A Core Calculus for Equational Proofs of Cryptographic Protocols

JOSHUA GANCHER, Carnegie Mellon University, USA
KRISTINA SOJAKOVA, INRIA, France
XIONG FAN, Rutgers University, USA
ELAINE SHI, Carnegie Mellon University, USA
GREG MORRISETT, Cornell University, USA

Many proofs of interactive cryptographic protocols (e.g., as in Universal Composability) operate by proving the protocol at hand to be *observationally equivalent* to an idealized specification. While pervasive, formal tool support for observational equivalence of cryptographic protocols is still a nascent area of research. Current mechanization efforts tend to either focus on diff-equivalence, which establishes observational equivalence between protocols with identical control structures, or require an explicit witness for the observational equivalence in the form of a bisimulation relation.

Our goal is to simplify proofs for cryptographic protocols by introducing a core calculus, IPDL, for cryptographic observational equivalences. Via IPDL, we aim to address a number of theoretical issues for cryptographic proofs in a simple manner, including probabilistic behaviors, distributed message-passing, and resource-bounded adversaries and simulators. We demonstrate IPDL on a number of case studies, including a distributed coin toss protocol, Oblivious Transfer, and the GMW multi-party computation protocol. All proofs of case studies are mechanized via an embedding of IPDL into the Coq proof assistant.

CCS Concepts: • **Security and privacy** → **Logic and verification**; • **Theory of computation** → *Equational logic and rewriting*.

Additional Key Words and Phrases: cryptographic protocols, equational reasoning, observational equivalence

ACM Reference Format:
Joshua Gancher, Kristina Sojakova, Xiong Fan, Elaine Shi, and Greg Morrisett. 2023. A Core Calculus for Equational Proofs of Cryptographic Protocols. *Proc. ACM Program. Lang.* 7, POPL, Article 30 (January 2023), 43 pages. https://doi.org/10.1145/3571223

1 INTRODUCTION

An important area in the design of secure systems is the use of *computer-aided proofs* for certifying the design of cryptographic protocols [Barbosa et al. 2021a]. As new and complex cryptographic mechanisms become deployed, it becomes increasingly important to mechanize security proofs in order to rule out unforeseen attacks not captured in on-paper proof developments.

While a number of sophisticated protocols have been proven secure using existing tools [Barthe et al. 2011, 2015; Blanchet 2006, 2013; Lochbihler and Sefidgar 2018; Meier et al. 2013; Petcher and Morrisett 2015], work to mechanize proofs for *distributed message-passing* protocols in the style of Universal Composability (UC) [Canetti 2000] is only in its initial stages [Barbosa et al. 2021b; Canetti et al. 2019; Lochbihler et al. 2019]. Since UC provides an extremely expressive and general

Authors' addresses: Joshua Gancher, Carnegie Mellon University, USA; Kristina Sojakova, INRIA, France; Xiong Fan, Rutgers University, USA; Elaine Shi, Carnegie Mellon University, USA; Greg Morrisett, Cornell University, USA.

2475-1421/2023/1-ART30
https://doi.org/10.1145/3571223

framework for defining the security of protocols in a modular way, it has the potential to serve as a common framework for verified security proofs across cryptographic domains.

Challenge for Verification: Observational Equivalence. In UC and related frameworks [Maurer 2012], cryptographic protocols are judged secure when they are judged *observationally equivalent* to an idealization which guarantees security using a trusted third party. Observational equivalence of protocols is ubiquitous in cryptography, as it provides a uniform framework for a broad spectrum of security properties, easily capturing privacy, integrity, and availability.

However, message-passing protocols pose semantic challenges for proving observational equivalence, due to the presence of distributed computations and interactivity. Distributed protocols raise issues of *nondeterminism* if two parties wish to concurrently send messages, while interactivity requires observational equivalence to be established using *bisimulations* on the protocol states, drastically raising the proof effort.

To date, these added complexities have not yet been fully addressed by verification methods. Prior verification efforts are either libraries [Canetti et al. 2019; Lochbihler et al. 2019] based on sequential program logics [Barthe et al. 2011; Lochbihler and Sefidgar 2018; Petcher and Morrisett 2015], which require explicit bisimulation witnesses, or are based on symbolic model checking [Blanchet 2013; Meier et al. 2013], or specialized security-preserving program transformations [Blanchet 2006], lacking enough expressivity to encode observational equivalences for general classes of message-passing systems.

Equational Reasoning for Protocols. In this paper, we address this gap in the literature by introducing a core language, IPDL (standing for Interactive Probabilistic Dependency Logic), for mechanizing observational equivalences between message-passing protocols. By designing an equational proof system for equivalences of interactive protocols, we deliver new, simplified proofs of protocol security in a style similar to UC *without* requiring hand-written bisimulation relations.

The core idea of IPDL is that while distributed message-passing can in general introduce a number of complexities due to scheduling, these issues do not typically arise in cryptographic protocols. Accordingly, we restrict our attention to the well-behaved (but still expressive) subset of *confluent* protocols, which are guaranteed to not introduce races due to scheduling. By restricting our attention to confluent protocols, we obtain equational proof principles which would not be sound in the more general setting.

We mechanize the equational logic of IPDL and demonstrate it on a number of case studies, including secure communication protocols employing encryptions and Diffie-Hellman key exchange, protocols for Oblivious Transfer [Goldreich et al. 1987], the GMW protocol for secure two-party computation [Goldreich et al. 1987], and a multi-party protocol for secure randomness generation [Blum 1983]. All proofs are written in a purely equational style, without requiring explicit bisimulation relations. Our proof developments are open-source.[1]

While we present IPDL through a stand-alone formalization and mechanization, we do not intend for IPDL to capture all desirable proof strategies in cryptography. Indeed, our confluent semantics and equational proof techniques are likely to excel "on top" of a lower-level probabilistic program logic, such as EasyCrypt [Barthe et al. 2011]. Indeed, EasyCrypt could be used to validate lower-level probabilistic reasoning steps currently out of scope for IPDL, while IPDL could handle all high-level equational reasoning for message passing.

[1] https://github.com/ipdl/ipdl

1.1 Contributions

- We introduce IPDL, a core language for distributed, interactive message-passing in cryptographic protocols. IPDL is packaged with an *equational logic* for protocols, enabling simple, high-level proofs without explicit bisimulations.
- We prove the equational logic of IPDL sound in the *computational model*: informally, whenever the logic proves that two families of protocols (indexed by the security parameter) are approximately equivalent, then *no* probabilistic polynomial-time distinguisher can distinguish them with greater than negligible error.
- We mechanize the core logic IPDL in Coq, and demonstrate it on a number of case studies, including basic communication and authentication protocols, a multi-party protocol for secure randomness generation [Blum 1983], and the GMW protocol for two-party computation [Goldreich et al. 1987].

2 RELATED WORK

EasyCrypt [Barthe et al. 2011], CryptHOL [Lochbihler and Sefidgar 2018], and FCF [Petcher and Morrisett 2015] are all *probabilistic program logics* for *sequential* programs. While very expressive for probabilistic reasoning, these tools by design provide no built-in support for interactive protocols with distributed message-passing behaviors. While a number of interactive protocols have been proven secure in these tools [Butler et al. 2020; Defrawy and Pereira 2019], these proof efforts employ ad-hoc techniques for reasoning about message passing.

To make message passing less ad-hoc, EasyUC [Canetti et al. 2019] for EasyCrypt, and the Constructive Cryptography effort for CryptHOL [Lochbihler et al. 2019] both encode general forms of interactive message passing into their ambient program logics. However, neither tool provides sophisticated proof techniques for conducting equivalence proofs, requiring the user to hand-write tedious bisimulation relations, which does not scale for larger protocols. Additionally, [Barbosa et al. 2021b] work to encode UC proofs in a modular fashion in EasyCrypt, but still rely on bisimulations for basic proof steps. The purpose of IPDL is, in part, to eliminate such hand-written bisimulations.

CryptoVerif [Blanchet 2006] is a tool for equivalence-based computational reasoning for security protocols in which parties communicate over fully untrusted networks asynchronously. While excellent at semi-automated proofs for privacy and authentication properties, CryptoVerif cannot express observational equivalences between dissimilar protocols, nor reason compositionally in the sense of embedding security proofs for subprotocols into larger proof developments. In contrast, IPDL directly encodes observational equivalences in a modular way.

Squirrel [Baelde et al. 2021] and its associated BC logic [Bana and Comon-Lundh 2014] proves similar properties to CryptoVerif and related symbolic tools [Blanchet 2013; Meier et al. 2013] through a first-order logic sound against polynomial time adversaries. While Squirrel does allow for diff-equivalence, which establishes observational equivalence between protocols with identical control structures, it cannot establish modular observational equivalences between dissimilar protocols. Indeed, both Squirrel and CryptoVerif assume that the prover knows the entire protocol all at once, which is incompatible with modular proofs.

Both Squirrel and CryptoVerif assume that protocol participants only communicate through the adversary, who controls the untrusted network. Through arbitrary manipulation of channels, IPDL can express many more kinds of dataflow in protocols, such as ideal communication channels between parties and functionalities. Indeed, our main case studies (such as the GMW protocol [Goldreich et al. 1987]) cannot even be expressed in Squirrel or Cryptoverif, due to a lack of generality in communication topology. We believe that IPDL sits in a sweet spot of modularity between expressive tools requiring explicit bisimulations (EasyUC and the CryptHOL-based framework) and tools for easy whole-protocol analyses (CryptoVerif and Squirrel).

Tamarin [Meier et al. 2013], Proverif [Blanchet 2013], and others [Bhargavan et al. 2021; Cremers 2008] are *symbolic* [Dolev and Yao 1983] protocol analysis tools, which abstract cryptographic mechanisms into term algebras. As described by Squirrel [Baelde et al. 2021], symbolic tools enumerate what actions attackers *may* do, while computational tools (including IPDL) state what the attacker *cannot* do. Thus, the computational model subsumes the symbolic one, and does not carry a risk that the attacker is not modeled with enough computational power. While a significant line of work has proven that the symbolic model is sufficient to guarantee computational soundness under certain conditions [Abadi and Rogaway 2002; Backes et al. 2012; Cortier and Warinschi 2011], such arguments require intricate completeness arguments not required by IPDL.

IPDL rests upon a long lineage of using observational equivalences to model cryptographic protocol security, both in the symbolic setting [Abadi and Rogaway 2002; Blanchet 2013; Lowe 1996; Meier et al. 2013; Schneider 1996] and in the computational one [Backes et al. 2007; Baelde et al. 2021; Canetti 2000]. The main novelty of IPDL is enabling computationally sound formal proofs of observational equivalence between cryptographic protocols without any explicit use of bisimulation relations.

ILC [Liao et al. 2019] uses programming language techniques such as affine typing to capture the semantics of Universal Composability [Canetti 2000] faithfully. Through two restrictions – processes may only send a single message after receiving a single message, and no two processes may listen on the same channel – ILC guarantees *confluence*, as is claimed by the native semantics of Universal Composability. ILC's main contribution is its core language, and does not deliver any proof methods for establishing observational equivalences. In contrast, IPDL makes a different set of restrictions to guarantee confluence (blocking reads, rather than single messages), and is attached to an equational proof system for protocol equivalence. Additionally, via its "choice" construct, protocols in ILC may make use of nontrivial timing information, such as deciding what to do next based on which channel receives input. IPDL explicitly rules out dependence on timing information in order to achieve simple equational rules.

Pirouette [Hirsch and Garg 2022] is a language for higher-order *choreographies*, which give a similarly concise syntax for specifying distributed protocols. Additionally similar to IPDL, Pirouette contains an equational proof system for reasoning about protocol behaviors. While the focus for Pirouette is higher-order programming of distributed protocols with *endpoint projections* to individual components, the focus for IPDL is using the proof system to conduct computationally sound reasoning for cryptographic protocols.

3 OVERVIEW OF IPDL

Before we turn to the formal details of IPDL, we outline the main ideas behind expressing security of protocols and proofs in IPDL.

3.1 Background on Simulation-Based Security

To motivate our setting of distributed cryptographic protocols, we give some details about UC-style security modeling independent of any formal framework. Simulation-based security in the style of UC [Canetti 2000] and Constructive Cryptography [Maurer 2012] provides an expressive and general way to model security for distributed cryptographic protocols, such as secure multi-party computation (MPC) [Lindell 2020]. The core idea is that cryptographic protocols π are modeled as *open*, *message-passing* systems of *parties* and *functionalities*, *i.e.*, services assumed by the protocol to be secure, such as an authenticated communication channel.

Interfaces. Virtually all protocols have two disjoint interfaces with the external world: an *environmental* interface, and an *attacker* interface (also called the *backdoor* in UC [Canetti 2000]). The environmental interface is used to model the high-level I/O of the protocol, and is used by the parties; *e.g.*, the inputs and outputs of a particular circuit for MPC, or the input votes and output decision of a secure voting protocol. In contrast, the attacker interface specifies how an attacker may subvert specific implementation details of the protocol, such as interacting with corrupted parties, or eavesdropping on communication channels.

Protocol Security. We define security for protocols π by comparing them to idealizations Ideal in which all computations are replaced by a trusted functionality that provides security by fiat. The external interfaces of π and Ideal are identical, but the attacker interfaces are not. Typically, the attacker may corrupt parties and eavesdrop on intermediate communications in π, while in Ideal the attacker is severely limited, such as only deciding whether or not the computation may complete. To compare the two protocols, we ask for a *simulator* Sim which *converts* the attacker's interface of Ideal to that of π. The simulator's role is to demonstrate that attacks in π are no more powerful than attacks in Ideal; indeed, this is the case if no attacker can tell the difference between interacting with π and Sim + Ideal, where + connects subcomponents by interface composition. We formalize this idea by stating that π *realizes* Ideal if $\pi \approx_{\mathrm{obs}} \mathsf{Sim} + \mathsf{Ideal}$, where $\approx_{\mathrm{obs}}$ expresses *observational equivalence*. Following the Dummy Adversary Theorem in UC [Canetti 2000], it is fully general to allow the environment (supplying high-level inputs and outputs to π and Ideal) to coincide with the attacker (attacking either π or Sim + Ideal). In the cryptographic domain, observational equivalence is expressed through *resource-bounded*, *probabilistic* machines that output a decision Boolean.

Proof Strategies. Proofs of security for complex protocols are rarely conducted in one single step. Instead, cryptographers use *hybrids*, or intermediate protocol equivalences, which allow the proof to be written modularly. Prototypically, proofs of security appear as chains of *exact* equivalences and *approximate congruence* steps:

$$\pi = R_1 + H_1 \approx R_1 + H_1' = \cdots = R_k + H_k \approx R_k + H_k' = \mathsf{Ideal}, \tag{1}$$

where each R_i is an intermediate *reduction*, and each pair (H_i, H_i') is an *indistinguishability assumption* of the form $H_i \approx H_i'$. In this format, each exact equivalence = is semantic equivalence of the two protocols, while each approximate equivalence $\approx$ is simply an application of a single indistinguishability assumption, using the fact that $\approx$ is a congruence for +. Crucially, the above proof strategy does not involve any cryptographic reasoning other than proper identification of the reductions R_i and assumptions $H_i \approx H_i'$. All nontrivial proof effort is discharged in proving semantic equivalences =, which in general require *bisimulations*, *i.e.*, relational invariants across the states of the two protocols in question.

3.2 Key Ideas of IPDL

Motivated by UC-style security, the purpose of IPDL is to enable cryptographers to state and prove observational equivalences, such as those in Equation 1, as easily as possible.

Channels and Reactions. As discussed in Section 3.1, UC-style proofs typically require hand-written bisimulations to prove one protocol semantically equivalent to another. While expressive, bisimulations are tedious to write and too low-level for serious proof efforts, thus diverting the proof effort away from the high-level security proof. We eliminate the need for hand-written bisimulations by choosing a language for protocols which is simultaneously expressive and well-behaved enough to admit *equational reasoning* principles.

$$\begin{aligned}
&\text{protocol P}\Big[\{\text{In}_i : \{0,1\}^{L(\lambda)}\}_{i=1}^{q(\lambda)}, \{\text{Out}_i : \{0,1\}^{L(\lambda)}\}_{i=1}^{q(\lambda)}, \{\text{Leak}_i : \{0,1\}^{C(\lambda)}\}_{i=1}^{q(\lambda)}\Big] := \\
&\quad \text{new Key} : \{0,1\}^{\text{len}_K(\lambda)} \text{ in} \\
&\quad \text{new } \{\text{Ctxt}_i : \{0,1\}^{C(\lambda)}\}_{i=1}^{q(\lambda)} \text{ in} \\
&\quad\quad \big(\text{Key} := \text{samp } (\text{unif}_{\text{len}_K(\lambda)})\big) \;||\\
&\quad\quad \Big|\Big|_{i=1}^{q(\lambda)} \big(\text{Ctxt}_i := x \leftarrow \text{read In}_i;\ k \leftarrow \text{read Key};\ \text{samp } (\text{enc}(k,x))\big) \;||\\
&\quad\quad \Big|\Big|_{i=1}^{q(\lambda)} \big(\text{Out}_i := c \leftarrow \text{read Ctxt}_i;\ k \leftarrow \text{read Key};\ \text{ret } (\text{dec}(k,c))\big) \;||\\
&\quad\quad \Big|\Big|_{i=1}^{q(\lambda)} \big(\text{Leak}_i := \text{read Ctxt}_i\big)
\end{aligned}$$

Fig. 1. Simple encryption protocol in IPDL.

At its core, IPDL is a process calculus for describing networks of interacting probabilistic computations, communicating via *write-once channels*. The basic computational unit in IPDL is *channel assignment* ($c := R$), which assigns the *reaction* R to channel c. Reactions are simple monadic programs which may read from other channels, perform probabilistic sampling, and branch with if statements. We enforce through typing that channels in IPDL carry one unique reaction.

Reactions interact through *protocols* P, which, other than channel assignment, are built out of parallel composition $P \;||\; Q$ and *local channel generation*, new $c : \tau$ in P, where τ is a data type. To ensure parity with semantics for computational cryptography, all data types represent bitstrings of a given length.

Protocol Families. Throughout, we make extensive use of *protocol families*, *i.e.*, structured families of protocols $\{P_i\}_{i=1}^N$, indexed by natural numbers. Given a family of protocols $\{P_i\}_i$, we write $\big|\big|_i P_i$ for the protocol $P_1 \;||\; \dots \;||\; P_N$. Similarly, we write new $\{c_i : \tau\}_{i=1}^N$ in P for the protocol given by new $c_1 : \tau$ in $\dots$ new $c_N : \tau$ in P. We do not give explicit syntax for protocol families, instead deferring their construction to the meta-language through the above abbreviations.

IPDL does not have an explicit construct for unbounded recursion. Since cryptographic protocols are typically constrained to run in polynomial time, virtually all candidate uses of recursion can be unfolded into a protocol family bounded by a polynomial in the security parameter.

Similarly, IPDL's assumption that all channels are write-once is not a practical restriction, since any channel o that carries multiple succesive messages can be split into a family $\{o_i\}$ of channels, again bounded by a polynomial in the security parameter.

3.3 Example: Secure Message Communication

We demonstrate IPDL on a simple example using encryption to communicate q secret messages over an authenticated (but not private) communication network. Assuming that a key for symmetric encryption has been distributed ahead of time, the sender may send encrypted messages over the network, which the receiver will be able to decrypt correctly. We assume the attacker may view the in-flight (encrypted) messages.

We model this protocol (simplified for brevity) in Figure 1, where all channel names are uppercase to enhance readability. Throughout, $\lambda \in \mathbb{N}$ is the *security parameter*, used to define computational soundness. The protocol operates as follows: it is parameterized by three collections of *free channels*,

$\{\mathsf{In}_i\}$, $\{\mathsf{Out}_i\}$, and $\{\mathsf{Leak}_i\}$. The channels In_i and Out_i are the inputs and outputs of the sender and receiver respectively, while channels Leak_i carry the in-flight messages observable by the adversary. Through typing, we will obtain that Out_i and Leak_i are outputs of the protocol, while In_i are inputs. All channels are typed with a length of values they carry. First, the protocol generates local channels: Key for key and the family $\{\mathsf{Ctxt}_i\}$ for ciphertexts. We choose the key uniformly by assigning Key the reaction that samples from $\mathsf{unif}_{\mathsf{len}_K(\lambda)}$, where $\mathsf{len}_K(\lambda)$ is the length of the key. To generate a ciphertext, we assign Ctxt_i the reaction $\mathsf{enc}(k, x)$, where k is read from Key, and x is read from In_i. We leak the value of Ctxt_i to the adversary along channel Leak_i, and output the decryption of Ctxt_i under k along channel Out_i.

While the protocol in Figure 1 is written monolithically, realistic developments in IPDL define the code for each party separately. This is easy to do using the parallel composition operator $||$ that supports arbitrary interleaving of protocols. Indeed, Figure 1 is derived from a simplification of the corresponding case study in Section 5, which is specified via two parties, the sender and the receiver, and two functionalities: an authenticated network and a trusted key distribution service.

Confluence via Blocking Reads. Crucially, the semantics of $\mathsf{read}(c)$ in reactions is to block until a value is available along c. This is in contrast to UC [Canetti 2000], which operates under an actor model: in UC, protocol code can check for the absence of a message, which is disallowed in IPDL. This subtle difference in expressiveness has large consequences for the semantics of protocols. Since protocols in UC may make decisions based on the absence of a message, the order in which messages are scheduled may influence party state; in turn, any presence of nondeterminism in scheduling is a potential security leak, and has to be ruled out by enforcing a programming model that only allows one in-flight message to exist at a time. While well-understood formally [Canetti et al. 2019; Liao et al. 2019], this programming model introduces subtle complexities around timing that complicate both protocol design and security proofs.

In IPDL, we instead prove a *confluence* theorem, guaranteeing that the order in which messages are delivered (*e.g.*, whether Out_i or Leak_i fires first) cannot affect any data present in the protocol. Through confluence, we are able to express protocols in a precise, simple way, avoiding all low-level issues around the sensitivity of timing in the semantics.

