedules the activation of protocol components: it is activated after each non-adversary machine halts and determines the next Turing machine to activate. The adversary plays the role of a mediator too. When two machines want to communicate, the sender informs the adversary about the message’s content and destination, i.e., the receiving Turing machine’s identifier, and the adversary forwards the message according to its underlying corruption model.

The generality of the communication model in simulation-based frameworks stems from the central adversary Turing machine that is universally quantified in security definitions. Consider the compound protocol $\rho|F$, and interpret $|$ according to the execution model explained above, $\rho$’s subroutine calls to $F$ result in a transfer of the execution flow that is essential for the completion of $\rho$’s execution. The subroutine call is carried out by two message broadcasts: the first message carries the “request” details while the second one conveys the “response” information. These messages need not be passed consecutively; therefore, each of $\rho$’s subroutine calls may correspond to an arbitrary sequence of Turing machine activations starting with

1A message can be any information that is transferred between protocol components, including meta-level information, subroutine calls, or protocol-specific data.
A specification is a set of resources. Parties can change frameworks, where it refers to one or more Turing machines working together.

The simulation-based frameworks’ concrete approach to modeling can affect the rigor of their resulting proofs. As Camenisch et al. put it \[5\], protocol descriptions in simulation-based frameworks include meta-level and model specific information that makes them unnecessarily complicated; but, ignoring such details can lead to ill-defined specifications, sketchy proofs, and flawed results. The existing formal methods tools have not been able to alleviate the situation either. The level of details in simulation-based frameworks surpasses what the formal-methods community can reasonably handle with existing techniques. As such, the existing formalization results consider stripped-down versions of simulation-based frameworks by restricting the communication and corruption models \[8\]. Hence, they are not applicable in a wide range of scenarios.

### C. Constructive Cryptography

Constructive Cryptography (CC) \[16\], \[17\], \[18\] is an abstract paradigm for developing a theory of cryptography. Instead of focusing on concrete systems and proving their properties, CC studies system classes, i.e., the shared behavior of similar systems, and their transformations. As such, the notions of runtime, algorithms, and complexity are not intrinsic parts of every definition and proof in CC; they just represent classification strategies.

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

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

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

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

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

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

### III. A NEW CC SEMANTIC DOMAIN

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

### A. Motivation

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

We start by considering the impact of singleton specifications on the modularity and reusability of security proofs. The point of compositability is that the realization of a specification can be decomposed into small and easy-to-understand steps, which can then be combined modularly. For example, in Figure 1 we may combine $\pi$ with various key-exchange protocols to obtain different realizations of a secure channel; however, we want to prove $\pi$’s security only once. That is, we do want to abstract away from the details of the key-exchange protocol in $\pi$’s proof.

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

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

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

### B. Fused Resource Templates

We define FRTs in terms of Lochbihler et al. [15]’s formalization of resources. Each FRT describes a parametrized resource: all instances share a common behavior while each instance exhibits a particular party-activation pattern.

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

Lohcböhler et al. [15] hide the internal state of resources to simplify the proofs and definitions. The resource type existentially quantifies over the state type in the pair of the transition function and the initial state:

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

where $\alpha$ represents the type of inputs, $\beta$ the type of outputs, $\times$ the product type, $\Rightarrow$ the function space, and $\mathbb{D}(\gamma)$ probability distributions over $\gamma$. The Isabelle/HOL formalization uses co-inductive datatypes, in short co-datatypes, instead of existential types. Let $e : \chi$ denote that expression $e$ has type $\chi$; the resource type is the final co-algebra for the transition function $\text{run}: \mathbb{R}(\alpha, \beta) \Rightarrow \alpha \Rightarrow \mathbb{D}(\beta \times \mathbb{R}(\alpha, \beta))$ that returns a distribution on the output and the successor resource for a given input. Finally means two resources are equal if their input-output behaviors are bisimilar. The following co-datatype formalizes this:

**codatatype** \( \mathbb{R}(\alpha, \beta) = \text{Resource}(\alpha \Rightarrow \mathbb{D}(\beta \times \mathbb{R}(\alpha, \beta))). \)

The existing formalizations [4, 15] introduce two more components that characterize the definition of resources with multiple interfaces. First, CryptHOL’s oracle composition operator $\odot$ interrelates two transition functions $\text{tr}_i : \sigma \Rightarrow \alpha_i \Rightarrow \mathbb{D}(\beta_i \times \sigma)$ (for $i = 1, 2$) operating on a shared state of type $\sigma$ into one transition function $\text{tr}_1 \odot \text{tr}_2 : \sigma \Rightarrow \alpha_1 + \alpha_2 \Rightarrow \mathbb{D}((\beta_1 + \beta_2) \times \sigma)$, where $\alpha + \beta$ denotes the disjoint union of the types $\alpha$ and $\beta$. Following the CryptHOL terminology, we use the terms “oracle” and “probabilistic transition function” interchangeably in this paper. Second, the function $\text{RES}$ converts an oracle into a resource, i.e., it introduces the existential type quantifier and hides the state. $\text{RES}(\text{tr}, s_0)$ is the co-inductive counterpart of a probabilistic transition system where the state is explicit, represented using the transition function $\text{tr}$ and the initial state $s_0$.

2) **Fused Resources:** FRT instances correspond to particular members of a resource family, but how do we know if an instance belongs to a family? Remember that family members share a common behavior while each of them will exhibit a particular behavior too. FRT’s only focus on the common behaviors and treat the particular ones abstractly as a parameter in their description. Nevertheless, we must ensure that every instantiation of such parameter will not affect the “common behavior”. For example, consider the specification for secure channels. The common behavior of every resource in the family of secure channels is that they would answer queries within some fixed time. So, the FRT describing this family must prohibit any instantiation that can leak more information.

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

The Core part is implemented as a record $\text{core} \equiv \langle \text{cinit} :: \cdots, \text{cpoke} :: \cdots, \text{cfunc} :: \cdots \rangle$ with three fields: an initial Core state $\text{cinit}$, a probabilistic event handler $\text{cpoke} : \sigma \Rightarrow \varepsilon \Rightarrow \mathbb{D}(\sigma)$, and a probabilistic transition function $\text{cfunc} : \sigma \Rightarrow \alpha \Rightarrow \mathbb{D}(\beta \times \sigma)$. The $\text{cpoke}$ field is an event handler that defines a notification mechanism: given the current Core state and an event, it defines a probability distribution on the successor state. The $\text{cfunc}$ field describes the input-output behavior of the interfaces: given the current Core state and an input, it defines a probability distribution on the pair of the output and the successor state.

The Rest part communicates with the Core ones via poke events. They are defined in terms of a record $\text{rest} \equiv \langle \text{rinit} :: \cdots, \text{rfunc} :: \cdots \rangle$ that consists of an initial Rest state $\text{rinit}$, a probabilistic event-augmented transition function $\text{rfunc} : \tau \Rightarrow \chi \Rightarrow \mathbb{D}((\delta \times L(\varepsilon)) \times \tau)$, where $L(\varepsilon)$ denotes the type of events list. The $\text{rfunc}$ field defines the input-output behavior of Rest interfaces as well as the information they leak to the Core part: given the current Rest state and an input, it defines a probability distribution on the output, the successor rest state, and a list of events that the Fused Resource’s core part will be notified about.

Given a $\text{rest} \equiv \langle \text{rinit} :: \text{rinit}_\text{def}, \text{rfunc} :: \text{rfunc}_\text{def} \rangle$ and a $\text{core} \equiv \langle \text{cinit} :: \text{cinit}_\text{def}, \text{cpoke} :: \text{cpoke}_\text{def}, \text{cfunc} :: \text{cfunc}_\text{def} \rangle$ with the types defined above, the Fusing function $\text{fuse}(\text{core}, \text{rest})$ outputs a probabilistic transition function with type $(\sigma \times \tau) \Rightarrow (\alpha + \chi \Rightarrow \mathbb{D}((\beta + \delta) \times (\sigma \times \tau)))$ that enforces the behavior of the interfaces as well as the information they leak to the Core part: given the current Rest state and an input, it defines a probability distribution on the output, the successor rest state, and a list of events that the Fused Resource’s core part will be notified about.
it from typical resources of the same type.

Core records suffice to describe the behavior that is common to every instantiation of an FRT. Let ♠ and ♣ respectively denote the types of core and rest records mentioned above. Given a core core : ♠, an FRT core ⊢ rest : ♠ ⇒ F R(α, χ, β, δ) is a function from Rest records to Fused Resources such that:

\[ \text{core}(\text{rest}) \equiv \text{FUSE}(\text{core}, \text{rest}). \] (1)

When essential, we specify the type of suitable Rest records using a subscript core.

3) A concrete example: We now explain how FRTs enable the modeling of specifications that include resources with arbitrary interfaces and activation patterns. The main idea is to define Core such that it returns ⊥ when an unwanted sequence of poke events has been received.

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

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

The probabilistic transition function cf_{key} ≡ tr_{Event} +₀ tr_{Alice} +₀ tr_{Bob} describes the interfaces for Alice, Bob, and the adversary, also called Eve, that are composed using the oracle composition operator. The adversary interface’s functionality tr_{Event} leaks no information to Eve; it is a dummy oracle that does not change the state and answers every input with a unit value ⊙. The interface functionality for Alice is defined using tr_{Alice} which outputs a key k upon a unit input ⊙ if the state is a pair (Hold k, shell) such that Alice ∈ shell. Queries to tr_{Alice} will invoke the failure mechanism if the state does not satisfy the aforementioned conditions. Bob’s interface functionality tr_{Bob} is the same as Alice’s interface except that it checks for Bob ∈ shell upon receiving queries.

Figure 2 is a pictorial representation of the ideal specification for keys. key represents all Fused Resources that accept queries on Alice and Bob interfaces after one or more queries to the i_1, · · · , i_n interfaces, which trigger the key generation event and enable (at least) one of the parties’ interfaces. Either Alice or Bob may attempt to receive the key first; it is only important that their corresponding interface has been enabled during one of the event triggers.

C. Constructions using FRTs

Defining complex systems in terms of simpler ones is the essence of composable security statements. CC captures this using the concepts of simultaneously-available resources and converter attachments, which are lifted to specifications as explained in Section II-C. That is, compound resources describing complex probabilistic systems can be defined as the attachment of simpler building blocks, i.e., converters and simpler resources. We show how these concepts are expressed using FRTs. Our formalization reuses the converters and resources from Lochbihler et al.’s [15].