While our case studies in Section 5 show that IPDL is expressive enough to capture a wide variety of cryptographic protocols, there are other protocols which are currently out of scope. Consensus-like protocols such as PBFT [Castro et al. 1999] exhibit threshold behaviors (do X if n out of m messages are received), which do not currently fit into the protocol formalism of IPDL. However, it is likely possible to expand the confluence theorem of IPDL to include theshold behaviors.

Equational Reasoning. The unique structure of protocols in IPDL is designed to enable easy equational proofs of observational equivalence. In line with the proof skeleton in Equation 1, we have two judgments for observational equality. First is exact equivalence, $\Delta \vdash P = Q$, where Δ is a *channel context*, specifying free channels common to both P and Q. Intuitively, $\Delta \vdash P = Q$ holds when P and Q coincide semantically, guaranteeing that *no* observer, regardless of resources, may distinguish them. To establish exact equivalences, we additionally use the judgment $\Delta,\ \Gamma \vdash R_1 = R_2$ for reactions, where Γ is a type context for variables.[2]

Approximate equivalence is captured through comparing two families of IPDL protocols, $\{P_\lambda\}_\lambda$ and $\{Q_\lambda\}_\lambda$. Informally, we say that $\{P_\lambda\}_\lambda \approx_\lambda \{Q_\lambda\}_\lambda$ when no polynomial time distinguisher can distinguish P_λ from Q_λ with probability greater than a negligible function of λ. Formally, we express this through the judgment $\Delta \vdash P_\lambda \approx_\lambda^{(k,l)} Q_\lambda$. Here, k and l are used to bound the size of the proof,

[2]Formally, we attach extra typing information to the judgments for both exact and approximate equivalences. We suppress them here for readability.

used for computational soundness: k bounds the number of approximate steps used, while l bounds the size of contexts used for approximate equivalences. As noted in Section 3.1, the bulk of security proofs establish exact equivalences, while approximate equivalences are mostly used to apply indistinguishability assumptions.

We will now demonstrate equational reasoning in IPDL by proving that the protocol in Figure 1 does not induce any dataflow from In_i to Leak_i. We do so by establishing an approximate equivalence to another protocol where this is guaranteed syntactically.

Decryption Soundness. The first step of the proof is to appeal to the *decryption soundness* assumption, which guarantees that encrypted values always decrypt correctly: $\mathrm{dec}(k, \mathrm{enc}(k, x)) = x$. We express this assumption in IPDL as the following exact equivalence axiom (with types suppressed):

$$\mathsf{K}, \mathsf{I}, \mathsf{C}, \mathsf{O} \vdash \big(\pi \mid\mid (\mathsf{O} \coloneqq (c \leftarrow \mathsf{read}\ \mathsf{C};\ k \leftarrow \mathsf{read}\ \mathsf{K};\ \mathsf{ret}\ (\mathrm{dec}(k, c))))\big) \ = \big(\pi \mid\mid (\mathsf{O} \coloneqq \mathsf{read}\ \mathsf{I})\big),$$

where π is the protocol

$$(\mathsf{K} \coloneqq \mathsf{samp}\ (\mathsf{unif}_{\mathsf{len}_K(\lambda)})) \mid\mid \big(\mathsf{C} \coloneqq x \leftarrow \mathsf{read}\ \mathsf{I};\ k \leftarrow \mathsf{read}\ \mathsf{K};\ \mathsf{samp}\ (\mathrm{enc}(k, x))\big).$$

Intuitively, the above equivalence states that whenever the key K is correctly sampled, any reaction which decrypts an encryption of message I may be replaced with a reaction which reads directly from I.

Since in IPDL protocol equivalence is a congruence for the connectives $\mid\mid$ and new, we apply the above axiom to Figure 1 q times to replace the definitions of Out_i with $\big|\big|_{i=1}^{q(\lambda)} (\mathsf{Out}_i \coloneqq \mathsf{read}\ \mathsf{In}_i)$.

Structural Rules. We may now apply some equational simplifications to the protocol. Since the channel Out_i no longer refers to encryption, the locally generated channel Ctxt_i that performs the ciphertext sampling is only used in one place: the leakage channel Leak_i. In this case, we are allowed to *fold* the definition of Ctxt_i into Leak_i, thereby removing this intermediate computation:

$$\Big|\Big|_{i=1}^{q(\lambda)} \big(\mathsf{Leak}_i \coloneqq x \leftarrow \mathsf{read}\ \mathsf{In}_i;\ k \leftarrow \mathsf{read}\ \mathsf{Key};\ \mathsf{samp}\ (\mathrm{enc}(k, x))\big).$$

Inlining channel definitions in this way is only permitted in certain special circumstances: *e.g.*, if the channel being inlined is not used in the rest of the protocol, as in this case, or if it does not use any probabilistic sampling.

Semantic Security. In the next step we employ a standard variant of *semantic security*: if the key K is secret, observing q encryptions of arbitrary messages is equivalent to observing q encryptions of a fixed message, *e.g.*, the all-zero message of length L. We express this in IPDL through the axiom for approximate equivalence in Figure 2. To use this axiom, we move the composition $\ldots \mid\mid \big|\big|_{i=1}^{q(\lambda)} (\mathsf{Out}_i \coloneqq \mathsf{read}\ \mathsf{In}_i)$ out of the scope of the local channel new $\mathsf{K} : \{0,1\}^{\mathsf{len}_K(\lambda)}$ in $\ldots$; we can do this since the channels Out_i no longer refer to Key. The protocol $\big|\big|_{i=1}^{q(\lambda)} (\mathsf{Out}_i ::= \mathsf{read}\ \mathsf{In}_i)$ is thus our reduction, and the axiom in Figure 2 is the indistinguishability assumption. Taking the bottom protocol from Figure 2 and moving the channels Out_i back into the scope of new $\mathsf{K} : \{0,1\}^{\mathsf{len}_K(\lambda)}$ in $\ldots$ yields the protocol in Figure 3. The leaked ciphertexts are now independent of the values of In_i, from which security follows.

4 CORE LANGUAGE AND LOGIC

IPDL is built from two main layers: *protocols* are networks of interacting *channels*, each of which is assigned a *reaction*: a simple monadic, probabilistic program that may read from other channels.

$$
\begin{aligned}
&\text{protocol EncReal}\big[\{\mathsf{I}_i : \{0,1\}^{L(\lambda)}\}_{i=1}^{q(\lambda)}, \{\mathsf{O}_i : \{0,1\}^{C(\lambda)}\}_{i=1}^{q(\lambda)}\big] \coloneqq \\
&\quad \text{new } \mathsf{K} : \{0,1\}^{\mathsf{len}_K(\lambda)} \text{ in} \\
&\qquad \big(\mathsf{K} \coloneqq \text{samp }(\mathsf{unif}_{\mathsf{len}_K(\lambda)})\big) \mid\mid \\
&\qquad \mathop{\Big|\Big|}_{i=1}^{q(\lambda)} \big(\mathsf{O}_i \coloneqq x \leftarrow \text{read } \mathsf{I}_i;\ k \leftarrow \text{read } \mathsf{K};\ \text{samp }(\mathsf{enc}(k, x))\big) \\
&\approx_\lambda \\
&\text{protocol EncZero}\big[\{\mathsf{I}_i : \{0,1\}^{L(\lambda)}\}_{i=1}^{q(\lambda)}, \{\mathsf{O}_i : \{0,1\}^{C(\lambda)}\}_{i=1}^{q(\lambda)}\big] \coloneqq \\
&\quad \text{new } \mathsf{K} : \{0,1\}^{\mathsf{len}_K(\lambda)} \text{ in} \\
&\qquad \big(\mathsf{K} \coloneqq \text{samp }(\mathsf{unif}_{\mathsf{len}_K(\lambda)})\big) \mid\mid \\
&\qquad \mathop{\Big|\Big|}_{i=1}^{q(\lambda)} \big(\mathsf{O}_i \coloneqq x \leftarrow \text{read } \mathsf{I}_i;\ k \leftarrow \text{read } \mathsf{K};\ \text{samp }(\mathsf{enc}(k, 0^L))\big)
\end{aligned}
$$

Fig. 2. Semantic Security in IPDL.

$$
\begin{aligned}
&\text{protocol P}\big[\{\mathsf{In}_i : \{0,1\}^{L(\lambda)}\}_{i=1}^{q(\lambda)}, \{\mathsf{Out}_i : \{0,1\}^{L(\lambda)}\}_{i=1}^{q(\lambda)}, \{\mathsf{Leak}_i : \{0,1\}^{C(\lambda)}\}_{i=1}^{q(\lambda)}\big] \coloneqq \\
&\quad \text{new } \mathsf{Key} : \{0,1\}^{\mathsf{len}_K(\lambda)} \text{ in} \\
&\qquad \big(\mathsf{Key} \coloneqq \text{samp }(\mathsf{unif}_{\mathsf{len}_K(\lambda)})\big) \mid\mid \\
&\qquad \mathop{\Big|\Big|}_{i=1}^{q(\lambda)} \big(\mathsf{Out}_i \coloneqq \text{read } \mathsf{In}_i\big) \mid\mid \\
&\qquad \mathop{\Big|\Big|}_{i=1}^{q(\lambda)} \big(\mathsf{Leak}_i \coloneqq \_ \leftarrow \text{read } \mathsf{In}_i;\ k \leftarrow \text{read } \mathsf{Key};\ \text{samp }(\mathsf{enc}(k, 0^L))\big)
\end{aligned}
$$

Fig. 3. The result of applying equational reasoning in IPDL to the encryption protocol in Figure 1. No dataflow dependency exists between In_i and Leak_i.

4.1 Core Syntax

The syntax of IPDL is outlined in Figure 4, and is parameterized by a user-defined *signature*, Σ:

Definition 4.1 (Signature). An IPDL signature Σ is a collection of:

- type symbols, t;
- typed function symbols, $\mathsf{f} : \tau \rightarrow \tau'$; and
- typed distribution symbols, $\mathsf{d} : \tau \rightarrow \tau'$.

We let Σ be implicitly parameterized throughout our formal developments. We assume a minimal set of data types, including the unit type 1, Booleans, products, as well as arbitrary type symbols t, drawn from the signature Σ.

Expressions are used for non-probabilistic computations, and are standard. All values in IPDL are bitstrings of a length given by data types, so we annotate the operations $\mathsf{fst}_{\tau_1 \times \tau_2}$ and $\mathsf{snd}_{\tau_1 \times \tau_2}$ with the type of the pair to determine the index to split the pair into two.

Function symbols f must appear in the signature Σ, and are assigned a typing $\Sigma \vdash \mathsf{f} : \tau \rightarrow \tau'$. We similarly assume a set of typed distribution symbols in Σ, which at least contains flip, returning bool.

Data Types	τ	::=	$1 \mid \mathsf{bool} \mid \tau \times \tau \mid \mathsf{t}$
Expressions	e	::=	$() \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{f}\ e$
		$\mid$	$(e_1, e_2) \mid \mathsf{fst}_{\tau_1 \times \tau_2}\ e \mid \mathsf{snd}_{\tau_1 \times \tau_2}\ e$
Distributions	D	::=	$\mathsf{flip} \mid \mathsf{d}\ e$
Channels	i, o, c		
Reactions	R, S	::=	$\mathsf{ret}\ (e) \mid \mathsf{samp}\ (D) \mid \mathsf{read}\ c$
		$\mid$	$\mathsf{if}\ e\ \mathsf{then}\ R_1\ \mathsf{else}\ R_2 \mid x : \tau \leftarrow R;\ S$
Protocols	P, Q	::=	$o \coloneqq R \mid P \mid\mid Q \mid \mathsf{new}\ o : \tau\ \mathsf{in}\ P$
Channel Sets	I, O	::=	$\{c_1, \ldots, c_n\}$
Channel Contexts	Δ	::=	$\cdot \mid \Delta, c : \tau$
Type Contexts	Γ	::=	$\cdot \mid \Gamma, x : \tau$
Expression Typing	$\Gamma \vdash e : \tau$		
Distribution Typing	$\Gamma \vdash D : \tau$		
Reaction Typing	$\Delta; \Gamma \vdash R : I \longrightarrow \tau$		
Protocol Typing	$\Delta \vdash P : I \longrightarrow O$		

Fig. 4. Syntax of IPDL.

As mentioned above, reactions are monadic programs which may return expressions, sample from distributions, read from channels, branch on a value of type bool, and sequentially compose.

Protocols in IPDL are given by a simple but expressive syntax: channel assignment $o \coloneqq R$ assigns the reaction R to channel o; parallel composition $P \mid\mid Q$ allows P and Q to freely interact concurrently; and channel generation new $o : \tau$ in P creates a new, internal channel for use in P. We identify protocols up to alpha equivalence of channels created by new.

Typing. We restrict our attention to well-typed IPDL reactions and protocols. In addition to respecting data types, the typing judgments guarantee that all reads from channels in reactions are in scope, and that all channels are assigned at most one reaction in protocols.

The two main typing judgments in IPDL are for reactions, $\Delta; \Gamma \vdash R : I \longrightarrow \tau$, and protocols, $\Delta; \Gamma \vdash P : I \longrightarrow O$. Here, Δ is a *channel context* – populated by free, external channels, as well as internal channels generated by new – while Γ is a *type context*, used for sequential computations inside reactions.

Figure 5 shows the typing rules for reactions. Intuitively, $\Delta; \Gamma \vdash R : I \longrightarrow \tau$ holds when R uses variables in Γ, reads from channels in I typed according to Δ, and returns a value of type τ. The typing rules for reactions are largely straightforward. We make use of standard typing rules for expressions, which we omit. Typing rules for distributions are likewise straightforward, with $\Gamma \vdash \mathsf{flip} : \mathsf{bool}$.

Figure 6 gives the typing rules for protocols: $\Delta \vdash P : I \longrightarrow O$ holds when P uses inputs in I to assign reactions to the channels in O, all typed according to Δ. Channel assignment $o \coloneqq R$ has the type $I \longrightarrow \{o\}$ when R is well-typed with an empty variable context, making use of inputs from I as well as o. We allow R to read from its own output o to express divergence: the protocol $o \coloneqq \mathsf{read}\ o$ cannot reduce, which is useful for (conditionally) deactivating certain outputs.

The typing rule for parallel composition $P \mid\mid Q$ states that P may use the outputs of Q as inputs while defining its own outputs, and vice versa. Importantly, the typing rules ensure that the outputs of P and Q are disjoint so that each channel carries a unique reaction. Finally, the rule for channel

$$\boxed{\Delta;\ \Gamma \vdash R : I \to \tau} \qquad \frac{\Gamma \vdash e : \tau}{\Delta;\ \Gamma \vdash \mathsf{ret}\ (e) : I \to \tau} \qquad \frac{\Gamma \vdash D : \tau}{\Delta;\ \Gamma \vdash \mathsf{samp}\ (D) : I \to \tau}$$

$$\frac{(i : \tau) \in \Delta \qquad i \in I}{\Delta;\ \Gamma \vdash \mathsf{read}\ i : I \to \tau} \qquad \frac{\Gamma \vdash e : \mathsf{bool} \qquad \Delta;\ \Gamma \vdash R_1 : I \to \tau \qquad \Delta;\ \Gamma \vdash R_2 : I \to \tau}{\Delta;\ \Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ R_1\ \mathsf{else}\ R_2 : I \to \tau}$$

$$\frac{\Delta;\ \Gamma \vdash R : I \to \tau \qquad \Delta;\ \Gamma, x : \tau \vdash S : I \to \sigma}{\Delta;\ \Gamma \vdash (x : \tau \leftarrow R;\ S) : I \to \sigma}$$

Fig. 5. Typing for IPDL reactions.

$$\boxed{\Delta \vdash P : I \to O} \qquad \frac{o : \tau \in \Delta \qquad o \notin I \qquad \Delta;\ \cdot \vdash R : I \cup \{o\} \to \tau}{\Delta \vdash (o \coloneqq R) : I \to \{o\}}$$

$$\frac{\Delta \vdash P : I \cup O_2 \to O_1 \qquad \Delta \vdash Q : I \cup O_1 \to O_2}{\Delta \vdash P \mid\mid Q : I \to O_1 \cup O_2} \qquad \frac{\Delta, o : \tau \vdash P : I \to O \cup \{o\}}{\Delta \vdash (\mathsf{new}\ o : A\ \mathsf{in}\ P) : I \to O}$$

Fig. 6. Typing for IPDL protocols.

generation allows a protocol to select a fresh channel name o, assign it a type τ, and use it for internal computation and communication.

Protocol typing plays a crucial role for modeling security. Simulation-based security in IPDL is modeled by existence of a *simulator* Sim with an appropriate typing judgment, $\Delta \vdash \mathsf{Sim} : I \to O$. Restricting the behavior of Sim to only use inputs along I is necessary to rule out trivial results (e.g., Sim simply copies a secret from the specification).

4.2 Semantics

The semantics of IPDL is given in two steps. First, we define an operational semantics for reactions and protocols. Our operational semantics is used to validate the *exact* fragment of our equational logic, which proves perfect observational equivalence.

The second step is to define an *interaction*, or *security game*, between an IPDL program and a resource-bounded, probabilistic *distinguisher*. The interaction semantics is used to validate *approximate* observational equivalences: these are used for cryptographic hardness assumptions, such as security of encryption schemes or Diffie-Hellman, as well as top-level statements of security for protocols.

To give semantics to user-defined symbols, we first define interpretations:

Definition 4.2 (Interpretation). An interpretation $\mathcal{I}$ for a signature Σ assigns:

- for each type symbol t, a bitstring length $[\![\mathsf{t}]\!]^{\mathcal{I}} \in \mathbb{N}$;
- to each function symbol $\mathsf{f} : \sigma \to \tau$ a function $[\![\mathsf{f}]\!]$ from bitstrings $\{0,1\}^{[\![\sigma]\!]^{\mathcal{I}}}$ to bitstrings $\{0,1\}^{[\![\tau]\!]^{\mathcal{I}}}$;
- to each distribution symbol $\mathsf{d} : \sigma \to \tau$ a function $[\![\mathsf{d}]\!]^{\mathcal{I}}$ from bitstrings $\{0,1\}^{[\![\sigma]\!]^{\mathcal{I}}}$ to *distributions* on bitstrings $\{0,1\}^{[\![\tau]\!]^{\mathcal{I}}}$.

Above, we naturally lift the interpretation $[\![\cdot]\!]^{\mathcal{I}}$ to data types by setting $[\![\mathsf{bool}]\!]^{\mathcal{I}} = 1$, $[\![1]\!]^{\mathcal{I}} = 0$, and $[\![\tau \times \tau']\!]^{\mathcal{I}} = [\![\tau]\!]^{\mathcal{I}} + [\![\tau']\!]^{\mathcal{I}}$. When the interpretation is clear from context, we omit it and simply write $[\![\cdot]\!]$.

4.2.1 Operational Semantics. To define our operational semantics, we first augment the syntax of protocols and reactions to contain intermediate values, where v is a bitstring:

$$\begin{array}{llll} \text{Expressions} & e & ::= & \ldots \mid v \\ \text{Reactions} & R & ::= & \ldots \mid \mathsf{val}(v) \\ \text{Protocols} & P & ::= & \ldots \mid o := v \end{array}$$

Throughout this section, we assume an ambient interpretation $\mathcal{I}$ for the signature Σ. Our semantics builds on a big-step semantics $e \Downarrow v$ for expressions. The semantics is standard, except that pairing is given by bitstring concatenation, and the projections fst_{τ_2}, snd_{τ_1} unambiguously split the pair according to $[\![\tau_2]\!]$ and $[\![\tau_1]\!]$, respectively. User defined function symbols f are given semantics through the ambient interpretation $\mathcal{I}$.

Our semantics uses finitely supported distributions throughout. We let $\mathsf{unit}(v)$ be the distribution assigning unit mass to v. Additionally, given a family of distributions η_i and constants $c_i \in [0, 1]$ for $i = 1 \ldots k$ with $\sum_i c_i = 1$, we let $\sum_i c_i \eta_i$ be the distribution induced by the linear combination.

We give semantics to reactions in Figure 7. Reactions have a straightforward small-step semantics of the form $R \to \eta$, where η is a probability distribution over reactions. All sums $\sum_i c_i\ \eta_i$ are implicitly finitely supported. Crucially, there is no semantic rule for stepping read c: we model communication via semantics for protocols, which substitute all instances of read for values.

Semantics for protocols are given in Figure 8. We give semantics to protocols via two main small-step rules, and a big-step rule which coordinates the small steps. First is the *output* relation $P \xmapsto{o := v} Q$, which is enabled when the reaction for channel o in P terminates, resulting in value v. When this happens, the value of o is broadcast to all other protocols set in parallel composition with P, resulting in read o commands in other reactions to be substituted with $\mathsf{val}(v)$. Note that the value of o is not broadcast above a new when the local channel is equal to o.

Secondly, we have the *internal stepping* relation $P \to \eta$, specified similarly to the small-step relation for reactions. The first rule lifts the stepping relation of R to the stepping relation for $(o := R)$, while the next three rules simply propagate the stepping relation through parallel composition and new. The last rule links the output relation with the stepping relation: whenever P steps to Q, resulting in the output $c := v$, we have that new $c : \tau$ in P steps with unit mass to new $c : \tau$ in Q.

Finally, we have the big-step relation $P \Downarrow \eta$, meaning that P takes as many steps as possible, resulting in a distribution η. The big-step relation applies as many output and internal steps as possible until the protocol cannot perform either.