1) A short recap of compound resources: Converters are probabilistic reactive systems that internally interact with other reactive systems. A converter has two types (or classes) of interfaces. First, the external interfaces are used to receive inputs and send outputs. Second, computing an output may invoke many interactions on the internal interfaces. A converter appears like a resource when one of its external interfaces is queried, however, such a query may invoke other converters and resources prior to its response. A converter’s internal interface is formalized as a CryptHOL [4] Generative Probabilistic Value (GPV) G(ξ, γ, ρ): a GPV produces results of type ξ by interacting with arbitrary reactive systems that answer queries of type γ with responses of type ρ. Formally,

\[ \text{codatatype } G(\xi, \gamma, \rho) = \text{GPV} \sqcup (\xi + (\gamma \times (\rho \Rightarrow G(\xi, \gamma, \rho)))) . \]

A converter then is modeled using an initial state s_0 : σ and a transition function tr : σ ⇒ α ⇒ G(β × σ, γ, ρ) that describes a GPV on the output and the new state of the converter given its current state and an input. Similar to resources, Lochbihler et al. [15] provide an operator CNV(tr, s_0) that can hide a converter’s internal state and represent it as a value of the following co-datatype:

\[ \text{codatatype } C(\alpha, \beta, \gamma, \rho) = \text{Converter } (\alpha \Rightarrow G(\beta 

\times C(\alpha, \beta, \gamma, \rho), \gamma, \rho)). \]
As shown in Figure 3, attaching $C : \text{C}(\alpha, \beta, \gamma, \rho)$ to a resource $R : \text{R}(\gamma, \rho)$, which means $R$ responds to $C$’s internal queries, creates a new resource $C \triangleright R : \text{R}(\alpha, \beta)$, which is represented as the surrounding gray rectangle.

Lochbihler et al. model multiple interfaces $i_1, \ldots, i_n$, which answer queries of types $\alpha_1, \ldots, \alpha_n$ with responses of types $\beta_1, \ldots, \beta_n$, using disjoint unions as a single interface with inputs of type $\alpha_1 + \cdots + \alpha_n$ and outputs of type $\beta_1 + \cdots + \beta_n$. So interfaces are identified by their position in the sum type. The attachment of (multiple) converters to a subset of (simultaneously available) resources’ interfaces are defined in terms of the following operators:

- Let $R_1 : \text{R}(\alpha_1, \beta_1)$ and $R_2 : \text{R}(\alpha_2, \beta_2)$ be resources. Their parallel composition $R_1 \parallel R_2 : \text{R}(\alpha_1 + \alpha_2, \beta_1 + \beta_2)$ directs queries $\alpha_1$ to $R_1$ and $\alpha_2$ to $R_2$ and forwards the responses accordingly.

- Let $C_1 : \text{C}(\alpha_1, \beta_1, \gamma_1, \eta_1)$ and $C_2 : \text{C}(\alpha_2, \beta_2, \gamma_2, \eta_2)$ be two converters. Their parallel composition $C_1 \uplus C_2 : \text{C}(\alpha_1 + \alpha_2, \beta_1 + \beta_2, \gamma_1 + \gamma_2, \eta_1 + \eta_2)$ makes both converters available at the same time, analogous to parallel resource composition.

- Let $C_1 : \text{C}(\alpha, \beta, \gamma, \eta)$ and $C_2 : \text{C}(\gamma, \eta, \chi, \delta)$ be two converters. Their sequential composition $C_1 \otimes C_2 : \text{C}(\alpha, \beta, \chi, \delta)$ uses $C_2$ to answer $C_1$’s queries on $C_1$’s internal interface.

- The identity converter $1 : \text{C}(\alpha, \beta, \alpha, \beta)$ simply forwards all queries and responses from the external to the internal interface and vice versa.

We provide an example to clarify how the above operators are used. Consider resource $R_1$ with one interface $x$, resource $R_2$ with two interfaces $y$ and $z$, and a converter $C$ with an external interface $a$ and two internal interfaces $a$ and $b$. The compound resource $((1 \mid C) \uplus C_W) \triangleright (R_1 \parallel R_2)$ results from attaching $a$ and $b$ to $z$ and $x$ respectively, where $C_W$ is defined in terms of special converters called Wirings [15] that transpose $x + (y + z)$ to $y + (z + x)$. To avoid boilerplate notation and operators, we often describe converter attachments using interface names similar to Section II.C. Moreover, we will drop the distinction between the terms "converter" and "protocol" from now on since multiple converter and resource attachments are encoded as a single attachment written in terms of composition operators.

2) Compound Fused Resources and FRTs: The aforementioned operators do not preserve the structure of Fused Resources. For instance, given the Fused Resource $F : \text{F}(\alpha, \chi, \beta, \delta)$ and the converter $C : \text{C}(\chi + \alpha, \delta + \beta, \alpha + \chi, \beta + \delta)$ that swaps interface positions, then $C \triangleright F$ may be inexpressible in terms of FUSE due to its inverted information flow. We now show typical cases that do preserve the structure of Fused Resources, and thus of FRTs. As we explain in Section IV-A, these cases suffice for the step in the security proof that allows us to abstract over the communication behavior and focus on the common properties. In the following four lemmas, consider the Core records $cr_i : \text{CR}$, the Rest records $rs_i : \text{RS}$, and the Fused Resources $F_i \equiv \text{FUSE}(cr_i, rs_i)$ for $i \in \{1, 2\}$. The first lemma explains the simultaneous access to Fused Resources.