Note that while the semantics for reactions is sequential, both output and internal step relations for protocols are nondeterministic. Indeed, any two channels in a protocol may produce outputs in any order. Ordinarily, this presents a problem for reasoning about cryptography, since nondeterministic choice may present a security leak. However, our language introduces *no* way to exploit this extra nondeterminism, essentially due to read commands in reactions being blocking. This is formalized by a *confluence* result for IPDL:

LEMMA 4.3 (CONFLUENCE). *If $\Delta \vdash P : I \to O$, then:*

- *If $P \xmapsto{o := v} Q$ and $P \xmapsto{o := v'} Q'$, then $v = v' \land Q = Q'$;*
- *If $P \xmapsto{o_1 := v_1} Q_1$ and $P \xmapsto{o_2 := v_2} Q_2$ with $o_1 \neq o_2$, then there exists Q such that $Q_1 \xmapsto{o_2 := v_2} Q$ and $Q_2 \xmapsto{o_1 := v_1} Q$.*

$$\boxed{R \to \eta}$$

$$\frac{e \Downarrow v}{\mathsf{ret}\ (e) \to \mathsf{unit}(\mathsf{val}(v))} \qquad \frac{e \Downarrow 1}{\mathsf{if}\ e\ \mathsf{then}\ R_1\ \mathsf{else}\ R_2 \to \mathsf{unit}(R_1)}$$

$$\frac{e \Downarrow 0}{\mathsf{if}\ e\ \mathsf{then}\ R_1\ \mathsf{else}\ R_2 \to \mathsf{unit}(R_2)} \qquad \frac{}{x : \tau \leftarrow \mathsf{val}(v);\ S \to \mathsf{unit}(S[x := v])}$$

$$\frac{D \Downarrow \sum_i c_i\ \mathsf{unit}(v_i)}{\mathsf{samp}\ (D) \to \sum_i c_i\ \mathsf{unit}(\mathsf{val}(v_i))} \qquad \frac{R \to \sum_i c_i\ \mathsf{unit}(R_i)}{x : \tau \leftarrow R;\ S \to \sum_i c_i\ \mathsf{unit}(x : \tau \leftarrow R_i;\ S)}$$

Fig. 7. Semantics for reactions.

- *If $P \xmapsto{o := v} Q$ and $P \to \eta$, then there exists η' such that $\eta \xmapsto{o := v} \eta'$ and $Q \xmapsto{\eta := '}$.*
- *If $P \to \eta_1$ and $P \to \eta_2$, then either $\eta_1 = \eta_2$, or there exists η such that $\eta_1 \to \eta$ and $\eta_2 \to \eta$.*

In the above definitions, we lift the two stepping relations $\xmapsto{o := v}$ and $\to$ to distributions in the natural way. The confluence result guarantees that the big-step relation for protocols is well-defined:

Corollary 4.4 (Determinism of $\Downarrow$). *Suppose $\Delta \vdash P : I \to O$. Then there exists a unique η such that $P \Downarrow \eta$.*

When ranging over multiple interpretations $\mathcal{I}$, we index our operational semantics by $\mathcal{I}$, obtaining $\Downarrow_{\mathcal{I}}$, $\to_{\mathcal{I}}$, and $\xmapsto{o := v}_{\mathcal{I}}$.

4.2.2 Computational Semantics. While the operational semantics is useful for validating exact observational equivalences between IPDL programs, we need more machinery to validate approximate equivalences. First, we define *distinguishers*, or resource-bounded algorithms who interact with IPDL protocols in a well-defined *interaction.*

Second, we define a notion of *size* for protocols, which constraints them to be polynomial time. Computing sizes for protocol contexts is necessary for soundness, as approximate equivalences are only sound against polynomial time distinguishers and program contexts.

We distinguish IPDL protocols using general polynomial time algorithms since IPDL protocols are not fully general. Indeed, IPDL protocols cannot test for the presence or non-presence of a value along a certain channel, while our distinguishers can. This extra expressivity in our distinguishers ensures that not only do IPDL protocols not *use* timing information, but they do not present any *leaks* through timing channels either.

Distinguishers and Interactions. Let $\mathcal{I}$ be an interpretation for Σ. Then, given channel sets I, O for channel context Δ, we define the set $\mathsf{Query}_{\mathcal{I},\Delta,I,O}$ to be:

$$\mathsf{Query}_{\mathcal{I},\Delta,I,O} := \{\mathsf{Input}(i, v) \mid i \in I, v \in \{0,1\}^{[\![\Delta(i)]\!]^{\mathcal{I}}}\} \cup \{\mathsf{Get}(o), o \in O\} \cup \{\mathsf{Step}\}.$$

Definition 4.5 (Δ-Distinguisher). Given an interpretation $\mathcal{I}$, A $(\mathcal{I}, \Delta, I, O)$-distinguisher $\mathcal{A}$ is a triple of probabilistic algorithms $(\mathcal{A}_{\mathsf{step}}, \mathcal{A}_{\mathsf{out}}, \mathcal{A}_{\mathsf{decide}})$ where:

- $\mathcal{A}_{\mathsf{step}} : \{0,1\}^* \to \{0,1\}^* \times \mathsf{Query}_{\mathcal{I},\Delta,I,O}$ takes input a state s (encoded as a bitstring), and returns a new state and a query;

$$\boxed{P \xmapsto{o := v} Q}$$

$$\frac{}{o \coloneqq \mathsf{val}(v) \xmapsto{o := v} o \coloneqq v} \qquad \frac{P \xmapsto{o := v} P'}{P \mid\mid Q \xmapsto{o := v} P' \mid\mid Q[\mathsf{read}\ o \coloneqq \mathsf{val}(v)]}$$

$$\frac{Q \xmapsto{o := v} Q'}{P \mid\mid Q \xmapsto{o := v} P[\mathsf{read}\ o \coloneqq \mathsf{val}(v)] \mid\mid Q'} \qquad \frac{P \xmapsto{o := v} Q \qquad o \neq c}{\mathsf{new}\ c : \tau\ \mathsf{in}\ P \xmapsto{o := v} \mathsf{new}\ c : \tau\ \mathsf{in}\ Q}$$

$$\boxed{P \to \eta}$$

$$\frac{R \to \sum_i c_i\ \mathsf{unit}(R_i)}{o \coloneqq R \to \sum_i c_i\ \mathsf{unit}(o \coloneqq R_i)} \qquad \frac{P \to \sum_i c_i\ \mathsf{unit}(P_i)}{P \mid\mid Q \to \sum_i c_i\ \mathsf{unit}(P_i \mid\mid Q)} \qquad \frac{Q \to \sum_i c_i\ \mathsf{unit}(Q_i)}{P \mid\mid Q \to \sum_i c_i\ \mathsf{unit}(P \mid\mid Q_i)}$$

$$\frac{P \to \sum_i c_i\ \mathsf{unit}(P_i)}{\mathsf{new}\ c : \tau\ \mathsf{in}\ P \to \sum_i c_i\ \mathsf{unit}(\mathsf{new}\ c : \tau\ \mathsf{in}\ P_i)} \qquad \frac{P \xmapsto{c := v} Q}{\mathsf{new}\ c : \tau\ \mathsf{in}\ P \to \mathsf{unit}(\mathsf{new}\ c : \tau\ \mathsf{in}\ Q)}$$

$$\boxed{P \Downarrow \eta}$$

$$\frac{P \not\to \qquad \forall o,\ v.\ P \not\xmapsto{o := v}}{P \Downarrow \mathsf{unit}(P)} \qquad \frac{P \to \sum_i c_i\ \mathsf{unit}(P_i) \qquad P_i \Downarrow \eta_i}{P \Downarrow \sum_i c_i\ \eta_i} \qquad \frac{P \xmapsto{o := v} Q \qquad Q \Downarrow \eta}{P \Downarrow \eta}$$

Fig. 8. Semantics for protocols

- $\mathcal{A}_{\mathsf{out}} : \{0,1\}^* \times (o : O) \times (1 + \{0,1\}^{[\![\Delta(o)]\!]^{\mathcal{I}}}) \to \{0,1\}^*$ takes a state s, a channel o, an optional value v for o, and returns a new state; and
- $\mathcal{A}_{\mathsf{decide}} : \{0,1\}^* \to \{0,1\}$ takes a state and returns a single bit.

We bound the running time of distinguishers by bounding the running time of each algorithm:

Definition 4.6 (k-Bounded Distinguisher). A $(\mathcal{I}, \Delta, I, O)$-distinguisher is k-bounded when its algorithms $(\mathcal{A}_{\mathsf{step}}, \mathcal{A}_{\mathsf{out}}, \mathcal{A}_{\mathsf{decide}})$ all run in at most k time steps.

For compositional reasoning, we do not wish to penalize the time bound of $\mathcal{A}$ for indexing into the channel sets I and O. Thus, in the above definitions, our model of computation for distinguishers consists of probabilistic Turing machines over the alphabet $I \cup O \cup \{0,1\}$. Increasing the alphabet size of $\mathcal{A}$ will not introduce unrealistic assumptions about our timing model: asymptotically, the size of Δ will be bounded by a polynomial (so will I and O), which allows simulation of the alphabets I and O using logarithmically many bits.

We then define the interaction of distinguishers and IPDL protocols in Figure 9.

Definition 4.7 (Interaction). Let $\mathcal{I}$ be an interpretation for the ambient signature $\Sigma, \Delta \vdash P : I \to O$, and $\mathcal{A}$ be a k-bounded $(\mathcal{I}, \Delta, I, O)$-distinguisher. Then, we let $\mathcal{A}^k(P^{\mathcal{I}})$ be the probability distribution on bits induced by the algorithm given in Figure 9.

Algorithm $\mathcal{A}^k(P^{\mathcal{I}})$:

$s := \epsilon$
For k rounds:
  $(s', q) \leftarrow \mathcal{A}_{\mathsf{step}}(s)$
  $s := s'$
  If $q = \mathsf{Input}(i, v)$:
    $P := P[\mathsf{read}\ i := \mathsf{ret}\ (v)]$
  If $q = \mathsf{Get}(o)$:
    If $(o := v) \in P$ for some v :
      $s := \mathcal{A}_{\mathsf{out}}(s, o, \mathsf{Some}(v))$
    Else :
      $s := \mathcal{A}_{\mathsf{out}}(s, o, \mathsf{None})$
  $P \leftarrow \eta$, where $P \Downarrow_{\mathcal{I}} \eta$
return $\mathcal{A}_{\mathsf{decide}}(s)$

Fig. 9. Interaction of IPDL program $\Delta \vdash P : I \to O$ with k-bounded $(\mathcal{I}, \Delta, I, O)$ distinguisher $\mathcal{A}$.

In Figure 9, we let the distinguisher interact with the protocol through a number of rounds, which we also bound by k. The distinguisher maintains a state variable s, which we initialize to the empty bitstring ε. Each round, the distinguisher outputs a query $q \in \mathsf{Query}$, which may optionally deliver an input or ask for an output If $q = \mathsf{Input}(i, v)$ with $i \in I$, we substitute read i with $\mathsf{val}(v)$ in P. (This has no effect if i already was assigned an input.) If $q = \mathsf{Get}(o)$ with $o \in O$, we check whether o has already been computed in P, which happens when $(o := v) \in P$ for some v. If such a value v exists, we output it to the distinguisher as $\mathsf{Some}(v)$; otherwise, we return None. If q is Step, we simply proceed to the next round. After k rounds, we obtain a decision bit from the distinguisher based on its current state.

To define approximate equivalence, we make use of *probabilistic polynomial-time* (PPT) families of distinguishers:

Definition 4.8 (PPT Distinguishers). Let $\{\mathcal{I}_\lambda\}$ be a family of interpretations for Σ, indexed by natural numbers λ. Additionally, let $\{\Delta_\lambda, I_\lambda, O_\lambda\}_\lambda$ be a family of channel contexts Δ_λ and channel sets for Δ_λ. Then a *PPT distinguisher* for $\{\Delta_\lambda, I_\lambda, O_\lambda\}$ is a family $\{\mathcal{A}_\lambda\}_\lambda$ such that $\mathcal{A}_\lambda$ is a $(\mathcal{I}_\lambda, \Delta_\lambda, I_\lambda, O_\lambda)$-distinguisher, along with a polynomial p such that $\mathcal{A}_\lambda$ is $p(\lambda)$-bounded for all λ.

Ensuring PPT for Protocols. To ensure that we apply approximate equivalences soundly, we need to ensure that they are only applied in polynomial-time IPDL contexts.

To capture probablistic polynomial time (PPT) for IPDL, we first consider PPT families of interpretations $\mathcal{I}_\lambda$. Intuitively, the family $\mathcal{I}_\lambda$ is PPT when it assigns polynomial lengths to type symbols t, and PPT computable functions to function symbols f and distribution symbols d.

To give semantics to distribution symbols, we need to allow for probabilistic algorithms which only succeed with *negligible* probability. Recall that a negligible function $\varepsilon : \mathbb{N} \to \mathbb{Q}$ is one that is eventually smaller than the inverse of any polynomial: $\forall K, \exists N, \forall n > N, \varepsilon(n) < \frac{1}{n^K}$.

Definition 4.9 (Realizable Distribution). Let D be a map from bitstrings to probability distributions over bitstrings, and let T be a probabilistic Turing machine. We say that T *realizes* D with error ε if, for all x and y, $|\Pr[T(x) = y] - D(x)(y)| \leq \varepsilon$.

Definition 4.10 (PPT Interpretation). Given an IPDL signature Σ, a family $\{\mathcal{I}_\lambda\}_\lambda$ of interpretations is *polynomial* if there exists a $K \in \mathbb{N}$ such that:

- for all type symbols t, $[\![\mathsf{t}]\!]^{\mathcal{I}_\lambda} \leq \lambda^K$ for all sufficiently large λ;
- for all function symbols f, that $[\![\mathsf{f}]\!]^{\mathcal{I}_\lambda}(\cdot)$ is computable by a Turing machine in time at most λ^K, for all sufficiently large λ; and
- for all distribution symbols d, there exists a negligible function ε such that for all sufficiently large λ, $[\![\mathsf{d}]\!]^{\mathcal{I}_\lambda}$ is realizable by a probabilistic Turing machine running in time at most λ^K with error $\varepsilon(\lambda)$.

Enforcing PPT for Protocols. Since IPDL protocols are finite networks of channels and do not contain recursion, we ensure polynomial time for protocols by simply counting the number of function symbols applied in reactions, and the number of channel bindings in protocols. Assuming that the interpretation $\mathcal{I}$ is bounded by a reasonable running time, we will obtain that the protocol is as well.

This count is given by a *symbolic size* $|\cdot|$, defined for expressions, reactions, and protocols. Since we assume that function and distribution symbols are PPT, our symbolic size for expressions and reactions simply counts the number of variables and function applications present in the syntax. (For example, $|\mathsf{f}\ e| := |e| + 1$, while $|(e_1, e_2)| := |e_1| + |e_2|$.) The only subtle rule is for ifs inside of reactions, where we take the max: $|\mathsf{if}\ e\ \mathsf{then}\ R_1\ \mathsf{else}\ R_2| := |e| + \max(|R_1|, |R_2|)$. We formally define symbolic size in the full version [Gancher et al. 2022].

IPDL *Contexts.* To support compositional reasoning, we additionally define typed *protocol contexts* for IPDL:

$$\text{Program Contexts} \quad C \quad ::= \quad \circ \mid \theta^\star(C) \mid C \,||\, Q \mid P \,||\, C \mid \mathsf{new}\ o : \tau\ \mathsf{in}\ C$$

Contexts are essentially protocols with a single hole. The exception is *channel embedding*, $\theta^\star(C)$, where $\theta : \Delta_1 \to \Delta_2$ is an injection from channels in a smaller context to a larger one. Channel embeddings operate naturally on programs, channel sets, and contexts, forming $\theta^\star(I)$, $\theta^\star(P)$, and $\theta^\star(C)$, respectively.

Contexts have a straightforward typing judgment $C : (\Delta \vdash I \to O) \to (\Delta' \vdash I' \to O')$ transforming well-typed protocols to well-typed protocols, given in the full version [Gancher et al. 2022]. We write $C(P)$ for the application of P to the context C. Symbolic sizes are lifted to contexts in a straightforward manner.

Given symbolic sizes for contexts, we say that the family $\{C_\lambda : (\Delta_\lambda \vdash I_\lambda \to O_\lambda) \to (\Delta'_\lambda \vdash I'_\lambda \to O'_\lambda)\}$ is PPT when there exists a polynomial p such that $|C_\lambda| \leq p(\lambda)$ for all λ, and $|\Delta_\lambda| \leq p(\lambda)$ for all λ.

Approximate Equivalence. Given two families $\{P_\lambda\}_\lambda$ and $\{Q_\lambda\}_\lambda$ of IPDL programs, we define them approximately equivalent when no PPT distinguisher can distinguish them *up to PPT contexts*:

Definition 4.11 (Approximate Equivalence). Let $\Delta_\lambda \vdash P_\lambda : I_\lambda \to O_\lambda$ and $\Delta_\lambda \vdash Q_\lambda : I_\lambda \to O_\lambda$ be two families of IPDL protocols with identical typing judgments. Then, we say that P_λ and Q_λ are *indistinguishable* under PPT interpretation, written $\mathcal{I}_\lambda; \Delta_\lambda \vDash P_\lambda \approx_\lambda Q_\lambda : I_\lambda \to O_\lambda$, when: $|\Delta_\lambda|$ is bounded by a polynomial in λ; and for any PPT family of program contexts $\{C_\lambda : (\Delta_\lambda \vdash I_\lambda \to O_\lambda) \to (\Delta'_\lambda \vdash I'_\lambda \to O'_\lambda)\}$, and for all PPT families of distinguishers $\{\mathcal{A}_\lambda\}$ for $\{\Delta'_\lambda, I_\lambda, O_\lambda\}$ bounded by $p(\cdot)$, there exists a negligible function ε such that

$$|\Pr[\mathcal{A}_\lambda^{p(\lambda)}(C_\lambda(P_\lambda)^{\mathcal{I}_\lambda})] - \Pr[\mathcal{A}_\lambda^{p(\lambda)}(C_\lambda(Q_\lambda)^{\mathcal{I}_\lambda})]| \leq \varepsilon(\lambda).$$

4.3 Equational Logic

We now present the equational logic of IPDL. The logic is divided into two halves: *exact* rules establish semantic equivalences between protocols, while *approximate* rules are used to discharge indistinguishability assumptions.

4.3.1 *Exact Equivalences.*

The bulk of the reasoning in IPDL is done using exact equivalences. We have rules for *reaction equivalence* and *protocol equivalence.*

Reaction Equivalence. Reaction equivalence $\Delta; \Gamma \vdash R_1 = R_2 : I \to \tau$ states when reactions R_1 and R_2 behave identically, reading from input channels in I and returning values of type τ. We informally highlight select rules here, and defer the formal rules to the full version [Gancher et al. 2022].

Since the nontrivial effects for reactions are reading from channels and probabilistic sampling, we have that reactions form a *commutative monad*: that is, $(x \leftarrow R_1;\ y \leftarrow R_2;\ R_3\ x\ y) = (y \leftarrow R_2;\ x \leftarrow R_1;\ R_3\ x\ y)$ holds whenever R_2 does not depend on x. All expected equivalences for commutative monads hold for reactions, including the usual monad laws and congruence of equivalence under monadic bind. The SAMP-PURE rule allows us to drop an unused sampling: $(\_ : \tau \leftarrow \mathsf{samp}\ (D);\ S) = S$. The READ-DET rule allows us to replace two reads from the same channel by a single one:

$$(x : \tau \leftarrow \mathsf{read}\ i;\ y : \tau \leftarrow \mathsf{read}\ i;\ S) = \big(x : \tau \leftarrow \mathsf{read}\ i;\ S[y := x]\big).$$

The rule FLIP-UNIF states that the distribution flip on Booleans is indeed uniform:

$$(x \leftarrow \mathsf{samp}\ (\mathsf{flip});\ \mathsf{if}\ x\ \mathsf{then\ false\ else\ true}) = \mathsf{samp}\ (\mathsf{flip}).$$

Finally, we have rules which allow us to manipulate conditionals. The rules IF-LEFT and IF-RIGHT allow us to rewrite inside of conditionals on either branch, while IF-EXT allows us to expand a conditional:

$$R[x := e] = \mathsf{if}\ e\ \mathsf{then}\ R[x := \mathsf{true}]\ \mathsf{else}\ R(x := \mathsf{false}).$$

Protocol Equivalence. Exact protocol equivalences allow reasoning about communication between subprotocols, functional correctness, and simplifying intermediate computations. We will see in Section 4.4 that exact equivalence implies the existence of a *bisimulation* on protocols, which in turn implies indistinguishability against an arbitrary distinguisher.

Our proof rules make use of *equivalence axioms*, which are used to specify user-defined assumptions about functional equivalence (e.g., correctness of decryption). We collect such axioms into an *exact theory*:

Definition 4.12 (Exact Theory). Given an ambient signature Σ, an *exact theory* $\mathbb{T}_=$ is a finite set of axioms of the form $\Delta \vdash P = Q : I \to O$, where $\Delta \vdash P : I \to O$ and $\Delta \vdash Q : I \to O$.

Our proof rules for protocol equivalence assume an ambient exact theory $\mathbb{T}_=$.

The rules for the exact equivalence of protocols are in Figures 10 and 11; we now describe them informally. The EMBED rule states that exact equivalence is invariant under channel embeddings $\theta : \Delta_1 \to \Delta_2$. The AXIOM rule incorporates axioms into the equational theory for exact equivalences.

The remaining equational rules in Figure 11 use red to distinguish the differences between the left and right hand sides. The COMP-NEW rule allows us to permute parallel composition and the creation of a new channel, and the same as *scope extrusion* in process calculi [Milner et al. 1992]. The ABSORB-LEFT rule allows us to discard a component in a parallel composition if it has no outputs; this allows us to eliminate internal channels once they are no longer used. The symmetric rule ABSORB-RIGHT (not shown) is derivable. The DIVERGE rule allows us to simplify diverging reactions: if a channel reads from itself and continues as an arbitrary reaction R, then we can safely discard R as we will never reach it in the first place.

The (un)folding rules FOLD-IF-LEFT and FOLD-BIND allow us to simplify composite reactions by bringing their components into the protocol level as separate internal channels. (We also have the symmetric rule FOLD-IF-RIGHT.) Finally, the three rules SUBSUME, SUBST, and UNUSED allow us to manipulate channel dependencies. The rule SUBSUME states that dependency is transitive: if we depend on o_1 and o_1 itself depends on o_0, then we depend on o_0 and this dependency can be made explicit.