**Lemma 1.** For $F_1 : \text{F}(\alpha, \chi, \beta, \delta)$ and $F_2 : \text{F}(\mu, \nu, \beta, \delta)$, consider the Fused Resource $F : \text{F}(\alpha + \beta, \chi + \beta, \beta + \delta, \delta + \delta)$ defined as $F \equiv \text{FUSE}(cr_1 \parallel cr_2, rs_1 \parallel rs_2)$, where $\parallel$ and $\parallel$ stand for the parallel composition of Core and Rest records respectively, analogous to $\uplus_0$ for oracles. Then $F = C_{F-PL} \triangleright (F_1 \parallel F_2)$, where $C_{F-PL}$ is the wiring that groups the Core and Rest interfaces of $F_1$ and $F_2$ together.

As a corollary of the following lemma, converters may attach to just the Core (or similarly the Rest) interfaces of a Fused Resource since either of the parallel converters could be replaced with the identity converter $1$.

**Lemma 2.** For $F_1 : \text{F}(\alpha, \chi, \beta, \delta)$ and $F_2 : \text{F}(\mu, \nu, \beta, \delta)$, consider the converters $C_1 : \text{C}(\pi, \theta, \alpha, \beta)$ and $C_2 : \text{C}(\mu, \nu, \chi, \delta)$. Then $F_2 = (C_1 \mid C_2) \triangleright F_1$ if $cr_2 = C_{\triangleright} cr_1$ and $rs_2 = C_{\triangleright} rs_1$, where $\triangleright$ attaches a converter to the cfunc oracle of a Core and $\triangleright$, does so for the rfunc oracle of a Rest part.

The next two lemmas determine if a converter’s simultaneous attachment to Core and Rest interfaces results in a new Fused Resource. Using them, one may rewrite an arbitrary converter $C$’s attachment to $F_2$, i.e., $C \triangleright F_2$, into $(C \otimes \_ \triangleright) F_1$, where _ is filled with some wiring, and then (if possible) apply the previous lemma. These lemmas stem from the one-way information flow from Rest to Core interfaces of Fused Resources.

**Lemma 3.** For $F_1 : \text{F}(\pi + \alpha, \chi, \theta + \beta, \delta)$, let $I$ refer to the left-most (w.l.o.g.) Core interface. Assume that $I$’s $\theta$ responses do not depend on $\alpha$ queries, and let $cr_2$ and $rs_2$ be the records that result from removing $I$’s functionality from $cr_1$ and adding it to $rs_1$. Then $(F_2 : \text{F}(\alpha, \pi + \chi, \beta, \theta + \delta)) = C_{\triangleright} cr_1$, where $C_{\triangleright}$ is the wiring converter that reorders the interface positions accordingly.

**Lemma 4.** For $F_1 : \text{F}(\alpha, \pi + \chi, \beta, \theta + \delta)$, let $I$ refer to the left-most (w.l.o.g.) Rest interface. Assume that $\delta$ responses do not depend on $\_ I$’s $\pi$ queries, and let $rs_2$ and $cr_2$ be the records that result from removing $I$’s functionality from $rs_1$ and adding it to $cr_1$. Then $(F_2 : \text{F}(\pi + \alpha, \chi, \theta + \beta, \_)) = C_{\triangleright} rs_1$, where $C_{\triangleright}$ is the wiring converter that reorders the interface positions accordingly.

While these lemmas refer to individual Fused resources, they naturally lift pointwise to FRTs. So these cases also preserve the FRT structure. We write $C_{\triangleright}^{\{\text{cr}_i\}_i}$ for the FRT that consists of the Fused resources $C \triangleright \{cr_i\}_i$ for $rs_i \in \_$. This notation is analogous to Section II.C where $\pi R$ represented the attachment of the protocol $\pi$ to the specification $R$.  

---

4The overloaded $+$ operator on interface names is only used to indicate that disjoint unions are not associative and Wirings enable interface reordering.
We only define the information-theoretic security here; the accordingly, a formal proof of simulation-based security must be done as a sequence of game transformations using the existing tool set from CryptHOL \cite{lochbihler2015formalization,lochbihler2014cryptohol} similar to Lochbihler et al.’s formalization of CC \cite{lochbihler2015formalization}. We now explain the common pattern for FRTs. We also augment this tool set with a new up-to proof rule for trace equivalence of resources.

Lochbihler et al. \cite{lochbihler2015formalization} have already formalized the distinguishability bound \( \mathfrak{d}(R, S) \) on two concrete resources \( R \) and \( S \) using explicit distinguishers. Formally, a distinguisher \( D : \mathbb{G}(B, \alpha, \beta) \) is a GPV that returns a Boolean \( B \) after having interacted with a resource through queries \( \alpha \) and responses \( \beta \). We write \( D \triangleright R : \mathbb{D}(B) \) for the resulting probability distribution over Booleans \( B \). \( D \)'s advantage \( \text{adv}(D, R, S) \) of distinguishing \( R \) and \( S \) is then given by

\[
\text{adv}(D, R, S) = |(D \triangleright R) \cdot \text{True} - (D \triangleright S) \cdot \text{True}|,
\]

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

The proof of \( (3) \) must work for all \( rs \in \mathbb{R}_R \). It should therefore concentrate on the Cores \( \text{real} \) and \( \text{ideal} \) of the Fused Resources \( \{\text{real}\}(rs) \) and \( \{\text{ideal}\}(f(rs)) \). The first step is therefore to pretend that the fusing is part of the distinguisher. For comparison, this step corresponds to applying the dummy adversary lemma in UC.