The SUBST rule allows inlining reactions into read commands. Inlining $(o_1 := R_1)$ into $(o_2 := (x \leftarrow \mathsf{read}\ o_1;\ R_2))$ is only sound when R_1 is *duplicable*: observing two independent results

$$\boxed{\Delta \vdash P = Q : I \to O} \qquad \frac{\Delta \vdash P : I \to O}{\Delta \vdash P = P : I \to O}\ \textsc{refl} \qquad \frac{\Delta \vdash P_1 = P_2 : I \to O}{\Delta \vdash P_2 = P_1 : I \to O}\ \textsc{sym}$$

$$\frac{\Delta \vdash P_1 = P_2 : I \to O \qquad \Delta \vdash P_2 = P_3 : I \to O}{\Delta \vdash P_1 = P_3 : I \to O}\ \textsc{trans}$$

$$\frac{\vdash \theta : \Delta_1 \to \Delta_2 \qquad \Delta_1 \vdash P = Q : I \to O}{\Delta_2 \vdash \theta^\star(P) = \theta^\star(Q) : \theta^\star(I) \to \theta^\star(O)}\ \textsc{embed} \qquad \frac{(\Delta \vdash P = Q : I \to O) \in \mathbb{T}_=}{\Delta \vdash P = Q : I \to O}\ \textsc{axiom}$$

$$\frac{o : \tau \in \Delta \qquad \Delta;\ \cdot \vdash R = R' : I \cup \{o\} \to \tau}{\Delta \vdash (o \coloneqq R) = (o \coloneqq R') : I \to \{o\}}\ \textsc{cong-react}$$

$$\frac{i \notin I, O \qquad \Delta \vdash P = Q : I \to O}{\Delta \vdash P = Q : I \cup \{i\} \to O}\ \textsc{input-unused}$$

$$\frac{\Delta \vdash P = P' : I \cup O_2 \to O_1 \qquad \Delta \vdash Q : I \cup O_1 \to O_2}{\Delta \vdash P \mid\mid Q = P' \mid\mid Q : I \to O_1 \cup O_2}\ \textsc{cong-comp-left}$$

$$\frac{\Delta, o : \tau \vdash P = P' : I \to O \cup \{o\}}{\Delta \vdash (\text{new } o : \tau \text{ in } P) = (\text{new } o : \tau \text{ in } P') : I \to O}\ \textsc{cong-new}$$

$$\frac{\Delta \vdash P_1 : I \to O_1 \qquad \Delta \vdash P_2 : I \to O_2}{\Delta \vdash P_1 \mid\mid P_2 = P_2 \mid\mid P_1 : I \to O_1 \cup O_2}\ \textsc{comp-comm}$$

$$\frac{\begin{array}{c}\Delta \vdash P_1 : I \cup O_2 \cup O_3 \to O_1 \\ \Delta \vdash P_2 : I \cup O_1 \cup O_3 \to O_2 \qquad \Delta \vdash P_2 : I \cup O_1 \cup O_3 \to O_1\end{array}}{\Delta \vdash (P_1 \mid\mid P_2) \mid\mid P_3 = P_1 \mid\mid (P_2 \mid\mid P_3) : I \to O_1 \cup O_2 \cup O_3}\ \textsc{comp-assoc}$$

$$\frac{\Delta, o_1 : \tau_1, o_2 : \tau_2 \vdash P : I \to O \cup \{o_1, o_2\}}{\Delta \vdash (\text{new } o_1 : \tau_1 \text{ in new } o_2 : \tau_2 \text{ in } P) = (\text{new } o_2 : \tau_2 \text{ in new } o_1 : \tau_1 \text{ in } P) : I \to O}\ \textsc{new-exch}$$

$$\frac{\Delta \vdash P : I \cup O_2 \to O_1 \qquad \Delta, o : \tau \vdash Q : I \cup O_1 \to O_2 \cup \{o\}}{\Delta \vdash P \mid\mid (\text{new } o : \tau \text{ in } Q) = \text{new } o : \tau \text{ in } (P \mid\mid Q) : I \to O_1 \cup O_2}\ \textsc{comp-new}$$

$$\frac{\Delta \vdash P : I \to O \qquad \Delta \vdash Q : I \cup O \to \emptyset}{\Delta \vdash P \mid\mid Q = P : I \to O}\ \textsc{absorb-left}$$

$$\frac{o : \tau \in \Delta \qquad \Delta;\ \cdot \vdash R : I \cup \{o\} \to \tau}{\Delta \vdash (o \coloneqq x : \tau \leftarrow \text{read } o;\ R) = (o \coloneqq \text{read } o) : I \to \{o\}}\ \textsc{diverge}$$

Fig. 10. Exact equality for IPDL protocols. Additional rules are given in Figure 11.

$$\boxed{\Delta \vdash P = Q : I \to O}$$

$$\frac{\begin{array}{c} o : \tau \in \Delta \\ \Delta;\ \cdot \vdash R : I \cup \{o\} \to \mathsf{bool} \qquad \Delta;\ \cdot \vdash S_1 : I \cup \{o\} \to \tau \qquad \Delta;\ \cdot \vdash S_2 : I \cup \{o\} \to \tau \end{array}}{\begin{array}{c} \Delta \vdash (\mathsf{new}\ l : \tau\ \mathsf{in}\ o \coloneqq x : \mathsf{bool} \leftarrow R;\ \mathsf{if}\ x\ \mathsf{then}\ \mathsf{read}\ l\ \mathsf{else}\ S_2 \mid\mid l \coloneqq S_1) = \\ (o \coloneqq x : \mathsf{bool} \leftarrow R;\ \mathsf{if}\ x\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2) : I \to \{o\} \end{array}} \text{ FOLD-IF-LEFT}$$

$$\frac{o : B \qquad \Delta;\ \cdot \vdash R : I \cup \{o\} \to \tau \qquad \Delta;\ x : \tau \vdash S : I \cup \{o\} \to B}{\Delta \vdash (\mathsf{new}\ c : \tau\ \mathsf{in}\ o \coloneqq x : \tau \leftarrow \mathsf{read}\ c; S \mid\mid c \coloneqq R) = (o \coloneqq x : \tau \leftarrow R; S) : I \to \{o\}} \text{ FOLD-BIND}$$

$$\frac{\begin{array}{c} o_1 \neq o_2 \qquad o_0 : \tau_0, o_1 : \tau_1, o_2 : \tau_2 \in \Delta \qquad o_0 \in I \cup \{o_1, o_2\} \\ \Delta;\ x_0 : \tau_0 \vdash R_1 : I \cup \{o_1, o_2\} \to \tau_1 \qquad \Delta;\ x_1 : \tau_1 \vdash R_2 : I \cup \{o_1, o_2\} \to \tau_2 \end{array}}{\begin{array}{c} \Delta \vdash (o_1 \coloneqq x_0 : \tau_0 \leftarrow \mathsf{read}\ o_0;\ R_1 \mid\mid o_2 \coloneqq x_0 : \tau_0 \leftarrow \mathsf{read}\ o_0; x_1 : \tau_1 \leftarrow \mathsf{read}\ o_1;\ R_2) = \\ (o_1 \coloneqq x_0 : \tau_0 \leftarrow \mathsf{read}\ o_0;\ R_1 \mid\mid o_2 \coloneqq x_1 : \tau_1 \leftarrow \mathsf{read}\ o_1;\ R_2) : I \to \{o_1, o_2\} \end{array}} \text{ SUBSUME}$$

$$\frac{\begin{array}{c} o_1 \neq o_2 \qquad o_1 : \tau_1, o_2 : \tau_2 \in \Delta \\ \Delta;\ \cdot \vdash R_1 : I \cup \{o_1, o_2\} \to \tau_1 \qquad \Delta;\ x_1 : \tau_1 \vdash R_2 : I \cup \{o_1, o_2\} \to \tau_2 \\ \Delta;\ \cdot \vdash \left(x_1 \leftarrow R_1;\ x_1' \leftarrow R_1; \mathsf{ret}\ ((x_1, x_1'))\right) = \left(x_1 \leftarrow R_1;\ \mathsf{ret}\ ((x_1, x_1))\right) : I \cup \{o_1, o_2\} \to \tau_1 \times \tau_1 \end{array}}{\Delta \vdash (o_1 \coloneqq R_1 \mid\mid o_2 \coloneqq x_1 \leftarrow \mathsf{read}\ o_1; R_2) = (o_1 \coloneqq R_1 \mid\mid o_2 \coloneqq x_1 \leftarrow R_1; R_2) : I \to \{o_1, o_2\}} \text{ SUBST}$$

$$\frac{\begin{array}{c} o_1 \neq o_2 \qquad o_1 : \tau_1, o_2 : \tau_2 \in \Delta \qquad \Delta;\ \cdot \vdash R_1 : I \cup \{o_1, o_2\} \to \tau_1 \\ \Delta;\ \cdot \vdash R_2 : I \cup \{o_1, o_2\} \to \tau_2 \qquad \Delta;\ \cdot \vdash (x_1 : \tau_1 \leftarrow R_1; R_2) = R_2 : I \cup \{o_1, o_2\} \to \tau_2 \end{array}}{\Delta \vdash (o_1 \coloneqq R_1 \mid\mid o_2 \coloneqq x_1 \leftarrow \mathsf{read}\ o_1; R_2) = (o_1 \coloneqq R_1 \mid\mid o_2 \coloneqq R_2) : I \to \{o_1, o_2\}} \text{ UNUSED}$$

Fig. 11. Additional rules for exact equality of IPDL protocols. Distinguishing changes of equalities are highlighed in red.

of evaluating R_1 is equivalent to observing the same result twice. This side condition is easily discharged whenever R_1 does not contain probabilistic sampling.

Finally, the UNUSED rule allows dropping unused reads from channels. Due to timing dependencies between channels, we only allow dropping reads from $(o_1 \coloneqq R_1)$ in the context of $(o_2 \coloneqq (\_ \leftarrow \mathsf{read}\ o_1;\ R_2))$ when we have that $(\_ \leftarrow R_1;\ R_2) = R_2$. This side condition is met whenever all reads present in R_1 are also present in R_2.

4.3.2 Approximate Equivalence. We now turn to *approximate equivalence*, which establishes indistinguishability between families of protocols $\{P_\lambda\}$ and $\{Q_\lambda\}$ against PPT adversaries. Similar to exact theories $\mathbb{T}_=$, we collect axioms for indistinguishability into an *approximate theory*, $\mathbb{T}_\approx$.

Definition 4.13 (Approximate Theory). Let Σ be a signature. An *approximate theory* $\mathbb{T}_\approx$ is a finite set of axioms of the form $\{\Delta_\lambda \vdash P_\lambda \approx_\lambda Q_\lambda : I_\lambda \to O_\lambda\}$, indexed by natural numbers λ.

We establish approximate equivalences between protocol families $\{P_\lambda\}$ and $\{Q_\lambda\}$ by proving an appropriate equivalence between P_λ and Q_λ for each security parameter λ.

To maintain soundness, our main approximate equivalence judgment $\Delta \vdash P \approx_\lambda^{(k,l)} Q : I \to O$ uses two parameters, k and l to track negligibility and resource-bounded contexts. The axiom parameter, k, simply counts the number of invocations of axioms applied during the proof: k is 1 when applying a single axiom in $\mathbb{T}_\approx$, and the transitivity rule adds the two values of k together.

$$\boxed{\Delta \vdash P_1 \approx_\lambda^{(k,l)} P_2 : I \to O} \qquad \frac{\Delta \vdash_{\Sigma,\mathbb{T}} P = Q : I \to O}{\Delta \vdash P \approx_\lambda^{(0,0)} Q : I \to O}\ \textsc{strict}$$

$$\frac{\Delta \vdash P \approx_\lambda^{(k_1,l_1)} Q : I \to O \qquad k_1 \le k_2 \qquad l_1 \le l_2}{\Delta \vdash P \approx_\lambda^{(k_2,l_2)} Q : I \to O}\ \textsc{subsume} \qquad \frac{\Delta \vdash P_1 \approx_\lambda^{(k,l)} P_2 : I \to O}{\Delta \vdash P_2 \approx_\lambda^{(k,l)} P_1 : I \to O}\ \textsc{sym}$$

$$\frac{\Delta \vdash P_1 \approx_\lambda^{(k_1,l_1)} P_2 : I \to O \qquad \Delta \vdash P_2 \approx_\lambda^{(k_2,l_2)} P_3 : I \to O}{\Delta \vdash P_1 \approx_\lambda^{(k_1+k_2,\max(l_1,l_2))} P_3 : I \to O}\ \textsc{trans}$$

$$\frac{\theta : \Delta_1 \to \Delta_2 \qquad \Delta_1 \vdash P \approx_\lambda^{(k,l)} Q : I \to O}{\Delta_2 \vdash \theta^\star(P) \approx_\lambda^{(k,l)} \theta^\star(Q) : \theta^\star(I) \to \theta^\star(O)}\ \textsc{embed} \qquad \frac{\{\Delta_n \vdash P_n \approx_\lambda Q_n : I_n \to O_n\} \in \mathbb{T}_\approx}{\Delta \vdash P_\lambda \approx_\lambda^{(1,0)} Q_\lambda : I_\lambda \to O_\lambda}\ \textsc{axiom}$$

$$\frac{i \notin I \cup O \qquad \Delta \vdash P \approx_\lambda^{(k,l)} Q : I \to O}{\Delta \vdash P \approx_\lambda^{(k,l)} Q : I \cup \{i\} \to O}\ \textsc{input-unused}$$

$$\frac{\Delta \vdash P \approx_\lambda^{(k,l)} P' : I \cup O_2 \to O_1 \qquad \Delta \vdash_\Sigma Q : I \cup O_1 \to O_2}{\Delta \vdash P \mid\mid Q \approx_\lambda^{(k,l+|Q|)} P' \mid\mid Q : I \to O_1 \cup O_2}\ \textsc{cong-comp-left}$$

$$\frac{\Delta, o : A \vdash P \approx_\lambda^{(k,l)} P' : I \to O \cup \{o\}}{\Delta \vdash (\mathsf{new}\ o : A\ \mathsf{in}\ P) \approx_\lambda^{(k,l)} (\mathsf{new}\ o : A\ \mathsf{in}\ P') : I \to O}\ \textsc{cong-new}$$

$$\boxed{\vdash \{\Delta_\lambda \vdash P_\lambda \approx_\lambda Q_\lambda : I_\lambda \to O_\lambda\}} \qquad \frac{\begin{array}{c}\forall \lambda, \Delta_\lambda \vdash P_\lambda \approx_\lambda^{(k_\lambda,l_\lambda)} Q_\lambda : I_\lambda \to O_\lambda \\ k_\lambda = O(\mathsf{poly}(\lambda)) \qquad l_\lambda = O(\mathsf{poly}(\lambda)) \qquad |\Delta_\lambda| = O(\mathsf{poly}(\lambda))\end{array}}{\vdash \{\Delta_\lambda \vdash P_\lambda \approx_\lambda Q_\lambda : I_\lambda \to O_\lambda\}}$$

Fig. 12. Approximate equality for IPDL protocols.

Our soundness result will bound k by a polynomial in the security parameter to ensure that we do not apply exponentially many axioms.

The second parameter, l, tracks the largest size of protocol contexts applied to axioms in $\mathbb{T}_\approx$. While our PPT interpretations (Definition 4.10) ensure that each function symbol is PPT, exponentially large IPDL contexts can encode non-PPT computations. Thus, our soundness result also requires that l is polynomial in the security parameter to ensure that all IPDL contexts are PPT.

The top of Figure 12 shows the rules for approximate equivalence. Since most nontrivial reasoning in IPDL is done in the exact half, the approximate equivalence rules are used mostly to apply indistinguishability axioms deeply nested inside protocols. Crucially, rule STRICT allows us to descend to the exact half of the proof system.

Finally, the bottom of Figure 12 defines when two protocol families $\{P_\lambda\}$ and $\{Q_\lambda\}$ are indistinguishable. This holds when, for each choice of λ, $\Delta_\lambda \vdash P_\lambda \approx_\lambda^{(k(\lambda),l(\lambda))} Q_\lambda : I_\lambda \to O_\lambda$, and the parameters $k(\lambda)$ and $l(\lambda)$ only grow polynomially with λ, as does the size $|\Delta_\lambda|$ of each channel context.

4.4 Soundness

We now sketch our soundness results for IPDL; full proofs are given in Section B. Our main result is that our judgment for approximate equivalence is sound. To state soundness, we first need to introduce our notion of soundness for exact protocol equality:

Definition 4.14 (Protocol bisimulation). Given an interpretation $\mathcal{I}$ for a signature Σ, a *protocol bisimulation* $\sim$ is a binary relation on distributions on protocols $\Delta \vdash P : I \rightarrow O$ satisfying the following conditions:

- *Closure under joint convex combinations*: We have $\sum_{i:=1,\ldots,k} c_i \eta_i \sim \sum_{i:=1,\ldots,k} c_i \varepsilon_i$ for any coefficients $\sum_{i:=1,\ldots,k} c_i = 1$ and distributions $\eta_i \sim \varepsilon_i$ for $i := 1, \ldots, k$.
- *Closure under input assignment*: For any distributions $\eta \sim \mu$, channel $i : \tau \in \Delta$, and value $v \in \{0,1\}^{[\![\tau]\!]^{\mathcal{I}}}$, we have that $\eta[\mathsf{read}\ i := \mathsf{val}(v)] \sim \mu[\mathsf{read}\ i := \mathsf{val}(v)]$.
- *Closure under evaluation*: For any distributions $\eta \sim \mu$, if $\eta \Downarrow \eta'$ and $\mu \Downarrow \mu'$, then $\eta' \sim \mu'$.
- *Valuation property*: For any output channel o and any distributions $\eta \sim \mu$, there exists a joint convex combination $\eta = \sum_{i:=1,\ldots,k} c_i \eta_i \sim \sum_{i:=1,\ldots,k} c_i \mu_i = \mu$ such that for each $i := 1, \ldots, k$, the distributions $\eta_i \sim \mu_i$ have the same value v, or lack thereof, on the channel o.

In the above definition, we write $\eta[\mathsf{read}\ i := \mathsf{val}(v)]$ by applying the corresponding substitution pointwise to each protocol in the support of η. Similarly, we write $\eta \Downarrow \eta'$ by expressing $\eta = \sum_i c_i \mathsf{unit}(P_i)$ and evaluating $P_i \Downarrow \eta'_i$ to obtain $\eta' = \sum_i c_i \eta'_i$.

Given the above notion of protocol bisimulation, we now state when exact and approximate theories are sound:

Definition 4.15 (Soundness for Exact Theories). The exact theory $\mathbb{T}_=$ is sound if for all $\Delta \vdash P = Q : I \rightarrow O$ in $\mathbb{T}_=$, there exists a protocol bisimulation $\mathsf{unit}(P) \sim \mathsf{unit}(Q)$.

Definition 4.16 (Soundness for Approximate Theories). The approximate theory $\mathbb{T}_\approx$ is sound under PPT interpretation $\mathcal{I}_\lambda$ if, whenever $\{\Delta_\lambda \vdash P_\lambda \approx_\lambda Q_\lambda : I_\lambda \rightarrow O_\lambda\} \in \mathbb{T}_\approx$, we have that $\mathcal{I}_\lambda; \Delta_\lambda \vDash P_\lambda \approx_\lambda Q_\lambda : I_\lambda \rightarrow O_\lambda$.

Our main result is that if $\mathbb{T}_=$ and $\mathbb{T}_\approx$ are sound, then our proof rules for approximate equivalence are sound:

Theorem 4.17 (Soundness theorem for the approximate equality of IPDL protocols). *Let Σ be an* IPDL *signature, and let $\mathbb{T}_=$ and $\mathbb{T}_\approx$ be sound exact and approximate theories with respect to a PPT interpretation $\{\mathcal{I}_\lambda\}$. If $\vdash \{\Delta_\lambda \vdash P_\lambda \approx_\lambda Q_\lambda : I_\lambda \rightarrow O_\lambda\}$, then $\mathcal{I}_\lambda; \Delta_\lambda \vDash P_\lambda \approx_\lambda Q_\lambda : I_\lambda \rightarrow O_\lambda$.*

The proof of Theorem 4.17 relies on the following soundness lemmas for exact equality. First, we have that exact equality guarantees observational equivalence:

Lemma 4.18 (Observational Equivalence). *Suppose* $\mathsf{unit}(P) \sim \mathsf{unit}(Q)$ *under interpretation $\mathcal{I}$. Then, for* any *$\mathcal{A}$ and k, and any well-typed* IPDL *context C,* $\Pr[\mathcal{A}^k(C(P)^{\mathcal{I}})] = \Pr[\mathcal{A}^k(C(Q)^{\mathcal{I}})]$.

Proof. An immediate consequence of the definition of protocol bisimulation. □

Next, we have that our proof system for exact equivalence is sound:

Lemma 4.19 (Soundness of Exact Equivalence). *Suppose $\mathbb{T}_=$ is sound under interpretation $\mathcal{I}$. If $\Delta \vdash P = Q : I \rightarrow O$, then* $\mathsf{unit}(P) \sim \mathsf{unit}(Q)$.

We establish Lemma 4.19 by exhibiting a bisimulation for each proof rule. Rules Sym and Trans correspond to symmetry and transitivity lemmas for protocol bisimulation, while the congurence rules comp-cong-left and cong-new require proving corresponding congruence rules for bisimulations. For example, if $\eta \sim \mu$, then $\eta \mid\mid Q \sim \mu \mid\mid Q$ (lifting $\mid\mid$ to act on distributions).

Table 1. Case Studies in IPDL. Lines of code are separated into definitions (relevant IPDL axioms, functionalities, and protocol parties) and proofs. The last column points out nontrivial cryptographic axioms and functionalities used in the protocol.