On the real side \( D \triangleright \{\text{real}\}(rs) \), Lemma 2 shows that the converter \( \pi \) can be attached directly to \( \text{real} \) rather than the Fused Resource, because \( \pi \) attaches only to the Core interface. This transformed core \( \text{real}' \) yields a transformed FRT \( \{\text{real}'\} \mathbb{R}_R \).

On the ideal side, however, we cannot directly use Lemma 2 to push the simulator \( \sigma \) through fuse because \( \sigma \) attaches to the adversary Core and Rest interfaces. This way, the simulator can create a flow of information from the core part to the rest part, which cannot happen in a Fused Resource. We now exploit that the simulator must work for all \( rs \) in the same way, in particular for the Rest record that does not provide any interfaces at all. Accordingly, \( \sigma \) uses only to the interfaces that \( f \) adds on top of \( rs \)'s. By a similar argument, \( f(rs) \) answers queries on these additional interfaces without consulting \( rs \) itself. Moreover, \( f \) may translate poke events from \( rs \) to poke events for the ideal Core and this translation is independent of \( rs \)'s state\footnote{This argument can be made precise in the computational setting: The skolemization \( f \) in \( (3) \) of the existential \( \exists S \in S \) in \( (2) \) ensures that \( f \) chooses the Rest records in \( \{\text{ideal}\} \) uniformly. Formally, this translates to \( f \) being relationally parametric in the queries and responses of \( rs \). So we can decompose \( f(rs) \) into two parts with disjoint state: (i) \( rs \) and (ii) additional independent interfaces and a translator for poke events coming from \( rs \). In the information-theoretic setting, this uniformity argument does not apply and the argument therefore need not hold in pathological cases. We have not yet encountered such a case in practice though.}. So we can move these additional interfaces to \( \text{ideal} \) by Lemma 4. Then, \( \sigma \) only attaches to Core interfaces and can thus be integrated into the Core by Lemma 2. This new core \( \text{ideal}' \) yields a new FRT \( \{\text{ideal}'\} \mathbb{R}_R \).

In summary, it suffices to prove

\[
\text{adv}(D, \{\text{real}\}(rs), \{\text{ideal}\}(rs)) \leq \epsilon
\]
for all $D$. Now, the Fused Resources $\{\text{real}\}'(rs)$ and $\{\text{ideal}\}'(rs)$ use the same Rest record $rs$. Since the Core and Rest in a Fused Resource operate on separate states, we can understand a Core record as a resource of its own, with the poke events as an additional interface that returns only dummy responses $\otimes$. Accordingly, we have

$$\text{adv}(D, \{\text{real}\}'(rs), \{\text{ideal}\}'(rs)) = \text{adv}(D', \text{real}', \text{ideal}')$$

for some $D'$. This key proof step abstracts over the party activation patterns: from now on, the security proof can focus on the cores real and ideal. Since these are like resources, this proof can be approached as described in the literature [13].

Frequently, such a proof must transform the resource into a particular shape, e.g., to prepare for the application of a reduction argument. Lochbihler et al. [15] found that these steps can often be justified by trace equivalence. They gave the following bisimulation-style proof rule for establishing trace equivalence of resources, where $\text{dirac}(x)$ is the one-point distribution on $x$, and $\overline{\text{run}}(p,a)$ denotes the weighted combination of running a resource from the distribution $p$ of resources with query $a$, and $p|_{x}$ conditions a distribution $p$ over pairs $\alpha \times \beta$ on the elementary event $x : \alpha$.

**Theorem 1** ([15 Thm. 1]). Two resources $R$ and $S$ are trace equivalent iff there exists a relation $X$ between distributions of resources such that

1) $X$ relates $\text{dirac}(R)$ to $\text{dirac}(S)$.

2) Whenever $(p,q) \in X$, then for all queries $a$, $\overline{\text{run}}(p,a)$ and $\overline{\text{run}}(q,a)$ have the same marginal distribution on responses and $(\overline{\text{run}}(p,a)|_{b}, \overline{\text{run}}(q,a)|_{b}) \in X$ for each possible response $b$ to the query $a$.

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

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

To counter this case explosion, we introduce a closure operator $[X]$ on $X$ and generalize the above theorem to an up-to proof rule. The closure $[X]$ of a relation $X$ of distributions is defined inductively as follows, where $p \gg f$ denotes the $p$-weighted combination of a family $f$ of distributions, i.e.,

$$(p \gg f) \cdot x = \sum_{y} p \cdot y \cdot f(y) \cdot x.$$

1) Whenever $(p,q) \in X$, then $(p,q) \in [X]$.

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

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

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

1) $X$ relates $\text{dirac}(R)$ to $\text{dirac}(S)$.

2) Whenever $(p,q) \in X$, then for all queries $a$, $\overline{\text{run}}[p,a]$ and $\overline{\text{run}}[q,a]$ have the same marginal distribution on responses and $(\overline{\text{run}}[p,a]|_{b}, \overline{\text{run}}[q,a]|_{b}) \in [X]$ for each possible response $b$ to the query $a$.