Case study	LoC (Defs)	LoC (Proof)	Axioms/Primitives Used
A2S: CPA [Maurer 2012]	97 LoC	128 LoC	IND-CPA
A2S: DHKE [Barbosa et al. 2021b]	183 LoC	532 LoC	DDH
OT: Trapdoor [Goldreich et al. 1987]	131 LoC	517 LoC	Hard-core Predicates
OT: 1-out-of-4 [Naor and Pinkas 1999]	128	835 LoC	Underlying OT
OT: Pre-Processing [Beaver 1995]	79 LoC	401 LoC	Underlying OT
Two-Party GMW [Goldreich et al. 1987]	285 Loc	1859 LoC	Underlying OT
Multi-Party Coin Flip [Blum 1983]	114 LoC	1905 LoC	Commitments

Rules which manipulate reactions, such as SUBST, BIND, and UNUSED, require a notion of bisimulation of reactions, along with a corresponding soundness lemma for reaction equality.

5 IPDL CASE STUDIES

In this section, we briefly describe the case studies we have completed in IPDL, and outline several key proof steps that conveniently employ equational reasoning. Our case studies range from simple communication protocols to a two-party GMW protocol and a multi-party coin flip protocol. We demonstrate through lines of code that the proof effort of IPDL scales well with increasing protocol complexity, see Table 1. All lines of code count both protocol-specific definitions and proofs.

5.1 Coq Mechanization

We have mechanized the proof system of IPDL along with our case studies in Coq. The embedding is shallow: we use functions in the metalanguage instead of function symbols derived from a signature (e.g., xor over bitstrings). Channels are embedded shallowly as well, making use of an abstract type `chan t` of channels of type `t`.

Throughout our developments, we take advantage of the metalanguage to define IPDL protocols inductively based on parameters. For example, the parallel composition $||_{i=0}^{q-1} P_i$ is written in Coq as `\||_(i < q) P i`, using the bigop library from ssreflect [Mahboubi and Tassi 2021].

Due to the shallow embedding, the mechanization has a few differences from the proof rules in Section 4. The notion of size $|P|$ for protocols does not track the size of reactions, since reactions are embedded shallowly into Coq (and thus would require runtime analysis of Coq expressions). In its place, one must check that all protocols used in the approximate congruence rules COMP-CONG-LEFT/COMP-CONG-RIGHT and CONG-NEW only use efficiently computable functions, and use fixed-size reactions. This check is easily guaranteed by all of our proofs. However, we *do* capture the number of reactions used in protocols.

Additionally, we take advantage of channels being shallowly embedded to restrict inputs of protocols based on scoping in Coq, rather than restricting them via the typing judgment.

Finally, for convenience we add an additional constructor, 0, to IPDL protocols, representing an inert protocol, serving as an identity for parallel composition. The protocol 0 can easily be encoded in IPDL as new $c : \mathsf{1}$ in $c := \mathsf{ret}\ (())$.

5.2 Communication Protocols

We prove secure two different communication protocols that construct a secure communication channel from an authenticated one. The authenticated channels allow the adversary to observe in-flight messages and schedule delivery of them; in contrast, the secure communication channels only allow the adversary to observe the *presence* of the channel, but none of the message contents.

5.2.1 Secure Communication from CPA Security. This case study is a generalization of our example from Section 3 to allow for the adversary to schedule delivery of each message. In line with Section 3.1, we prove that a CPA-secure encryption scheme may be used alongside an authenticated channel to achieve a secure one. This case study is similar to one used in other proof frameworks, such as Constructive Cryptography [Maurer 2012].

5.2.2 One-Time Pad from Diffie-Hellman Key Exchange. We complete a one-time pad example using Diffie-Hellman key exchange, in comparison with EasyUC [Canetti et al. 2019] and Barbosa et al. [Barbosa et al. 2021b]. Similar to both, this example constructs a *one-time* use secure channel by performing Diffie-Hellman key exchange to establish a shared secret, then using the shared secret as a one-time pad. Our proof is similarly modular: we first prove the key exchange secure, then prove the one-time pad protocol secure, assuming an idealized key exchange. The simulator for the final protocol is naturally the composition of the simulators for the two sub-protocols.

We stress that while the comparable proof by Barbosa et al. [Barbosa et al. 2021b] is also relatively short, the IPDL proof technique requires a much lower proof *density* than their formalization. Indeed, our Coq proof involves one equational rewrite per line (e.g., substitute channel c into channel d), while their proof requires hand-written explicit bisimulations on states. While succinct once written, bisimulation relations are quite intricate and error-prone to invent.

5.3 OT Protocols

We next prove several Oblivious Transfer (OT) constructions secure. These examples are proven in the *semi-honest* (or *honest-but-curious*) setting, where we assume the parties operate correctly, but corrupted parties leak all private data to the adversary. We prove that leaked values reveal no private information about the uncorrupted parties. To encode semi-honest corruption, we augment the protocols with *leakage* functions that send all values visible to the corrupted party to the adversary. In turn, the simulator must take as input the leakages in the ideal protocol (usually minimal), and output suitable leakages in the real protocol.

In (1-out-of-2) OT, Bob wishes to obtain exactly one of Alice's two messages, without revealing his choice [Goldreich et al. 1987]. Alice doesn't learn which message Bob asked for, while Bob doesn't learn the other of the two messages. The ideal functionality simply receives the two messages m_0, m_1 from the sender, the choice bit i from the receiver, and outputs m_i. In each construction, we analyze the most interesting case when the Bob is semi-honest and the Alice is honest. Hence, the real-world leakages are derived solely from the input i coming from the receiver and the output m_i coming from the ideal functionality, with no access to any information about message m_{1-i}.

We prove the security of three main OT constructions from the literature: first, we show that 1-out-of-4 OT, which is used by our GMW example, can be realized from three instances of an ideal 1-out-of-2 OT [Naor and Pinkas 1999]; then, we show a *preprocessing* result for OT, which allows Alice and Bob to establish an OT in an offline phase, then use this OT for a fast online phase [Beaver 1995]; finally, we show that 1-out-of-2 OT can be realized using a trapdoor permutation and a hard-core bit predicate [Goldreich et al. 1987].

To illustrate how IPDL allows us to carry out probabilistic reasoning, we outline here a few key steps from the second construction. In the pre-processing phase, Alice randomly generates a new pair of keys (k_0, k_1), while Bob randomly decides on one of these keys, obtaining a choice bit j. They then use the underlying (idealized) OT to securely transfer the randomly chosen key k_j to Bob.

In the online phase, Bob encrypts his actual choice bit i by xor-ing it with j, chosen randomly in the prior phase. He sends his encrypted choice $i \oplus j$ to Alice, who responds by first swapping her two keys if $i \oplus j$ is true, then sending Bob her two keys, xor-ed with their respective messages. Bob has enough information to recover his chosen message, but the other one appears uniformly random.

To prove that Bob does not learn any information about the message he did not ask for, we carry out two probabilistic arguments. The first, which we call *decoupling*, observes that selecting two keys k_0, k_1 from the same distribution μ, and then randomly deciding to return either (k_0, k_1) or (k_1, k_0) is perfectly indistinguishable from just returning (k_0, k_1). To see this, consider the protocol below:

$$\begin{aligned}\mathsf{KeyPair} \coloneqq f \leftarrow \mathsf{samp}\ (\mathsf{flip});\ \mathsf{if}\ f\ \mathsf{then}\ k_0 \leftarrow \mathsf{samp}\ (\mu);\ k_1 \leftarrow \mathsf{samp}\ (\mu); \mathsf{ret}\ ((k_1, k_0))\\ \mathsf{else}\ k_0 \leftarrow \mathsf{samp}\ (\mu);\ k_1 \leftarrow \mathsf{samp}\ (\mu); \mathsf{ret}\ ((k_0, k_1))\end{aligned}$$

Since the two samplings inside each branch of the if are interchangeable, we may commute $k_0 \leftarrow \mathsf{samp}\ (\mu)$ with $k_1 \leftarrow \mathsf{samp}\ (\mu)$ inside of the then branch. This shows that the two branches behave exactly the same, so we may just as well not flip. We emphasize that no complex probabilistic reasoning is necessary in the argument, but only a few simple application of equational proof rules.

The second probabilistic argument concerns the distribution μ, which represents uniform randomness. Rather than modeling uniform randomness intrinsically in Coq, we only need to introduce the (sound) axiom that $\mu = (x \leftarrow \mu;\ \mathsf{unit}(x \oplus y))$ for any bitstring y. We include a full proof of this case study in our repository [Gancher et al. 2022].

5.4 Two-Party GMW Protocol

Our first large case study for IPDL is the GMW protocol [Goldreich et al. 1987], where two parties securely compute a function given by an arbitrary Boolean circuit. The protocol utilizes a 1-out-of-4 OT instance for each multiplication gate in the circuit. We analyze the GMW protocol in the semi-honest setting, with Alice (the sender of the OTs) corrupted.

Taking advantage of Coq as our metalanguage, we prove the GMW protocol secure for *arbitrary* circuits. We model circuits in Coq as finitely supported functions from wire IDs $[1, \ldots, n]$ to *operations*, where each operation may only reference wires that have been previously defined. Our model supports multiple circuit outputs and is *reactive*, in that the protocol does not dictate that all inputs must come in before starting the computation. Similarly, outputs are shared as soon as they are available, which may happen before other, unrelated inputs arrive. Our ideal functionality is similarly reactive.

The simulator for the GMW proof operates by evaluating a censored version of the real protocol in its head, having access to only Alice's private data (since she is corrupt), but not Bob's. The proof proceeds by establishing an inductive invariant between the real protocol and the ideal functionality: Bob's view of wire w in the real protocol is equal to the xor of the true value of w, along with Alice's simulated view (coming from the simulator).

5.5 Multi-Party Coin Flip Protocol

Our second large case study is for a protocol that allows an arbitrary number of mutually distrusting parties to collaboratively generate fair randomness, due to Blum [Blum 1983]. To do so, each party locally generates randomness, and commits it to all other parties. We assume an idealized commitment functionality which also bakes in a notion of broadcast, to prevent equivocation. Each party decommits their randomness once all other commitments have been collected; the output of the protocol is the Boolean sum of all decommitments.

Unlike previous examples, this example is secure in the *malicious* model. We model malicious parties by assigning them a *shell*, which simply forwards all information between the protocol and the adversary. The entire worked-out example is available in our repostiroy [Gancher et al. 2022].

5.6 Proof Effort

We have collected our case studies and their required lines of code in Table 1, in ascending order of complexity. All proof scripts for the case studies, together with the IPDL Coq library, take less than five minutes to verify on a 2020 MacBook Pro. Most proofs take only a few seconds to verify, while some – such as our Multi-Party Coin Flip example, or the 1-out-of-4 OT example – take a few minutes, due to the use of Coq tactics to aid verification.

We highlight our example of Diffie-Hellman Key Exchange + OTP, which totals 736 lines of code for definitions and proofs. This compares favorably to EasyUC [Canetti et al. 2019], which performs a similar case study using 18,000 lines of code in EasyCrypt [Barthe et al. 2011], and Barbosa et al. [Barbosa et al. 2021b], which takes over 2000 lines of code, also in EasyCrypt (albeit with reasoning about running time, which we do not explicitly perform). While difficult to compare line counts exactly, our relative simplicity is derived from the use of a high-level logic for cryptographic protocols along with a lack of hand-written bisimulations. Achieving similarly concise proofs in EasyCrypt will likely require further engineering for proof automation.

The largest examples – the Two-Party GMW, and the Multi-Party Coin Flip – are less than 2200 lines of code. While the number of lines is moderate, the complexity of the proof script is low: most of the lines consist of repetitive tactic invocations and intermediate rewriting steps. These proofs can be likely condensed further with additional proof engineering.

6 CONCLUSION AND FUTURE WORK

We introduce IPDL, a core language and proof system for equational security proofs of cryptographic protocols. Our core technical result is that IPDL is computationally sound: approximate equivalences in IPDL are sound against arbitrary probabilistic polynomial-time adversaries. We demonstrate the use of IPDL in a number of case studies, including the GMW protocol [Goldreich et al. 1987] for multi-party computation. All case studies have been mechanized in an embedding of IPDL in Coq. We now outline a few directions for future work:

Proof Automation. While we explored the use of interactive equational proofs in this work, we expect IPDL proofs to be amenable to proof automation. Indeed, directed applications of substitution and channel folding could likely drive a proof engine towards discharging many low-level equational steps, leaving the user to only specify a high-level outline of the proof.

Integration with Cryptographic Proof Assistants. As described in the introduction, IPDL is not designed to handle all subtleties of cryptographic proofs, such as rewinding, probabilistic coupling arguments, or complex cost analysis, all of which are expressible in probabilistic program logics such as EasyCrypt [Barbosa et al. 2021b; Barthe et al. 2011; Firsov and Unruh 2022]. Combining the simplicity of IPDL with the expressivity of EasyCrypt is likely to enable new proof developments which are out of reach of each system individually.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grants No. 1704788 and No. 1801369. This work was partly supported by ANR TECAP (decision number ANR-17-CE39-0004-03). This work was also supported by a Packard Fellowship and an ONR YIP award. This project was partially funded through the NGI Assure Fund, a fund established by NLnet with financial support from the European Commission's Next Generation Internet programme, under the aegis of DG Communications Networks, Content and Technology under grant agreement No 957073.

A ADDITIONAL FIGURES

$$
\begin{aligned}
|x| &:= 1 \\
|\checkmark| &:= 1 \\
|\mathsf{true}| &:= 1 \\
|\mathsf{false}| &:= 1 \\
|\mathsf{f}\ e| &:= |e| + 1 \\
|(e_1, e_2)| &:= |e_1| + |e_2| \\
|\mathsf{fst}\ e| &:= |e| + 1 \\
|\mathsf{snd}\ e| &:= |e| + 1 \\
|\mathsf{flip}| &:= 1 \\
|\mathsf{d}\ e| &:= |e| + 1
\end{aligned}
\qquad
\begin{aligned}
|\mathsf{ret}\ (e)| &:= |e| \\
|\mathsf{samp}\ (D)| &:= |D| \\
|\mathsf{read}\ c| &:= 1 \\
|\mathsf{if}\ e\ \mathsf{then}\ R_1\ \mathsf{else}\ R_2| &:= |e| + \max(|R_1|, |R_2|) \\
|x : \tau \leftarrow R;\ S| &:= |R| + |S| \\
|o := R| &:= |R| \\
|P_1 \mid\mid P_2| &:= |P_1| + |P_2| \\
|\mathsf{new}\ c : \tau\ \mathsf{in}\ P| &:= |P|
\end{aligned}
$$

$$
\begin{aligned}
|\circ| &:= 0 \\
|\theta^\star(C)| &:= |C| \\
|C \mid\mid Q| &:= |C| + |Q| \\
|P \mid\mid C| &:= |P| + |C| \\
|\mathsf{new}\ o : \tau\ \mathsf{in}\ C| &:= |C|.
\end{aligned}
$$

Fig. 13. Symbolic sizes $|\cdot|$ in IPDL. Left: sizes for expressions and distributions. Middle: sizes for reactions and protocols. Right: sizes for protocol contexts.

$$
\boxed{C : (\Delta_1 \vdash_\Sigma I_1 \to O_1) \to (\Delta_2 \vdash_\Sigma I_2 \to O_2)}
\qquad
\frac{}{\circ : (\Delta \vdash_\Sigma I \to O) \to (\Delta \vdash_\Sigma I \to O)}
$$

$$
\frac{\theta : \Delta_1 \to \Delta_2 \qquad C : (\Delta_\star \vdash_\Sigma I_\star \to O_\star) \to (\Delta_2 \vdash_\Sigma I \to O)}{\theta^\star(C) : (\Delta_\star \vdash_\Sigma I_\star \to O_\star) \to (\Delta_1 \vdash_\Sigma \theta^\star(I) \to \theta^\star(O))}
$$

$$
\frac{C : (\Delta_\star \vdash_\Sigma I_\star \to O_\star) \to (\Delta \vdash_\Sigma I \cup O_2 \to O_1) \qquad \Delta \vdash_\Sigma Q : I \cup O_1 \to O_2}{C \mid\mid Q : (\Delta_\star \vdash_\Sigma I_\star \to O_\star) \to (\Delta \vdash I \to O_1 \cup O_2)}
$$

$$
\frac{\Delta \vdash_\Sigma P : I \cup O_2 \to O_1 \qquad C : (\Delta_\star \vdash_\Sigma I_\star \to O_\star) \to (\Delta \vdash_\Sigma I \cup O_1 \to O_2)}{P \mid\mid C : (\Delta_\star \vdash_\Sigma I_\star \to O_\star) \to (\Delta \vdash_\Sigma I \to O_1 \cup O_2)}
$$

$$
\frac{C : (\Delta_\star \vdash_\Sigma I_\star \to O_\star) \to (\Delta, o : A \vdash_\Sigma I \to O \cup \{o\})}{\mathsf{new}\ o : A\ \mathsf{in}\ C : (\Delta_\star \vdash_\Sigma I_\star \to O_\star) \to (\Delta \vdash_\Sigma I \to O)}
$$

Fig. 14. Typing for IPDL contexts.

$$\boxed{\Delta;\ \Gamma \vdash R_1 = R_2 : I \longrightarrow \tau}$$

$$\frac{\Gamma \vdash e : \tau \qquad \Delta;\ \Gamma, x : \tau \vdash R : I \longrightarrow \sigma}{\Delta;\ \Gamma \vdash (x \leftarrow \mathsf{ret}\ (e);\ R) = R[x := e] : I \longrightarrow \sigma}\ \textsc{ret-bind}$$

$$\frac{\Delta;\ \Gamma \vdash R : I \longrightarrow \tau}{\Delta;\ \Gamma \vdash (x \leftarrow R;\ \mathsf{ret}\ (x)) = R : I \longrightarrow \tau}\ \textsc{bind-ret}$$

$$\frac{\Delta;\ \Gamma \vdash R_1 : I \longrightarrow \tau_1 \qquad \Delta;\ \Gamma, x_1 : \tau_1 \vdash R_2 : I \longrightarrow \tau_2 \qquad \Delta;\ \Gamma, x_2 : \tau_2 \vdash R_3 : I \longrightarrow \tau_3}{\Delta;\ \Gamma \vdash \big(x_2 : \tau_2 \leftarrow (x_1 : \tau_1 \leftarrow R_1;\ R_2);\ R_3\big) = (x_1 : \tau_1 \leftarrow R_1;\ x_2 : \tau_2 \leftarrow R_2;\ R_3) : I \longrightarrow \tau_3}\ \textsc{bind-bind}$$

$$\frac{\Delta;\ \Gamma \vdash R_1 : I \longrightarrow \tau_1 \qquad \Delta;\ \Gamma \vdash R_2 : I \longrightarrow \tau_2 \qquad \Delta;\ \Gamma, x_1 : \tau_1, x_2 : \tau_2 \vdash S : I \longrightarrow \sigma}{\Delta;\ \Gamma \vdash (x_1 : \tau_1 \leftarrow R_1;\ x_2 : \tau_2 \leftarrow R_2;\ S) = (x_2 : \tau_2 \leftarrow R_2;\ x_1 : \tau_1 \leftarrow R_1;\ S) : I \longrightarrow \sigma}\ \textsc{exch}$$

$$\frac{\Gamma \vdash d : \tau \qquad \Delta;\ \Gamma \vdash S : I \longrightarrow \sigma}{\Delta;\ \Gamma \vdash (x : \tau \leftarrow \mathsf{samp}\ (D);\ S) = S : I \longrightarrow \sigma}\ \textsc{samp-pure}$$

$$\frac{i : \tau \in \Delta \qquad i \in I \qquad \Delta;\ \Gamma, x : \tau, y : \tau \vdash S : \sigma}{\Delta;\ \Gamma \vdash (x : \tau \leftarrow \mathsf{read}\ i;\ y : \tau \leftarrow \mathsf{read}\ i;\ S) = \big(x : \tau \leftarrow \mathsf{read}\ i;\ S[y := x]\big) : I \longrightarrow \sigma}\ \textsc{read-det}$$

$$\frac{}{\Delta;\ \Gamma \vdash (x \leftarrow \mathsf{samp}\ (\mathsf{flip});\ \mathsf{if}\ x\ \mathsf{then\ false\ else\ true}) = \mathsf{samp}\ (\mathsf{flip}) : I \longrightarrow \mathsf{bool}}\ \textsc{flip-unif}$$

$$\frac{\Delta;\ \Gamma \vdash R_1 : I \longrightarrow \tau \qquad \Delta;\ \Gamma \vdash R_2 : I \longrightarrow \tau}{\Delta;\ \Gamma \vdash \mathsf{if\ true\ then}\ R_1\ \mathsf{else}\ R_2 = R_1 : I \longrightarrow \tau}\ \textsc{if-left}$$

$$\frac{\Delta;\ \Gamma \vdash R_1 : I \longrightarrow \tau \qquad \Delta;\ \Gamma \vdash R_2 : I \longrightarrow \tau}{\Delta;\ \Gamma \vdash \mathsf{if\ false\ then}\ R_1\ \mathsf{else}\ R_2 = R_2 : I \longrightarrow \tau}\ \textsc{if-right}$$

$$\frac{\Delta;\ \Gamma, x : \mathsf{bool} \vdash R : I \longrightarrow \tau \qquad \Gamma \vdash e : \mathsf{bool}}{\Delta;\ \Gamma \vdash R[x := e] = \mathsf{if}\ e\ \mathsf{then}\ R[x := \mathsf{true}]\ \mathsf{else}\ R(x := \mathsf{false}) : I \longrightarrow \tau}\ \textsc{if-ext}$$

Fig. 15. Equality for IPDL reactions.

B SOUNDNESS FOR IPDL

We prove soundness for IPDL through proving soundness for the *exact* fragment (Lemma 4.19), then proving soundness for the *approximate* fragment (Lemma 4.17).

B.1 Soundness for Exact Fragment

Throughout, we use an implicit interpretation $\mathcal{I}$, interpreting the semantics $[\![\cdot]\!]$. Soundness of equality at the expression level means that if we substitute the same valued expression for each free variable, the resulting closed expressions will compute to the same value:

Definition B.1. An axiom $\Gamma \vdash e_1 = e_2 : \tau$ is *sound* if for any valued substitution $\theta : \cdot \longrightarrow \Gamma$, we have that $\theta^\star(e_1) \Downarrow v$ and $\theta^\star(e_2) \Downarrow v$ for the same value $v \in \{0,1\}^{[\![\tau]\!]}$.