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

**B. Composability**

We now show that our security definition [1] yields composable security statements. In CC, a protocol $\pi$ typically uses multiple resources $R_{1}, \ldots, R_{n}$ to create a resource $\pi \triangleright (R_{1} \parallel \ldots \parallel R_{n})$, which should be hard to distinguish from a simulated ideal resource $\sigma \triangleright S$. Similarly, another protocol $\rho$ may use $S$ and possibly other resources to achieve another ideal resource $T$. To abstract over the party activation patterns of these resources, we consider FRTs rather than individual resources. This ensures that the security proof for $\rho$ is independent of the patterns that we need for $R_{i}$ during the composition.

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

**Definition 2** (Concrete security for individual resources). A resource $R$ $\epsilon$-securely realizes a resource $S$ if there is a converter $\sigma$ that attaches to $S$’s adversary interfaces such that

$$\text{adv}(D, R, \sigma \triangleright S) \leq \epsilon$$
Theorem 3 (Composability for individual resources).

1) Every resource $R$ 0-securely realizes itself.

2) If $R_1$ $\epsilon_1$-securely realizes $S_1$ as witnessed by $\sigma_1$ and $R_2$ $\epsilon_2$-securely realizes $S_2$ as witnessed by $\sigma_2$, then $R_1 \parallel R_2$ $(\epsilon_1 + \epsilon_2)$-securely realizes $S_1 \parallel S_2$ as witnessed by $\sigma_1 \sigma_2$.

3) If $R$ $\epsilon$-securely realizes $S$ as witnessed by $\sigma$ and $S'$ $\epsilon'$-securely realizes $T$ as witnessed by $\tau$, then $R \parallel (\epsilon + \epsilon')$-securely realizes $T$ as witnessed by $\tau \circ \sigma$.

4) If $R$ $\epsilon$-securely realizes $S$ as witnessed by $\sigma$ and $C$ is a converter that attaches to $R$'s user interface, then $C \parallel R$ $\epsilon$-securely realizes $C \parallel S$ as witnessed by $\sigma$.

These composability results suffice to show that secure realization between FRTs is also compositional. As a FRT $\{cr\} \bullet$ is a family of Fused Resources $\{cr\}(rs)$ for $rs \in \bullet$, we obtain composability by lifting Thm. 3 pointwise to specifications. Note that the simulator and the rest embeddings remain independent of the chosen Fused Resources, as required by Def. 1.

Theorem 4 (Composability for FRTs).

1) The identity protocol $\mathbb{1}$ applied to a FRT $\{cr\} \bullet$ 0-securely realizes the FRT $\{cr\} \bullet$, for $\bullet \supseteq \bullet$ with the simulator being the identity converter and the rest embedding being the identity function.

2) Let $\pi_1\{real\}_{\bullet_1}$ $\epsilon_1$-securely realizes the FRT $\{ideal\}_{\bullet_1}$ with simulator $\sigma_1$ and rest embedding $f_1$, for $i = 1, 2$. The parallel composition

\[
(\pi_1 \mid \pi_2)\{real\}_{\bullet_1 \times \bullet_1} \{\epsilon, \epsilon real\\}_{\bullet_1 \times \bullet_1}
\]

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

\[
\{ideal\}_{\bullet_1 \times \bullet_2}
\]

with simulator $\sigma_1 \sigma_2$ and rest embedding $f(\pi_1 \pi_2) = (f_1(\pi_1), f_2(\pi_2))$.

3) Let $\pi_1\{real\}_{\bullet_1}$ $\epsilon$-securely realizes the FRT $\{middle\}_{\bullet_1}$ with simulator $\sigma$ and rest embedding $f$ and let $\rho_1\{middle\}_{\bullet_1}$ $\epsilon'$-securely realize $\{ideal\}_{\bullet_1}$ with simulator $\tau$ and rest embedding $g$ such that $\bullet_1 \subseteq \bullet_1$. Then, $\pi_1\{real\}_{\bullet_1}$ $(\epsilon + \epsilon')$-securely realizes $\{ideal\}_{\bullet_1}$ with simulator $\tau \circ \sigma$ and rest embedding $g \circ f$.

4) If $\pi_1\{real\}_{\bullet_1}$ $\epsilon$-securely realizes the FRT $\{ideal\}_{\bullet_1}$ and $C$ is a converter that attaches to the Core user interface, then $C \pi_1\{real\}_{\bullet_1}$ $\epsilon$-securely realize $\{ideal\}_{\bullet_1}$, with the same simulator and rest embedding.

V. CASE STUDIES

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

We apply the following conventions in diagrams. Fused Resources are depicted on the right; we use subscripted $R$ as short names for them, e.g. $R_{aut1}$ corresponds to the FRT $\{aut1\}$ that is instantiated with the Rest record $rs_1$ in Figure 1a. Converters’ attachment to Fused Resources is represented using arrows, where solid and dashed arrows represent the attachment to Core and Rest interfaces respectively. Such an attachment results in new Fused Resources that are presented using rounded rectangles with different colors or borders. Fused Resources or converters that are in the same column are composed in parallel using $\parallel$ or $|$ respectively. The left-most interfaces of every diagram belong to Eve, Alice, Bob, and “Rest” from top to bottom. For simplicity, we merge multiple Rest or adversary interfaces into a single arrow and mark their meeting point using a black circle. The interfaces are preserved among all the diagrams, so we name them only in some of the subfigures.

Our example construction combines two cryptographic primitives in Figure 1a. First, the Diffie-Hellman key exchange protocol with the converters $C_{dhA}$ and $C_{dhB}$ is used to construct a key $R_{key}$ from two of the authenticated channels, i.e., $R_{aut1}$ and $R_{aut2}$ that are in the opposite directions. Second, we use one-time-pad in the converters $C_{enc}$ and $C_{dec}$ to construct a secure channel $R_{sec}$ from the key $R_{key}$, which is the result of Diffie-Hellman key exchange, and the last authenticated channel $R_{aut3}$.

The aforementioned components behave as their name suggests. The key and channel resources are defined as we have explained in Section [III-B3]. The $C_{enc}$ and $C_{dec}$ converters are stateless and each provide a single external interface for Alice and Bob respectively. When the $C_{enc}$ converter is queried with a message $m$, it fetches a key $k$ via its internal interface that is attached to $C_{dhA}$ and forwards $m \oplus k$, where $\oplus$ is the xor operator for bit-strings, to its internal interface that is connected to the authenticated channel $R_{aut3}$. The $C_{dec}$ converter works analogously to decrypt the cipher-text that it fetches from $R_{aut3}$.

The $C_{dhA}$ and $C_{dhB}$ are more involved. We describe $C_{dhA}$ as a representative of these converters since they are like duals. $C_{dhA}$ is a stateful converter with two external interfaces. The first external interface, which is placed at the top and redirected to $R_{e1}$’s rest part, stores a half-key $x$ in its state upon receiving a unit query $\oplus$ and puts $g^n$ into the authenticated channel $R_{aut1}$, where $g$ is the cyclic group’s generator. The second external interface, which is attached to $C_{dec}$, can only get queried after the query to the first interface; otherwise, the failure mechanism is triggered. It fetches the half-key $g^n$, which is sent by the other party, from the authenticated channel $R_{aut2}$ and stores $(g^n)^x$ in its state.

Figure 1 depicts the central steps of the security proof. First, corresponding to steps from Figure 4b to 4c we prove that the specification $R_{e1}$ securely realizes the key specification $R_{key}$. Second, corresponding to the dotted area in Figures 4c and 4d.
we make our key-exchange protocol lazy. That is, we prove that with three major steps. First, going from Figure 4a to Figure 4b, we show that the specification \( C \) is trace equivalent to a fused resource \( R \) that samples the pair \((x, y)\) when it receives a query on any of \( \text{Act}_{x} \text{Act}_{y} \) and \( \text{Eve}_1 \), or \( \text{Eve}_2 \). This proof step utilizes Lemma 2 to reason about the attachment of Diffie-Hellman converters to the authenticated channels and Lemma 3 to move the protocol initiation queries, i.e., \( \text{Act}_{x} \) and \( \text{Act}_{y} \), to \( R \)’s rest part.

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

Third, corresponding to the step between Figures 4b and 4c we prove an arbitrary distinguisher \( D \)'s advantage in distinguishing \( R_{\text{FRT}} \) and \( R_{\text{FRT}} \) is bounded by \( D \)'s advantage in the decisional Diffie-Hellman game, in short \( \text{ddh} \). This proof step is based on a reduction in which \( R_{\text{FRT}} \) and \( R_{\text{FRT}} \) query an external oracle to receive the triples \((g^x, g^y, g^{x+y})\) and \((g^x, g^y, g^z)\) respectively, which are used to answer all the user and adversary queries.

The secure realization of \( R_{\text{sec}} \) using \( R_{\text{key}} \) requires less effort. This corresponds to the step from Figure 4c to Figure 4d. We prove that \( R_{\text{key}} \) is trace-equivalent to the fused resource resulting from attaching \( C_{\text{sim}2} \) to \( R_{\text{sec}} \). The simulator \( C_{\text{sim}2} \) does not need to communicate with \( R_{\text{sec}} \)’s rest part. Upon receiving a look query on \( \text{Eve}_3 \), it queries \( R_{\text{sec}} \) with a look query to get the length of \( R_{\text{sec}} \)’s content and outputs a uniformly sampled bit string with the same length; the output is stored in \( C_{\text{sim}2} \)’s state to answer future look queries. The rest of the adversary queries to \( C_{\text{sim}2} \), e.g. forward or drop queries, are simply forwarded to \( R_{\text{sec}} \). The rest embedding \( E_{\text{opt}} \) keeps track of the events that are received on \( R_{\text{key}} \) and \( R_{\text{aut}} \)’s rest interfaces to trigger the corresponding events on \( R_{\text{sec}} \)’s core.

As mentioned earlier, the composition theorem combines this construction step with the previous ones.

In summary, we obtain the following security results. Let \( \epsilon \) denote the distinguishing advantage for the decisional Diffie-Hellman game and consider the protocol \( \pi \equiv \Pi \left( \left( \left| C_{\text{dec}} \circ C_{\text{dh}} \right| \right| \left| C_{\text{dh}} \circ C_{\text{dec}} \right| \right) \). Then, \( \pi \left( \left| \text{aut1} \right| \left| \text{aut2} \right| \left| \text{aut3} \right\rangle \right) \) -securely realizes the specification \( \Pi_{\text{sec}} \) with the simulator \( C_{\text{sim}1} \circ C_{\text{sim}2} \)
and the rest embedding $E_{opt}(E_{dh}(rs_1||rs_2)||rs_3)$.