The ambient IPDL theory for expressions is said to be sound if each of its axioms is sound. It is straightforward to show that this implies overall soundness:

Lemma B.2 (Soundness of equality of expressions). *If the ambient* IPDL *theory for expressions is sound, then for any equal expressions* $\Gamma \vdash e_1 = e_2 : \tau$ *and any valued substitution* $\theta : \cdot \longrightarrow \Gamma$, *we have that* $\theta^\star(e_1) \Downarrow v$ *and* $\theta^\star(e_1) \Downarrow v$ *for the same value* $v \in \{0,1\}^{[\![\tau]\!]}$.

At the reaction level, two equal reactions should behave in a way that is indistinguishable by an external observer. We formally capture this notion of indistinguishability by a logical relation known as a *bisimulation* – a binary relation on measures on reactions that satisfies certain closure properties, together with the crucial *valuation property* that allows us to jointly partition two related measures so that any two corresponding components are again related and have the same *value*: a reaction R is said to have value v if R is of the form val v (otherwise the value is undefined), and we lift this notion to measures on reactions in the obvious way. At the reaction level, we only require the valuation property for measures that are *final*, *i.e.*, no reaction in the support steps.

We work with (finitely-supported) measures rather than just distributions purely out of convenience; importantly, however, we rule out the zero measure. We denote the total measure assigned by a measure η to the set of all reactions by $\Sigma\,\eta$.

Definition B.3 (Reaction bisimulation). A *reaction bisimulation* $\sim$ is a binary relation on measures on reactions $\Delta;\ \cdot \vdash R : I \longrightarrow \tau$ satisfying the following conditions:

- *Closure under input assignment*: For any measures $\eta \sim \varepsilon$, input channel $i \in I$ of type τ, and value $v \in \{0,1\}^{[\![\tau]\!]}$, we have $\eta[\text{read } i := \text{val } v] \sim \varepsilon[\text{read } i := \text{val } v]$.
- *Closure under computation*: For any measures $\eta \sim \varepsilon$, if $\eta \Downarrow \eta'$ and $\varepsilon \Downarrow \varepsilon'$, then $\eta' \sim \varepsilon'$.
- *Measure property*: For any measures $\eta \sim \varepsilon$, we have $\Sigma\,\eta = \Sigma\,\varepsilon$.
- *Valuation property*: For any measures $\eta \sim \varepsilon$ that are final, there exists a joint sum
$$\eta = \sum_i \eta_i \ \sim\ \sum_i \varepsilon_i = \varepsilon$$
such that
  - the respective components $\eta_i \sim \varepsilon_i$ are again related, and
  - the measures η_i and ε_j have the same value v, or lack thereof, if and only if $i = j$.

Crucially, we note that the joint sum in the valuation property is unique up to the order of the summands.

Lemma B.4. *We have the following:*

- *The identity relation is a reaction bisimulation.*
- *The inverse of a reaction bisimulation is a reaction bisimulation.*
- *The composition of two reaction bisimulations is a reaction bisimulation.*

We now describe one canonical way to construct reaction bisimulations:

Definition B.5. Let $\sim$ be an arbitrary binary relation on measures on reactions $\Delta;\ \cdot \vdash R : I \longrightarrow \tau$. The *lifting* $\sim_{\mathcal{L}}$ is the closure of $\sim$ under joint linear combinations. Explicitly, $\sim_{\mathcal{L}}$ is defined by
$$\sum_i c_i\,\eta_i \ \sim_{\mathcal{L}}\ \sum_i c_i\,\varepsilon_i$$
for coefficients $c_i > 0$ and measures $\eta_i \sim \varepsilon_i$.

Lemma B.6. *Let $\sim$ be a binary relation on measures on reactions $\Delta;\ \cdot \vdash R : I \longrightarrow \tau$ with the following properties:*

- Closure under input assignment*: For any measures $\eta \sim \varepsilon$, input channel $i \in I$ of type τ, and value $v \in \{0,1\}^{[\![\tau]\!]}$, we have $\eta[\mathsf{read}\ i \coloneqq \mathsf{val}\ v] \sim \varepsilon[\mathsf{read}\ i \coloneqq \mathsf{val}\ v]$.*
- Lifting closure under computation*: For any measures $\eta \sim \varepsilon$, if $\eta \Downarrow \eta'$ and $\varepsilon \Downarrow \varepsilon'$, then $\eta' \sim_{\mathcal{L}} \varepsilon'$.*
- Measure property*: For any measures $\eta \sim \varepsilon$, we have $\Sigma\,\eta = \Sigma\,\varepsilon$.*
- Valuation property*: For any measures $\eta \sim \varepsilon$ that are final, there exists a joint sum*

$$\eta = \sum_i \eta_i \ \sim\ \sum_i \varepsilon_i = \varepsilon$$

such that

  - *the respective components $\eta_i \sim \varepsilon_i$ are again related, and*
  - *the measures η_i and ε_j have the same value v, or lack thereof, if and only if $i = j$.*

Then the lifting $\sim_{\mathcal{L}}$ is a reaction bisimulation.

Example B.7. If the expressions $\cdot \vdash e_1 : \sigma$ and $\cdot \vdash e_2 : \sigma$ evaluate to the same value $v \in \{0,1\}^{[\![\sigma]\!]}$, then the relation $\sim$ defined by

- $\mathsf{unit}(R(x \coloneqq e_1)) \sim \mathsf{unit}(R(x \coloneqq e_2))$ for reaction $\Delta;\ x : \sigma \vdash R : I \longrightarrow \tau$

satisfies the hypotheses of Lemma B.6. In particular, $\sim_{\mathcal{L}}$ is a reaction bisimulation.

Having defined reaction bisimulations, we can now formally state what it means for reaction equality to be sound:

Definition B.8. An axiom $\Delta;\ \Gamma \vdash R_1 = R_2 : I \longrightarrow \tau$ is *sound* if there is a reaction bisimulation $\sim$ such that for any valued substitution $\theta : \cdot \longrightarrow \Gamma$, we have $\mathsf{unit}(\theta^\star(R_1)) \sim \mathsf{unit}(\theta^\star(R_2))$.

The ambient IPDL theory for reactions is said to be sound if each of its axioms is sound. We now show that this implies overall soundness:

Lemma B.9 (Soundness of equality of reactions). *If the ambient* IPDL *theory for reactions is sound, then for any equal reactions $\Delta;\ \Gamma \vdash R_1 = R_2 : I \longrightarrow \tau$, there exists a reaction bisimulation $\sim$ such that for any valued substitution $\theta : \cdot \longrightarrow \Gamma$, we have* $\mathsf{unit}(\theta^\star(R_1)) \sim \mathsf{unit}(\theta^\star(R_2))$.

Proof. We first replace the exchange rule exch by the three rules exch-samp-samp, exch-samp-read, and exch-read-read in Figure 16; it is easy to see that this new set of rules is equivalent to the original one. We now proceed by induction on the alternative set of rules for reaction equality. We will freely use a measure in place of a value (rule exch-samp-read) or a reaction (rules embed, cong-bind) to indicate the obvious lifting of the corresponding construct to measures on reactions.

- refl: Our desired bisimulation is the identity relation.
- sym: Our desired bisimulation is the inverse of the bisimulation obtained from the premise.
- trans: Our desired bisimulation is the composition of the two bisimulations obtained from the two premises.
- axiom: The desired bisimulation exists by assumption.
- subst: Our desired bisimulation is precisely the bisimulation obtained from the premise.
- embed: Let $\sim$ be the bisimulation obtained from the premise. Our desired bisimulation $\sim_\phi$ is defined by
  - $\phi^\star(\eta) \sim_\phi \phi^\star(\varepsilon)$ if $\eta \sim \varepsilon$
- input-unused: Our desired bisimulation is precisely the bisimulation obtained from the premise, seen as a bisimulation on measures on reactions with the additional input i.
- cong-ret: Our desired bisimulation is the lifting of the relation $\sim$ defined by

- $\mathsf{unit}(\mathsf{ret}\ (e)) \sim \mathsf{unit}(\mathsf{ret}\ (e'))$ for
  - expressions $\cdot \vdash e : \tau$ and $\cdot \vdash e' : \tau$, and value $v \in \{0,1\}^{[\![\tau]\!]}$ such that $e \Downarrow v$ and $e' \Downarrow v$
- $\mathsf{unit}(\mathsf{val}\ v) \sim \mathsf{unit}(\mathsf{val}\ v)$ for value $v \in \{0,1\}^{[\![\tau]\!]}$

- CONG-SAMP: Our desired bisimulation is the lifting of the relation $\sim$ defined by
  - $\mathsf{unit}(\mathsf{samp}\ ((\mathsf{d}\ e))) \sim \mathsf{unit}(\mathsf{samp}\ ((\mathsf{d}\ e')))$ for
    - expressions $\cdot \vdash e : \sigma$ and $\cdot \vdash e' : \sigma$, and value $v \in \{0,1\}^{[\![\sigma]\!]}$ such that $e \Downarrow v$ and $e' \Downarrow v$
  - $\mathsf{unit}(\mathsf{val}\ v) \sim \mathsf{unit}(\mathsf{val}\ v)$ for value $v \in \{0,1\}^{[\![\tau]\!]}$
- CONG-IF: Let $\sim_1$ and $\sim_2$ be the two bisimulations obtained from the two premises. Our desired bisimulation is the lifting of the relation $\sim_{\mathsf{if}}$ defined by
  - $\mathsf{unit}(\mathsf{if}\ e\ \mathsf{then}\ R_1\ \mathsf{else}\ R_2) \sim_{\mathsf{if}} \mathsf{unit}(\mathsf{if}\ e'\ \mathsf{then}\ R_1'\ \mathsf{else}\ R_2')$ for
    - expressions $\cdot \vdash e : \mathsf{bool}$ and $\cdot \vdash e' : \mathsf{bool}$, and value $v \in \{0,1\}$ such that $e \Downarrow v$ and $e' \Downarrow v$
    - reactions $\Delta;\ \cdot \vdash R_1 : I \longrightarrow \tau$ and $\Delta;\ \cdot \vdash R_1' : I \longrightarrow \tau$ such that $\mathsf{unit}(R_1) \sim_1 \mathsf{unit}(R_1')$
    - reactions $\Delta;\ \cdot \vdash R_2 : I \longrightarrow \tau$ and $\Delta;\ \cdot \vdash R_2' : I \longrightarrow \tau$ such that $\mathsf{unit}(R_2) \sim_2 \mathsf{unit}(R_2')$
  - $\eta_1 \sim_{\mathsf{if}} \eta_1'$ if $\eta_1 \sim_1 \eta_1'$
  - $\eta_2 \sim_{\mathsf{if}} \eta_2'$ if $\eta_2 \sim_2 \eta_2'$
- CONG-BIND: Let $\sim_1$ and $\sim_2$ be the two bisimulations obtained from the two premises. Our desired bisimulation is the lifting of the relation $\sim_{\mathsf{bind}}$ defined by
  - $(x \longleftarrow \eta;\ S) \sim_{\mathsf{bind}} (x \longleftarrow \eta';\ S')$ for
    - measures $\eta \sim_1 \eta'$
    - reactions $\Delta;\ x : \sigma \vdash S : I \longrightarrow \tau$ and $\Delta;\ x : \sigma \vdash S' : I \longrightarrow \tau$ such that for any value $v \in \{0,1\}^{[\![\sigma]\!]}$, we have $\mathsf{unit}(S(x := v)) \sim_2 \mathsf{unit}(S'(x := v))$
  - $\varepsilon \sim_{\mathsf{bind}} \varepsilon'$ if $\varepsilon \sim_2 \varepsilon'$
- RET-BIND: Our desired bisimulation is the lifting of the relation $\sim$ defined by
  - $\mathsf{unit}(x \longleftarrow \mathsf{ret}\ (e);\ R) \sim \mathsf{unit}(R(x := e))$ for expression $\cdot \vdash e : \sigma$ and reaction $\Delta;\ x : \sigma \vdash R : I \longrightarrow \tau$
  - $\mathsf{unit}(R(x := v)) \sim \mathsf{unit}(R(x := e))$ for
    - reaction $\Delta;\ x : \sigma \vdash R : I \longrightarrow \tau$
    - expression $\cdot \vdash e : \sigma$ and value $v \in \{0,1\}^{[\![\sigma]\!]}$ such that $e \Downarrow v$
- BIND-RET: Our desired bisimulation is the lifting of the relation $\sim$ defined by
  - $\mathsf{unit}(x \longleftarrow R;\ \mathsf{ret}\ (x)) \sim \mathsf{unit}(R)$ for reaction $\Delta;\ \cdot \vdash R : I \longrightarrow \tau$
  - $\mathsf{unit}(\mathsf{val}\ v) \sim \mathsf{unit}(\mathsf{val}\ v)$ for value $v \in \{0,1\}^{[\![\tau]\!]}$
- BIND-BIND: Our desired bisimulation is the lifting of the relation $\sim$ defined by
  - $\mathsf{unit}(x_2 \longleftarrow (x_1 \longleftarrow R_1;\ R_2);\ S) \sim \mathsf{unit}(x_1 \longleftarrow R_1;\ x_2 \longleftarrow R_2;\ S)$ for
    - reaction $\Delta;\ \cdot \vdash R_1 : I \longrightarrow \sigma_1$
    - reaction $\Delta;\ x_1 : \sigma_1 \vdash R_2 : I \longrightarrow \sigma_2$
    - reaction $\Delta;\ x_2 : \sigma_2 \vdash S : I \longrightarrow \tau$
  - $\mathsf{unit}(x_2 \longleftarrow R_2;\ S) \sim \mathsf{unit}(x_2 \longleftarrow R_2;\ S)$ for
    - reaction $\Delta;\ \cdot \vdash R_2 : I \longrightarrow \sigma_2$
    - reaction $\Delta;\ x_2 : \sigma_2 \vdash S : I \longrightarrow \tau$
  - $\mathsf{unit}(S) \sim \mathsf{unit}(S)$ for reaction $\Delta;\ \cdot \vdash S : I \longrightarrow \tau$
- SAMP-PURE: Our desired bisimulation is the lifting of the relation $\sim$ defined by
  - $\mathsf{unit}(x \longleftarrow \mathsf{samp}\ ((\mathsf{d}\ e));\ R) \sim \mathsf{unit}(R)$ for reaction $\Delta;\ \cdot \vdash R : I \longrightarrow \tau$
  - $\mathsf{unit}(R) \sim \mathsf{unit}(R)$ for reaction $\Delta;\ \cdot \vdash R : I \longrightarrow \tau$
- READ-DET: Our desired bisimulation is the lifting of the relation $\sim$ defined by
  - $\mathsf{unit}(x \longleftarrow \mathsf{read}\ i;\ y \longleftarrow \mathsf{read}\ i;\ R) \sim \mathsf{unit}(x \longleftarrow \mathsf{read}\ i;\ R(y := x))$ for reaction $\Delta;\ x : \sigma, y : \sigma \vdash R : I \longrightarrow \tau$
  - $\mathsf{unit}(x \longleftarrow \mathsf{val}\ v;\ y \longleftarrow \mathsf{val}\ v;\ R) \sim \mathsf{unit}(x \longleftarrow \mathsf{val}\ v;\ R(y := x))$ for

  - ∗ reaction $\Delta;\ x : \sigma, y : \sigma \vdash R : I \to \tau$
  - ∗ value $v \in \{0,1\}^{[\![\sigma]\!]}$
- – $\mathsf{unit}(R) \sim \mathsf{unit}(R)$ for reaction $\Delta;\ \cdot \vdash R : I \to \tau$
- IF-LEFT: Our desired bisimulation is the lifting of the relation ∼ defined by
  - – $\mathsf{unit}(\mathsf{if\ true\ then}\ R_1\ \mathsf{else}\ R_2) \sim \mathsf{unit}(R_1)$ for reactions $\Delta;\ \cdot \vdash R_1 : I \to \tau$ and $\Delta;\ \cdot \vdash R_2 : I \to \tau$
  - – $\mathsf{unit}(R_1) \sim \mathsf{unit}(R_1)$ for reaction $\Delta;\ \cdot \vdash R_1 : I \to \tau$
- IF-RIGHT: Our desired bisimulation is the lifting of the relation ∼ defined by
  - – $\mathsf{unit}(\mathsf{if\ false\ then}\ R_1\ \mathsf{else}\ R_2) \sim \mathsf{unit}(R_2)$ for valued reactions $\Delta;\ \cdot \vdash R_1 : I \to \tau$ and $\Delta;\ \cdot \vdash R_2 : I \to \tau$
  - – $\mathsf{unit}(R_2) \sim \mathsf{unit}(R_2)$ for reaction $\Delta;\ \cdot \vdash R_2 : I \to \tau$
- IF-EXT: Our desired bisimulation is the lifting of the relation ∼ defined by
  - – $\mathsf{unit}(R(x := e)) \sim \mathsf{unit}(\mathsf{if}\ e\ \mathsf{then}\ R(x := \mathsf{true})\ \mathsf{else}\ R(x := \mathsf{false}))$ for
    - ∗ reaction $\Delta;\ x : \mathsf{bool} \vdash R : I \to \tau$
    - ∗ expression $\cdot \vdash e : \mathsf{bool}$
  - – $\mathsf{unit}(R(x := e)) \sim \mathsf{unit}(R(x := \mathsf{true}))$ for
    - ∗ reaction $\Delta;\ x : \mathsf{bool} \vdash R : I \to \tau$
    - ∗ expression $\cdot \vdash e : \mathsf{bool}$ such that $e \Downarrow 1$
  - – $\mathsf{unit}(R(x := e)) \sim \mathsf{unit}(R(x := \mathsf{false}))$ for
    - ∗ reaction $\Delta;\ x : \mathsf{bool} \vdash R : I \to \tau$
    - ∗ expression $\cdot \vdash e : \mathsf{bool}$ such that $e \Downarrow 0$
- EXCH-SAMP-SAMP: Our desired bisimulation is the lifting of the relation ∼ defined by
  - – $\mathsf{unit}(x_1 \leftarrow \mathsf{samp}\ ((\mathsf{d}_1\ e_1));\ x_2 \leftarrow \mathsf{samp}\ ((\mathsf{d}_2\ e_2));\ \mathsf{ret}\ ((x_1, x_2))) \sim$ $\mathsf{unit}(x_2 \leftarrow \mathsf{samp}\ ((\mathsf{d}_2\ e_2));\ x_1 \leftarrow \mathsf{samp}\ ((\mathsf{d}_1\ e_1));\ \mathsf{ret}\ ((x_1, x_2)))$ for
    - ∗ expressions $\cdot \vdash e_1 : \sigma_1$ and $\cdot \vdash e_2 : \sigma_2$
  - – $\mathsf{unit}(\mathsf{val}\ v_1 v_2) \sim \mathsf{unit}(\mathsf{val}\ v_1 v_2)$ for values $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$ and $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$
- EXCH-SAMP-READ: Our desired bisimulation is the lifting of the relation ∼ defined by
  - – $\mathsf{unit}(x_1 \leftarrow \mathsf{samp}\ ((\mathsf{d}\ e));\ x_2 \leftarrow \mathsf{read}\ i;\ \mathsf{ret}\ ((x_1, x_2))) \sim \mathsf{unit}(x_2 \leftarrow \mathsf{read}\ i;\ x_1 \leftarrow \mathsf{samp}\ ((\mathsf{d}\ e));\ \mathsf{ret}\ ((x_1, x_2)))$ for
    - ∗ expression $\cdot \vdash e : \sigma$
  - – $\mathsf{unit}(x_1 \leftarrow \mathsf{samp}\ ((\mathsf{d}\ e));\ x_2 \leftarrow \mathsf{val}\ v_2;\ \mathsf{ret}\ ((x_1, x_2))) \sim \mathsf{unit}(x_2 \leftarrow \mathsf{val}\ v_2;\ x_1 \leftarrow \mathsf{samp}\ ((\mathsf{d}\ e));\ \mathsf{ret}\ ((x_1, x_2)))$ for
    - ∗ expression $\cdot \vdash e : \sigma$
    - ∗ value $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$
  - – $\big(x_2 \leftarrow \mathsf{read}\ i;\ \mathsf{ret}\ (([\![\mathsf{d}]\!](v), x_2))\big) \sim \mathsf{unit}(x_2 \leftarrow \mathsf{read}\ i;\ x_1 \leftarrow \mathsf{samp}\ ((\mathsf{d}\ e));\ \mathsf{ret}\ ((x_1, x_2)))$ for
    - ∗ expression $\cdot \vdash e : \sigma$ and value $v \in \{0,1\}^{[\![\sigma]\!]}$ such that $e \Downarrow v$
  - – $\big(x_2 \leftarrow \mathsf{val}\ v_2;\ \mathsf{ret}\ (([\![\mathsf{d}]\!](v), x_2))\big) \sim \mathsf{unit}(x_2 \leftarrow \mathsf{val}\ v_2;\ x_1 \leftarrow \mathsf{samp}\ ((\mathsf{d}\ e));\ \mathsf{ret}\ ((x_1, x_2)))$ for
    - ∗ expression $\cdot \vdash e : \sigma$ and value $v \in \{0,1\}^{[\![\sigma]\!]}$ such that $e \Downarrow v$
    - ∗ value $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$
  - – $\mathsf{unit}(\mathsf{val}\ v_1 v_2) \sim \mathsf{unit}(\mathsf{val}\ v_1 v_2)$ for values $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$ and $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$
- EXCH-READ-READ: Our desired bisimulation is the lifting of the relation ∼ defined by
  - – $\mathsf{unit}(x_1 \leftarrow \mathsf{read}\ i_1;\ x_2 \leftarrow \mathsf{read}\ i_2;\ \mathsf{ret}\ ((x_1, x_2))) \sim \mathsf{unit}(x_2 \leftarrow \mathsf{read}\ i_2;\ x_1 \leftarrow \mathsf{read}\ i_1;\ \mathsf{ret}\ ((x_1, x_2)))$
  - – $\mathsf{unit}(x_1 \leftarrow \mathsf{val}\ v_1;\ x_2 \leftarrow \mathsf{read}\ i_2;\ \mathsf{ret}\ ((x_1, x_2))) \sim \mathsf{unit}(x_2 \leftarrow \mathsf{read}\ i_2;\ x_1 \leftarrow \mathsf{val}\ v_1;\ \mathsf{ret}\ ((x_1, x_2)))$ for
    - ∗ value $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$