VI. DISCUSSION AND RELATED WORK

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

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

Our work is the first that formulates requirements for system communication modeling and provides a rigorous solution as an instance of CC. In Section III-A, we explained the importance of such a solution for the reusability of security proofs. The pen-and-paper presentations of existing CC results [12], [16], [17], [18] do not necessitate providing details on the communication model and, by eliding them, they must be understood as just a high-level overview of security proofs and their composition. However, in our approach, security proofs are mathematical objects and leaving out any details would invalidate them.

Comparing to the simulation-based frameworks [2], [7], [11], [13], we consider a less intricate type of system communication. Nevertheless, there are scenarios in which our approach excels. In simulation-based frameworks, the adversary Turing machine controls the communication between all protocol components. The composition theorems in these frameworks fix the corruption model but consider arbitrary adversary machines besides that. This resembles the role that the FRTs’ Rest part plays in our definition of secure constructions and their composition. We provide composition theorems that consider arbitrary Rests; however, we only consider a one-way information flow from the Rest to the Core part, which corresponds to the non-adversary components in the simulation-based frameworks. But note that we allow multiple Rests in our model, e.g. when two FRTs are composed in parallel. This is convenient for modeling scenarios where parties are not corrupted by the same adversary, which would be challenging in simulation-based framework since they use a central adversary machine.

We build on an existing formalization of CC in CryptHOL [15], to which we refer as CCHOL in this section. We reuse large parts of CCHOL’s formalization of resources, converters, distinguishers, and various composition operators that were briefly recapped in the paper. Overall, our formalization required 9.7K lines of lemmas and definitions, where the case study constitutes 3.5k lines and the rest form a library of reusable resource definitions and proof rules that can be used for modelling and reasoning about other security protocols.

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

There are few results on formalizing simulation-based proofs. Most of them [5], [10] focus on individual (or class of) protocols and hence they do not offer composition theorems that can be applied in arbitrary context. EasyUC [8], which is the most similar to our work, does support the composition of security proofs. It uses EasyCrypt [3] to formalize a simplified version of the Universal Composability framework with a restricted adversary machine. Such simplifications are essential since the formalization of Universal Composability in its most general form is challenging, even with a state-of-the-art formal-methods tool like EasyCrypt.

We compare EasyUC with our result in two respects. First, EasyUC’s restriction on the adversary machine affects the reusability of security proofs, as explained in Section III-A since it leads to a fixed communication pattern among the non-adversary machines. Second, formalizing security statements in EasyUC is more difficult. This is a benefit that stems from CC’s abstract approach to cryptography; we don’t need to delve into concrete details that are intrinsic part of every security statement in UC-style frameworks. For instance, the EasyUC formalization that constructs a secure channel using Diffie-Hellman key exchange and a one-time-pad constitutes 18K lines of proofs and definitions, where 12K lines are devoted to composing the concrete security statement alone.

VII. CONCLUSION AND FUTURE WORK

We have presented an abstract approach to communication modeling in Constructive Cryptography that is suitable for mechanized verification. We highlight the importance of systems communication patterns on the reusability of security proofs and offer a rigorous approach that allows protocol designers to abstractly capture it. We explain the limitations of existing formalized composability results that do not model systems communications to a full extent. By lifting such limitations, we support the modular formalization of a wider range of scenarios than existing formal-methods tools support. Carrying out further case studies and enhancing automation support is left as future work.
The root directory consists of many theory files, the Specifications folder that contains the ideal specifications for keys and channels, and the Constructions folder that stores our case study’s formalization. The theory files in the root directory contain the lemmas and definitions that correspond to their names:

- **Fold_Spmf.thy** and **Goodies.thy** formalize the probabilistic fold function and a series of helper lemmas that are used in the other theory files.
- **More_CC.thy** and **State_Isomorphism.thy** formalize our extensions to the theory of resources and converters. The lemmas and definitions in this theory file are not specific to Fused Resources only.
- **Observe_Failure.thy** formalizes the indistinguishability of resources when a bottom element \( \bot \) is added to every distribution’s sample space to model failures.
- **Fused_Resource.thy** formalizes Fused Resources, their trace equivalence, and various operators on Core and Rest records. In particular, the proposition \( \text{trace’}_\text{eq}_\text{=} \text{simI_upto} \) proves the Theorem 2 for Fused Resources.
- **Construction_Utility.thy** formalizes common building blocks for defining compound Fused Resources. In particular, the propositions \( \text{parallel_oracle}_\text{fuse} \) and \( \text{attach_parallel}_\text{fuse} \) prove the Lemmas 1 and 2 and the propositions \( \text{fuse}_\text{ishift_core}_\text{to} \) and \( \text{move}_\text{simulator}_\text{interface} \) prove the Lemmas 3 and 4 respectively.
- **Concrete_Security.thy** formalizes the notion of information-theoretic concrete security. In particular, the propositions \( \text{constructive_security}_\text{obsf_trivial}, \text{parallel_constructive_security}_\text{obsf}, \text{constructive_security}_\text{obsf_composability}, \) and \( \text{constructive_security}_\text{obsf_lifting}_\text{usr} \) prove the four claims of Theorem 3.
- **Asymptotic_Security.thy** extends the above to asymptotic notion of security.

The Specifications folder provides the ideal specifications for keys and channels. We explained the details of **Key.thy** in Section III-B.3

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

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

### References