– $\mathsf{unit}(x_1 \leftarrow \mathsf{read}\, i_1;\ x_2 \leftarrow \mathsf{val}\, v_2;\ \mathsf{ret}\ ((x_1, x_2))) \sim \mathsf{unit}(x_2 \leftarrow \mathsf{val}\, v_2;\ x_1 \leftarrow \mathsf{read}\, i_1;\ \mathsf{ret}\ ((x_1, x_2)))$ for
  * value $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$
– $\mathsf{unit}(x_1 \leftarrow \mathsf{val}\, v_1;\ x_2 \leftarrow \mathsf{val}\, v_2;\ \mathsf{ret}\ ((x_1, x_2))) \sim \mathsf{unit}(x_2 \leftarrow \mathsf{val}\, v_2;\ x_1 \leftarrow \mathsf{val}\, v_1;\ \mathsf{ret}\ ((x_1, x_2)))$ for
  * values $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$ and $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$
– $\mathsf{unit}(x_2 \leftarrow \mathsf{read}\ i_2;\ \mathsf{ret}\ ((v_1, x_2))) \sim \mathsf{unit}(x_2 \leftarrow \mathsf{read}\ i_2;\ x_1 \leftarrow \mathsf{val}\ v_1;\ \mathsf{ret}\ ((x_1, x_2)))$ for value $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$
– $\mathsf{unit}(x_1 \leftarrow \mathsf{read}\ i_1;\ x_2 \leftarrow \mathsf{val}\ v_2;\ \mathsf{ret}\ ((x_1, x_2))) \sim \mathsf{unit}(x_1 \leftarrow \mathsf{read}\ i_1;\ \mathsf{ret}\ ((x_1, v_2)))$ for value $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$
– $\mathsf{unit}(x_2 \leftarrow \mathsf{val}\, v_2;\ \mathsf{ret}\ ((v_1, x_2))) \sim \mathsf{unit}(x_2 \leftarrow \mathsf{val}\, v_2;\ x_1 \leftarrow \mathsf{val}\, v_1;\ \mathsf{ret}\ ((x_1, x_2)))$ for
  * values $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$ and $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$
– $\mathsf{unit}(x_1 \leftarrow \mathsf{val}\, v_1;\ x_2 \leftarrow \mathsf{val}\, v_2;\ \mathsf{ret}\ ((x_1, x_2))) \sim \mathsf{unit}(x_1 \leftarrow \mathsf{val}\, v_1;\ \mathsf{ret}\ ((x_1, v_2)))$ for
  * values $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$ and $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$
– $\mathsf{unit}(\mathsf{val}\ v_1 v_2) \sim \mathsf{unit}(\mathsf{val}\ v_1 v_2)$ for values $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$ and $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$

□

$$\frac{\mathsf{d}_1 : \sigma_1 \longrightarrow \tau_1, \mathsf{d}_2 : \sigma_2 \longrightarrow \tau_2 \in \Sigma \qquad \Gamma \vdash e_1 : \sigma_1 \qquad \Gamma \vdash e_2 : \sigma_2}{\begin{array}{c}\Delta;\ \Gamma \vdash \big(x_1 : \tau_1 \leftarrow \mathsf{samp}\ ((\mathsf{d}_1\ e_1));\ x_2 : \tau_2 \leftarrow \mathsf{samp}\ ((\mathsf{d}_2\ e_2));\ \mathsf{ret}\ ((x_1, x_2))\big) = \\ \big(x_2 : \tau_2 \leftarrow \mathsf{samp}\ ((\mathsf{d}_2\ e_2));\ x_1 : \tau_1 \leftarrow \mathsf{samp}\ ((\mathsf{d}_1\ e_1));\ \mathsf{ret}\ ((x_1, x_2))\big) : I \longrightarrow \tau_1 \times \tau_2\end{array}}\ \textsc{exch-samp-samp}$$

$$\frac{\mathsf{d} : \sigma \longrightarrow \tau_1 \in \Sigma \qquad \Gamma \vdash e : \sigma \qquad i : \tau_2 \in \Delta \qquad i \in I}{\begin{array}{c}\Delta;\ \Gamma \vdash \big(x_1 : \tau_1 \longrightarrow \mathsf{samp}\ ((\mathsf{d}\ e));\ x_2 : \tau_2 \leftarrow \mathsf{read}\ i;\ \mathsf{ret}\ ((x_1, x_2))\big) = \\ \big(x_2 : \tau_2 \leftarrow \mathsf{read}\ i;\ x_1 : \tau_1 \leftarrow \mathsf{samp}\ ((\mathsf{d}\ e));\ \mathsf{ret}\ ((x_1, x_2))\big) : I \longrightarrow \tau_1 \times \tau_2\end{array}}\ \textsc{exch-samp-read}$$

$$\frac{i_1 : \tau_1, i_2 : \tau_2 \in \Delta \qquad i_1, i_2 \in I}{\begin{array}{c}\Delta;\ \Gamma \vdash \big(x_1 : \tau_1 \leftarrow \mathsf{read}\ i_1;\ x_2 : \tau_2 \leftarrow \mathsf{read}\ i_2;\ \mathsf{ret}\ ((x_1, x_2))\big) = \\ \big(x_2 : \tau_2 \leftarrow \mathsf{read}\ i_2;\ x_1 : \tau_1 \leftarrow \mathsf{read}\ i_1;\ \mathsf{ret}\ ((x_1, x_2))\big) : I \longrightarrow \tau_1 \times \tau_2\end{array}}\ \textsc{exch-read-read}$$

Fig. 16. Alternative formulation of the EXCH rule for reaction equality.

At last we get to the protocol level. A protocol bisimulation is entirely analogous to a reaction bisimulation, except we require the valuation property to hold: *i)* per output channel o, and *ii)* for all measures (not necessarily final).

Definition B.10 (Protocol bisimulation). A *protocol bisimulation* $\sim$ is a binary relation on measures on protocols $\Delta \vdash P : I \longrightarrow O$ satisfying the following conditions:

- *Closure under input assignment*: For any measures $\eta \sim \varepsilon$, input channel $i \in I$ of type τ, and value $v \in \{0,1\}^{[\![\tau]\!]}$, we have $\eta[\mathsf{read}\ i := \mathsf{val}\ v] \sim \varepsilon[\mathsf{read}\ i := \mathsf{val}\ v]$.
- *Closure under computation*: For any measures $\eta \sim \varepsilon$, if $\eta \Downarrow \eta'$ and $\varepsilon \Downarrow \varepsilon'$, then $\eta' \sim \varepsilon'$.
- *Measure property*: For any measures $\eta \sim \varepsilon$, we have $\Sigma\, \eta = \Sigma\, \varepsilon$.
- *Valuation property*: For any output channel $o \in O$, and any measures $\eta \sim \varepsilon$, there exists a joint sum
$$\eta = \sum_i \eta_i \ \sim\ \sum_i \varepsilon_i = \varepsilon$$
such that

- the respective components $\eta_i \sim \varepsilon_i$ are again related, and
- the measures η_i and ε_j have the same value v, or lack thereof, on o if and only if $i = j$.

Lemma B.11. *We have the following:*

- *The identity relation is a protocol bisimulation.*
- *The inverse of a protocol bisimulation is a protocol bisimulation.*
- *The composition of two protocol bisimulations is a protocol bisimulation.*

Definition B.12. Let $\sim$ be an arbitrary binary relation on measures on protocols $\Delta \vdash P : I \longrightarrow O$. The *lifting* $\sim_{\mathcal{L}}$ is the closure of $\sim$ under joint linear combinations. Explicitly, $\sim_{\mathcal{L}}$ is defined by

$$\sum_i c_i\,\eta_i \;\sim_{\mathcal{L}}\; \sum_i c_i\,\varepsilon_i$$

for coefficients $c_i > 0$ and measures $\eta_i \sim \varepsilon_i$.

Lemma B.13. *Let $\sim$ be a binary relation on measures on protocols $\Delta \vdash P : I \longrightarrow O$ with the following properties:*

- Closure under input assignment*: For any measures $\eta \sim \varepsilon$, input channel $i \in I$ of type τ, and value $v \in \{0,1\}^{[\![\tau]\!]}$, we have $\eta[\mathsf{read}\ i := \mathsf{val}\ v] \sim \varepsilon[\mathsf{read}\ i := \mathsf{val}\ v]$.*
- Lifting closure under computation*: For any measures $\eta \sim \varepsilon$, if $\eta \Downarrow \eta'$ and $\varepsilon \Downarrow \varepsilon'$, then $\eta' \sim_{\mathcal{L}} \varepsilon'$.*
- Measure property*: For any measures $\eta \sim \varepsilon$, we have $\Sigma\,\eta = \Sigma\,\varepsilon$.*
- Valuation property*: For any output channel $o \in O$, and any measures $\eta \sim \varepsilon$, there exists a joint sum*

$$\eta = \sum_i \eta_i \;\sim\; \sum_i \varepsilon_i = \varepsilon$$

  such that
  - *the respective components $\eta_i \sim \varepsilon_i$ are again related, and*
  - *the measures η_i and ε_j have the same value v, or lack thereof, on o if and only if $i = j$.*

Then the lifting $\sim_{\mathcal{L}}$ is a protocol bisimulation.

We can now formally state what it means for exact protocol equality to be sound:

Definition B.14. An axiom $\Delta \vdash P_1 = P_2 : I \longrightarrow O$ is *sound* if there is a protocol bisimulation $\sim$ such that $\mathsf{unit}(P_1) \sim \mathsf{unit}(P_2)$.

The ambient IPDL theory for protocols is said to be sound if each of its axioms is sound. We now show that this implies overall soundness for exact equality:

Lemma B.15 (Soundness of exact equality of protocols). *If the ambient* IPDL *theory for protocols is sound, then for any equal protocols $\Delta \vdash P_1 = P_2 : I \longrightarrow O$, there exists a protocol bisimulation $\sim$ such that* $\mathsf{unit}(P_1) \sim \mathsf{unit}(P_2)$.

Proof. We first replace the rules fold-if-left and fold-if-right by the equivalent formulation in Figure 17. We now proceed by induction on this alternative set of rules for exact protocol equality. We will freely use a measure in place of a reaction (rule cong-react) or a protocol (rules embed, absorb-left) to indicate the obvious lifting of the corresponding construct to measures on protocols.

- refl: Our desired bisimulation is the identity relation.
- sym: Our desired bisimulation is the inverse of the bisimulation obtained from the premise.
- trans: Our desired bisimulation is the composition of the two bisimulations obtained from the two premises.
- axiom: The desired bisimulation exists by assumption.

- EMBED: Let $\sim$ be the bisimulation obtained from the premise. Our desired bisimulation $\sim_\phi$ is defined by
  - $\phi^\star(\eta) \sim_\phi \phi^\star(\varepsilon)$ if $\eta \sim \varepsilon$
- INPUT-UNUSED: Our desired bisimulation is precisely the bisimulation obtained from the premise, seen as a bisimulation on measures on protocols with the additional input i.
- CONG-REACT: Let $\sim$ be the reaction bisimulation obtained from the premise. Our desired bisimulation is the lifting of the relation $\sim_{\text{react}}$ defined by
  - $(o \coloneqq \eta) \sim_{\text{react}} (o \coloneqq \eta')$ for measures $\eta \sim \eta'$
  - $\mathsf{unit}(o \coloneqq v) \sim_{\text{react}} \mathsf{unit}(o \coloneqq v)$ for value $v \in \{0,1\}^{[\![\tau]\!]}$
- CONG-COMP-LEFT: Let $\sim$ be the bisimulation obtained from the premise. Our desired bisimulation is the lifting of the relation $\sim_{\text{par}}$ defined by
  - $(\eta \mid\mid Q) \sim_{\text{par}} (\eta' \mid\mid Q)$ for $\eta \sim \eta'$ and protocol $\Delta \vdash Q : I \cup O_1 \to O_2$
- CONG-NEW: Let $\sim$ be the bisimulation obtained from the premise. Our desired bisimulation $\sim_{\text{new}}$ is defined by
  - $(\mathsf{new}\ o : \tau\ \mathsf{in}\ \eta) \sim_{\text{new}} (\mathsf{new}\ o : \tau\ \mathsf{in}\ \eta')$ if $\eta \sim \eta'$
- COMP-COMM: Our desired bisimulation is the lifting of the relation $\sim$ defined by
  - $\mathsf{unit}(P_1 \mid\mid P_2) \sim \mathsf{unit}(P_2 \mid\mid P_1)$ for protocols $\Delta \vdash P_1 : I \cup O_2 \to O_1$ and $\Delta \vdash P_2 : I \cup O_1 \to O_2$
- COMP-ASSOC: Our desired bisimulation is the lifting of the relation $\sim$ defined by
  - $1\big[(P_1 \mid\mid P_2) \mid\mid P_3\big] \sim 1\big[P_1 \mid\mid (P_2 \mid\mid P_3)\big]$ for
    - protocol $\Delta \vdash P_1 : I \cup O_2 \cup O_3 \to O_1$
    - protocol $\Delta \vdash P_2 : I \cup O_1 \cup O_3 \to O_2$
    - protocol $\Delta \vdash P_3 : I \cup O_1 \cup O_2 \to O_3$
- NEW-EXCH: The desired bisimulation is the lifting of the relation $\sim$ defined by
  - $\mathsf{unit}(\mathsf{new}\ o_1 : \tau_1\ \mathsf{in}\ \mathsf{new}\ o_2 : \tau_2\ \mathsf{in}\ P) \sim \mathsf{unit}(\mathsf{new}\ o_2 : \tau_2\ \mathsf{in}\ \mathsf{new}\ o_1 : \tau_1\ \mathsf{in}\ P)$ for
    - protocol $\Delta, o_1 : \tau_1, o_2 : \tau_2 \vdash P : I \to O \cup \{o_1, o_2\}$
- COMP-NEW: Our desired bisimulation is the lifting of the relation $\sim$ defined by
  - $\mathsf{unit}(P \mid\mid (\mathsf{new}\ o : \tau\ \mathsf{in}\ Q)) \sim \mathsf{unit}(\mathsf{new}\ o : \tau\ \mathsf{in}\ (P \mid\mid Q))$ for
    - protocol $\Delta \vdash P : I \cup O_2 \to O_1$
    - protocol $\Delta, o : \tau \vdash Q : I \cup O_1 \to O_2 \cup \{o\}$
- ABSORB-LEFT: Our desired bisimulation is the lifting of the relation $\sim$ defined by
  - $\mathsf{unit}(P \mid\mid Q) \sim \mathsf{unit}(P)$ for protocols $\Delta \vdash P : I \to O$ and $\Delta \vdash Q : I \cup O \to \emptyset$
- DIVERGE: Our desired bisimulation is the lifting of the relation $\sim$ defined by
  - $\mathsf{unit}(o \coloneqq x \leftarrow \mathsf{read}\ o;\ R) \sim \mathsf{unit}(o \coloneqq \mathsf{read}\ o)$ for reaction $\Delta;\ \cdot \vdash R : I \cup \{o\} \to \tau$
- FOLD-IF-LEFT: Our desired bisimulation is the lifting of the relation $\sim$ defined by
  - $\mathsf{unit}(\mathsf{new}\ l : \tau\ \mathsf{in}\ o \coloneqq x \leftarrow \mathsf{read}\ b;\ \mathsf{if}\ x\ \mathsf{then}\ \mathsf{read}\ l\ \mathsf{else}\ S_2 \mid\mid l \coloneqq x \leftarrow \mathsf{read}\ b;\ S_1) \sim \mathsf{unit}(o \coloneqq x \leftarrow \mathsf{read}\ b;\ \mathsf{if}\ x\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2)$ for
    - reaction $\Delta;\ \cdot \vdash S_1 : I \cup \{o\} \to \tau$
    - reaction $\Delta;\ \cdot \vdash S_2 : I \cup \{o\} \to \tau$
  - $\mathsf{unit}(\mathsf{new}\ l : \tau\ \mathsf{in}\ o \coloneqq x \leftarrow \mathsf{val}\ v;\ \mathsf{if}\ x\ \mathsf{then}\ \mathsf{read}\ l\ \mathsf{else}\ S_2 \mid\mid l \coloneqq x \leftarrow \mathsf{val}\ v;\ S_1) \sim \mathsf{unit}(o \coloneqq x \leftarrow \mathsf{val}\ v;\ \mathsf{if}\ x\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2)$ for
    - value $v \in \{0, 1\}$
    - reaction $\Delta;\ \cdot \vdash S_1 : I \cup \{o\} \to \tau$
    - reaction $\Delta;\ \cdot \vdash S_2 : I \cup \{o\} \to \tau$
  - $\mathsf{unit}(\mathsf{new}\ l : \tau\ \mathsf{in}\ o \coloneqq \mathsf{read}\ l \mid\mid l \coloneqq S_1) \sim \mathsf{unit}(o \coloneqq S_1)$ for reaction $\Delta;\ \cdot \vdash S_1 : I \cup \{o\} \to \tau$
  - $\mathsf{unit}(\mathsf{new}\ l : \tau\ \mathsf{in}\ o \coloneqq S_2 \mid\mid l \coloneqq S_1) \sim \mathsf{unit}(o \coloneqq S_2)$ for
    - reaction $\Delta;\ \cdot \vdash S_1 : I \cup \{o\} \to \tau$
    - reaction $\Delta;\ \cdot \vdash S_2 : I \cup \{o\} \to \tau$

- $\mathsf{unit}(\mathsf{new}\ l : \tau\ \mathsf{in}\ o \coloneqq v_2 \mid\mid l \coloneqq S_1) \sim \mathsf{unit}(o \coloneqq v_2)$ for
  * reaction $\Delta;\ \cdot \vdash S_1 : I \cup \{o\} \rightarrow \tau$
  * value $v_2 \in \{0,1\}^{[\![\tau]\!]}$
- $\mathsf{unit}(\mathsf{new}\ l : \tau\ \mathsf{in}\ o \coloneqq S_2 \mid\mid l \coloneqq v_1) \sim \mathsf{unit}(o \coloneqq S_2)$ for
  * value $v_1 \in \{0,1\}^{[\![\tau]\!]}$
  * reaction $\Delta;\ \cdot \vdash S_2 : I \cup \{o\} \rightarrow \tau$
- $\mathsf{unit}(\mathsf{new}\ l : \tau\ \mathsf{in}\ o \coloneqq v_2 \mid\mid l \coloneqq v_1) \sim \mathsf{unit}(o \coloneqq v_2)$ for values $v_1, v_2 \in \{0,1\}^{[\![\tau]\!]}$

• FOLD-IF-RIGHT: Our desired bisimulation is the lifting of the relation $\sim$ defined by
- $\mathsf{unit}(\mathsf{new}\ r : \tau\ \mathsf{in}\ o \coloneqq x \leftarrow \mathsf{read}\ b;\ \mathsf{if}\ x\ \mathsf{then}\ S_1\ \mathsf{else}\ \mathsf{read}\ r \mid\mid r \coloneqq x \leftarrow \mathsf{read}\ b;\ S_2) \sim$ $\mathsf{unit}(o \coloneqq x \leftarrow \mathsf{read}\ b;\ \mathsf{if}\ x\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2)$ for
  * reaction $\Delta;\ \cdot \vdash S_1 : I \cup \{o\} \rightarrow \tau$
  * reaction $\Delta;\ \cdot \vdash S_2 : I \cup \{o\} \rightarrow \tau$
- $\mathsf{unit}(\mathsf{new}\ r : \tau\ \mathsf{in}\ o \coloneqq x \leftarrow \mathsf{val}\ v;\ \mathsf{if}\ x\ \mathsf{then}\ S_1\ \mathsf{else}\ \mathsf{read}\ r \mid\mid r \coloneqq x \leftarrow \mathsf{val}\ v;\ S_2) \sim$ $\mathsf{unit}(o \coloneqq x \leftarrow \mathsf{val}\ v;\ \mathsf{if}\ x\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2)$ for
  * value $v \in \{0,1\}$
  * reaction $\Delta;\ \cdot \vdash S_1 : I \cup \{o\} \rightarrow \tau$
  * reaction $\Delta;\ \cdot \vdash S_2 : I \cup \{o\} \rightarrow \tau$
- $\mathsf{unit}(\mathsf{new}\ r : \tau\ \mathsf{in}\ o \coloneqq \mathsf{read}\ r \mid\mid r \coloneqq S_2) \sim \mathsf{unit}(o \coloneqq S_2)$ for reaction $\Delta;\ \cdot \vdash S_2 : I \cup \{o\} \rightarrow \tau$
- $\mathsf{unit}(\mathsf{new}\ r : \tau\ \mathsf{in}\ o \coloneqq S_1 \mid\mid r \coloneqq S_2) \sim \mathsf{unit}(o \coloneqq S_1)$ for
  * reaction $\Delta;\ \cdot \vdash S_1 : I \cup \{o\} \rightarrow \tau$
  * reaction $\Delta;\ \cdot \vdash S_2 : I \cup \{o\} \rightarrow \tau$
- $\mathsf{unit}(\mathsf{new}\ r : \tau\ \mathsf{in}\ o \coloneqq v_1 \mid\mid r \coloneqq S_2) \sim \mathsf{unit}(o \coloneqq v_1)$ for
  * value $v_1 \in \{0,1\}^{[\![\tau]\!]}$
  * reaction $\Delta;\ \cdot \vdash S_2 : I \cup \{o\} \rightarrow \tau$
- $\mathsf{unit}(\mathsf{new}\ r : \tau\ \mathsf{in}\ o \coloneqq S_1 \mid\mid r \coloneqq v_2) \sim \mathsf{unit}(o \coloneqq S_1)$ for
  * reaction $\Delta;\ \cdot \vdash S_1 : I \cup \{o\} \rightarrow \tau$
  * value $v_2 \in \{0,1\}^{[\![\tau]\!]}$
- $\mathsf{unit}(\mathsf{new}\ r : \tau\ \mathsf{in}\ o \coloneqq v_1 \mid\mid r \coloneqq v_2) \sim \mathsf{unit}(o \coloneqq v_1)$ for values $v_1, v_2 \in \{0,1\}^{[\![\tau]\!]}$

• FOLD-BIND: Our desired bisimulation is the lifting of the relation $\sim$ defined by
- $\mathsf{unit}(\mathsf{new}\ c : \tau_1\ \mathsf{in}\ o \coloneqq x \leftarrow \mathsf{read}\ c;\ R_2 \mid\mid c \coloneqq R_1) \sim \mathsf{unit}(o \coloneqq x \leftarrow R_1;\ R_2)$ for
  * reaction $\Delta;\ \cdot \vdash R_1 : I \cup \{o\} \rightarrow \tau_1$
  * reaction $\Delta;\ x : \tau_1 \vdash R_2 : I \cup \{o\} \rightarrow \tau_2$
- $\mathsf{unit}(\mathsf{new}\ c : \tau_1\ \mathsf{in}\ o \coloneqq R_2 \mid\mid c \coloneqq v_1) \sim \mathsf{unit}(o \coloneqq R_2)$ for
  * value $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$
  * reaction $\Delta;\ \cdot \vdash R_2 : I \cup \{o\} \rightarrow \tau_2$
- $\mathsf{unit}(\mathsf{new}\ c : \tau_1\ \mathsf{in}\ o \coloneqq v_2 \mid\mid c \coloneqq v_1) \sim \mathsf{unit}(o \coloneqq v_2)$ for
  * values $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$ and $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$

• SUBSUME: Our desired bisimulation is the lifting of the relation $\sim$ defined by
- $\mathsf{unit}(o_1 \coloneqq x_0 \leftarrow \mathsf{read}\ o_0;\ R_1 \mid\mid o_2 \coloneqq x_0 \leftarrow \mathsf{read}\ o_0;\ x_1 \leftarrow \mathsf{read}\ o_1;\ R_2) \sim$ $\mathsf{unit}(o_1 \coloneqq x_0 \leftarrow \mathsf{read}\ o_0;\ R_1 \mid\mid o_2 \coloneqq x_1 \leftarrow \mathsf{read}\ o_1;\ R_2)$ for
  * reaction $\Delta;\ x_0 : \tau_0 \vdash R_1 : I \cup \{o_1, o_2\} \rightarrow \tau_1$
  * reaction $\Delta;\ x_1 : \tau_1 \vdash R_2 : I \cup \{o_1, o_2\} \rightarrow \tau_2$
- $\mathsf{unit}(o_1 \coloneqq x_0 \leftarrow \mathsf{val}\ v_0;\ R_1 \mid\mid o_2 \coloneqq x_0 \leftarrow \mathsf{val}\ v_0;\ x_1 \leftarrow \mathsf{read}\ o_1;\ R_2) \sim$ $\mathsf{unit}(o_1 \coloneqq x_0 \leftarrow \mathsf{val}\ v_0;\ R_1 \mid\mid o_2 \coloneqq x_1 \leftarrow \mathsf{read}\ o_1;\ R_2)$ for
  * value $v_0 \in \{0,1\}^{[\![\tau_0]\!]}$
  * reaction $\Delta;\ x_0 : \tau_0 \vdash R_1 : I \cup \{o_1, o_2\} \rightarrow \tau_1$
  * reaction $\Delta;\ x_1 : \tau_1 \vdash R_2 : I \cup \{o_1, o_2\} \rightarrow \tau_2$
- $\mathsf{unit}(o_1 \coloneqq R_1 \mid\mid o_2 \coloneqq x_1 \leftarrow \mathsf{read}\ o_1;\ R_2) \sim \mathsf{unit}(o_1 \coloneqq R_1 \mid\mid o_2 \coloneqq x_1 \leftarrow \mathsf{read}\ o_1;\ R_2)$ for

    * reaction $\Delta;\ \cdot \vdash R_1 : I \cup \{o_1, o_2\} \longrightarrow \tau_1$
    * reaction $\Delta;\ x_1 : \tau_1 \vdash R_2 : I \cup \{o_1, o_2\} \longrightarrow \tau_2$
  - $\mathsf{unit}(o_1 := v_1 \mid\mid o_2 := R_2) \sim \mathsf{unit}(o_1 := v_1 \mid\mid o_2 := R_2)$ for
    * value $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$
    * reaction $\Delta;\ \cdot \vdash R_2 : I \cup \{o_1, o_2\} \longrightarrow \tau_2$
  - $\mathsf{unit}(o_1 := v_1 \mid\mid o_2 := v_2) \sim \mathsf{unit}(o_1 := v_1 \mid\mid o_2 := v_2)$ for values $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$ and $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$
- Subst: Let $\sim$ be the reaction bisimulation obtained from the premise. Our desired bisimulation is the lifting of the relation $\sim_{\mathsf{subst}}$ defined by
  - $\big(o_1 := \eta \mid\mid o_2 := x_1 \leftarrow \mathsf{read}\ o_1;\ R_2\big) \sim_{\mathsf{subst}} \big(o_1 := \eta \mid\mid o_2 := x_1 \leftarrow \eta;\ R_2\big)$ for
    * distribution η on reactions $\Delta;\ \cdot \vdash R_1 : I \cup \{o_1, o_2\} \longrightarrow \tau_1$
    * reaction $\Delta;\ \cdot \vdash R_1 : I \cup \{o_1, o_2\} \longrightarrow \tau_1$ evaluating to the same distribution as η
    * reaction $\Delta;\ x_1 : \tau_1 \vdash R_2 : I \cup \{o_1, o_2\} \longrightarrow \tau_2$

    such that $\mathsf{unit}(x_1 \leftarrow R_1;\ x_1' \leftarrow R_1;\ \mathsf{ret}\ ((x_1, x_1'))) \sim \mathsf{unit}(x_1 \leftarrow R_1;\ \mathsf{ret}\ ((x_1, x_1)))$
  - $\mathsf{unit}(o_1 := v_1 \mid\mid o_2 := R_2) \sim_{\mathsf{subst}} \mathsf{unit}(o_1 := v_1 \mid\mid o_2 := R_2)$ for
    * value $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$
    * reaction $\Delta;\ \cdot \vdash R_2 : I \cup \{o_1, o_2\} \longrightarrow \tau_2$
  - $\mathsf{unit}(o_1 := v_1 \mid\mid o_2 := v_2) \sim_{\mathsf{subst}} \mathsf{unit}(o_1 := v_1 \mid\mid o_2 := v_2)$ for values $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$ and $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$
- Drop: Let $\sim$ be the reaction bisimulation obtained from the premise. Our desired bisimulation is the lifting of the relation $\sim_{\mathsf{drop}}$ defined by
  - $\big(o_1 := \eta_1 \mid\mid o_2 := x_1 \leftarrow \mathsf{read}\ o_1;\ R_2\big) \sim_{\mathsf{drop}} \big(o_1 := \eta_1 \mid\mid o_2 := \eta_2\big)$ for
    * measure η_1 on reactions $\Delta;\ \cdot \vdash R_1 : I \cup \{o_1, o_2\} \longrightarrow \tau_1$
    * reaction $\Delta;\ \cdot \vdash R_1 : I \cup \{o_1, o_2\} \longrightarrow \tau_1$ such that
      i) R_1 either evaluates to the same distribution as η_1, or
      ii) there exists a measure $\overline{\eta_1}$ on reactions $\Delta;\ \cdot \vdash R_1 : I \cup \{o_1, o_2\} \longrightarrow \tau_1$ such that R_1 evaluates to the same distribution as $\eta_1 + \overline{\eta_1}$
    * distribution η_2 on reactions $\Delta;\ \cdot \vdash R_2 : I \cup \{o_1, o_2\} \longrightarrow \tau_2$
    * reaction $\Delta;\ \cdot \vdash R_2 : I \cup \{o_1, o_2\} \longrightarrow \tau_2$ evaluating to the same distribution as η_2

    such that $\mathsf{unit}(x_1 \leftarrow R_1;\ R_2) \sim \mathsf{unit}(R_2)$
  - $(o_1 := v_1 \mid\mid o_2 := R_2) \sim_{\mathsf{drop}} (o_1 := v_1 \mid\mid o_2 := R_2)$ for
    * value $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$
    * reaction $\Delta;\ \cdot \vdash R_2 : I \cup \{o_1, o_2\} \longrightarrow \tau_2$
  - $(o_1 := v_1 \mid\mid o_2 := v_2) \sim_{\mathsf{drop}} (o_1 := v_1 \mid\mid o_2 := v_2)$ for values $v_1 \in \{0,1\}^{[\![\tau_1]\!]}$ and $v_2 \in \{0,1\}^{[\![\tau_2]\!]}$

□

$$\frac{o \notin I \qquad b \in I \qquad b : \mathsf{bool}, o : \tau \in \Delta \qquad \Delta;\ \cdot \vdash S_1 : I \cup \{o\} \longrightarrow \tau \qquad \Delta;\ \cdot \vdash S_2 : I \cup \{o\} \longrightarrow \tau}{\begin{array}{c}\Delta \vdash \big(\mathsf{new}\ l : \tau\ \mathsf{in}\ o := x : \mathsf{bool} \leftarrow \mathsf{read}\ b;\ \mathsf{if}\ x\ \mathsf{then}\ \color{red}{\mathsf{read}\ l}\ \mathsf{else}\ S_2 \mid\mid \color{red}{l := x : \mathsf{bool} \leftarrow \mathsf{read}\ b;\ S_1}\big) = \\ \big(o := x : \mathsf{bool} \leftarrow \mathsf{read}\ b;\ \mathsf{if}\ x\ \mathsf{then}\ \color{red}{S_1}\ \mathsf{else}\ S_2\big) : I \longrightarrow \{o\}\end{array}}$$

$$\frac{o \notin I \qquad b \in I \qquad b : \mathsf{bool}, o : \tau \in \Delta \qquad \Delta;\ \cdot \vdash S_1 : I \cup \{o\} \longrightarrow \tau \qquad \Delta;\ \cdot \vdash S_2 : I \cup \{o\} \longrightarrow \tau}{\begin{array}{c}\Delta \vdash \big(\mathsf{new}\ r : \tau\ \mathsf{in}\ o := x : \mathsf{bool} \leftarrow \mathsf{read}\ b;\ \mathsf{if}\ x\ \mathsf{then}\ S_1\ \mathsf{else}\ \color{red}{\mathsf{read}\ r} \mid\mid \color{red}{r := x : \mathsf{bool} \leftarrow \mathsf{read}\ b;\ S_2}\big) = \\ \big(o := x : \mathsf{bool} \leftarrow \mathsf{read}\ b;\ \mathsf{if}\ x\ \mathsf{then}\ S_1\ \mathsf{else}\ \color{red}{S_2}\big) : I \longrightarrow \{o\}\end{array}}$$

Fig. 17. Alternative formulation of the rules fold-if-left(top) and fold-if-right(bottom).

Definition B.16 (Sound exact theory). Fix a signature Σ and an interpretation $\mathcal{I}$. An *exact* IPDL theory $\mathbb{T}$ is a triple $(\mathbb{T}_e, \mathbb{T}_r, \mathbb{T}_p)$ of expression-level, reaction-level, and protocol-level IPDL theories, respectively. The exact theory $\mathbb{T}$ is *sound* with respect to $\mathcal{I}$, written $\mathcal{I} \vDash \mathbb{T}$, if each of $\mathbb{T}_e$, $\mathbb{T}_r$, and $\mathbb{T}_p$ is sound with respect to $\mathcal{I}$.

B.2 Soundness for Approximate Fragment

Recall that the equational theory for the approximate fragment is composed of two judgments: $\vdash \{\Delta_\lambda \vdash P_\lambda \approx_\lambda Q_\lambda : I_\lambda \to O_\lambda\}$, for top-level asymptotic equality against polynomial time adversaries, and $\Delta \vdash P_1 \approx_\lambda^{(k,l)} P_2 : I \to O$, for soundness against a specific security parameter λ. We begin with the latter.

$$\begin{array}{llll} \text{Program Contexts} & C & ::= & \circ \mid C[\circ : \_ \cup \{i\} \to \_] \\ & & \mid & C[\theta^\star(\circ)] \mid C[\circ \mid\mid Q] \mid C[P \mid\mid \circ] \mid C[\mathsf{new}\ o : \tau\ \mathsf{in}\ \circ] \end{array}$$

$$\boxed{C : (\Delta_1 \vdash I_1 \to O_1) \to (\Delta_2 \vdash I_2 \to O_2)} \qquad \frac{}{\circ : (\Delta \vdash I \to O) \to (\Delta \vdash I \to O)}$$

$$\frac{i \notin I \cup O \qquad C : (\Delta \vdash I \cup \{i\} \to O) \to (\Delta_\star \vdash I_\star \to O_\star)}{C[\circ : \_ \cup \{i\} \to \_] : (\Delta \vdash I \to O) \to (\Delta_\star \vdash I_\star \to O_\star)}$$

$$\frac{\Delta \vdash Q : I \cup O_1 \to O_2 \qquad C : (\Delta \vdash I \to O_1 \cup O_2) \to (\Delta_\star \vdash I_\star \to O_\star)}{C[\circ \mid\mid Q] : (\Delta \vdash I \cup O_2 \to O_1) \to (\Delta_\star \vdash I_\star \to O_\star)}$$

$$\frac{\Delta \vdash P : I \cup O_2 \to O_1 \qquad C : (\Delta \vdash I \to O_1 \cup O_2) \to (\Delta_\star \vdash I_\star \to O_\star)}{C[P \mid\mid \circ] : (\Delta \vdash I \cup O_1 \to O_2) \to (\Delta_\star \vdash I_\star \to O_\star)}$$

$$\frac{C : (\Delta_1 \vdash I_1 \to O_1) \to (\Delta_2 \vdash I_2 \to O_2)}{C[\mathsf{new}\ o : \tau\ \mathsf{in}\ \circ] : (\Delta_1, o : \tau \vdash I \to O \cup \{o\}) \to (\Delta_2 \vdash I_2 \to O_2)}$$

$$\frac{\theta : \Delta_1 \to \Delta_2 \qquad C : (\Delta_1 \vdash \theta^\star(I) \to \theta^\star(O)) \to (\Delta_\star \vdash I_\star \to O_\star)}{C[\theta^\star(\circ)] : (\Delta_2 \vdash I \to O) \to (\Delta_\star \vdash I_\star \to O_\star)}$$

Fig. 18. Typing for IPDL contexts.

$$\begin{aligned} |\circ| &:= 0 \\ |C[\circ : \_ \cup \{i\} \to \_]| &:= |C| + 1 \\ |C[\theta^\star(\circ)]| &:= |C| \\ |C[\circ \mid\mid Q]| &:= |C| + |Q| \\ |C[P \mid\mid \circ]| &:= |C| + |P| \\ |C[\mathsf{new}\ o : \tau\ \mathsf{in}\ \circ]| &:= |C| \end{aligned}$$

C APPROXIMATE EQUIVALENCE

For approximate equivalence of IPDL protocols, we assume a finite set $\mathbb{T}_\approx$ of ambient *approximate axioms* of the form $\Delta \vdash P \approx Q : I \to O$. An approximate equivalence of two protocols $\Delta \vdash P : I \to O$ and $\Delta \vdash Q : I \to O$ with identical typing judgements takes the form of a judgement $\Delta \vdash P \approx^{(k,l)} Q : I \to O$, where k, l are natural numbers. When we need to make the theory $\mathbb{T}_\approx$ explicit, we write the judgement as $\mathbb{T}_\approx \Rightarrow \Delta \vdash P \approx^{(k,l)} Q : I \to O$.

Definition C.1 (Errors). An *error* ε is a function $\mathsf{N} \to \mathsf{N} \to [0, 1]$ that is monotonically increasing in both arguments. Given a signature Σ, an interpretation $\mathcal{I}$, and two protocols $\Delta \vdash P : I \to O$ and $\Delta \vdash Q : I \to O$ with identical typing judgements, we write $\mathcal{I} \models \Delta \vdash P \approx_\varepsilon Q : I \to O$ to mean that for any two natural numbers $p, q \in \mathsf{N}$, any program context $C : (\Delta \vdash I \to O) \to (\Delta' \vdash I' \to O')$ bounded by p, and any distinguisher $\mathcal{A}$ for $\Delta' \vdash I' \to O'$ bounded by q, we have

$$\left|\Pr[\mathcal{A}(C[P])_\mathcal{I}] - \Pr[\mathcal{A}(C[Q])_\mathcal{I}]\right| \le \varepsilon(p, q)$$

Informally, soundness for approximate equivalence means that if P approximately rewrites to Q, then we can derive an error for this judgement that is a reasonable combination of the errors for each axiom.

Lemma C.2 (Soundness of approximate equivalence of protocols). *Fix a signature Σ, an interpretation $\mathcal{I}$, an ambient approximate theory $\Delta_1 \vdash P_1 \approx Q_1 : I_1 \to O_1, \ldots, \Delta_n \vdash P_n \approx Q_n : I_n \to O_n$, errors $\varepsilon_1, \ldots, \varepsilon_n$ such that $\mathcal{I} \models \Delta_i \vdash P_i \approx_{\varepsilon_i} Q_i : I_i \to O_i$, and two protocols $\Delta \vdash P : I \to O$ and $\Delta \vdash Q : I \to O$ with identical typing judgments. Then $\Delta \vdash P \approx^{(k,l)} Q : I \to O$ implies $\mathcal{I} \models \Delta \vdash P =_{\varepsilon_l^k} Q : I \to O$, where*

$$\varepsilon_l^k(p, q) \coloneqq k \max\left(\varepsilon_1(p + l, q), \ldots, \varepsilon_n(p + l, q)\right)$$

Proof. We proceed by induction on the derivation.

- strict By the soundness of exact equivalence, the hypothesis implies that P and Q are perfectly computationally indistinguishable. Thus, we have

$$\left|\Pr[\mathcal{A}(C[P_1])_\mathcal{I}] - \Pr[\mathcal{A}(C[P_2])_\mathcal{I}]\right| \le 0$$

  as desired.
- sym We have

$$\begin{aligned}\left|\Pr[\mathcal{A}(C[P_2])_\mathcal{I}] - \Pr[\mathcal{A}(C[P_1])_\mathcal{I}]\right| &= \left|\Pr[\mathcal{A}(C[P_1])_\mathcal{I}] - \Pr[\mathcal{A}(C[P_2])_\mathcal{I}]\right| \\ &\le k \max\left(\varepsilon_1(p + l, q), \ldots, \varepsilon_n(p + l, q)\right)\end{aligned}$$

  as desired.
- trans We have

$$\begin{aligned}&\left|\Pr[\mathcal{A}(C[P_1])_\mathcal{I}] - \Pr[\mathcal{A}(C[P_3])_\mathcal{I}]\right| \\ &\le \left|\Pr[\mathcal{A}(C[P_1])_\mathcal{I}] - \Pr[\mathcal{A}(C[P_2])_\mathcal{I}]\right| + \left|\Pr[\mathcal{A}(C[P_2])_\mathcal{I}] - \Pr[\mathcal{A}(C[P_3])_\mathcal{I}]\right| \\ &\le k_1 \max\left(\varepsilon_1(p + l_1, q), \ldots, \varepsilon_n(p + l_1, q)\right) + k_2 \max\left(\varepsilon_1(p + l_2, q), \ldots, \varepsilon_n(p + l_2, q)\right) \\ &\le k_1 \max\left(\varepsilon_1(p + \max(l_1, l_2), q), \ldots, \varepsilon_n(p + \max(l_1, l_2), q)\right) \\ &\qquad + k_2 \max\left(\varepsilon_1(p + \max(l_1, l_2), q), \ldots, \varepsilon_n(p + \max(l_1, l_2), q)\right) \\ &= (k_1 + k_2) \max\left(\varepsilon_1(p + \max(l_1, l_2), q), \ldots, \varepsilon_n(p + \max(l_1, l_2), q)\right)\end{aligned}$$

  as desired.

- AXIOM By assumption, we have
$$\left|\Pr[\mathcal{A}(C[P_i])_{\mathcal{I}}] - \Pr[\mathcal{A}(C[Q_i])_{\mathcal{I}}]\right| \leq \varepsilon_i(p,q) \leq \max\left(\varepsilon_1(p,q),\ldots,\varepsilon_n(p,q)\right)$$
as desired.
- SUBSUME We have
$$\begin{aligned}\left|\Pr[\mathcal{A}(C[P])_{\mathcal{I}}] - \Pr[\mathcal{A}(C[Q])_{\mathcal{I}}]\right| & \\ &\leq k_1 \max\left(\varepsilon_1(p+l_1,q),\ldots,\varepsilon_n(p+l_1,q)\right) \\ &\leq k_2 \max\left(\varepsilon_1(p+l_2,q),\ldots,\varepsilon_n(p+l_2,q)\right)\end{aligned}$$
as desired.
- INPUT-UNUSED Plugging the protocols $P : I \cup \{i\} \to O$ and $Q : I \cup \{i\} \to O$ into the context C is the same as plugging the protocols $P : I \to O$ and $Q : I \to O$ into the context $C[\circ : \_ \cup \{i\} \to \_]$, so we have
$$\begin{aligned}&\left|\Pr[\mathcal{A}(C[P : \_ \cup \{i\} \to \_])_{\mathcal{I}}] - \Pr[\mathcal{A}(C[Q : \_ \cup \{i\} \to \_])_{\mathcal{I}}]\right| \\ &\leq k \max\left(\varepsilon_1(p+1+l,q),\ldots,\varepsilon_n(p+1+l,q)\right)\end{aligned}$$
as desired.
- EMBED Plugging the protocols $\theta^\star(P)$ and $\theta^\star(Q)$ into the context C is the same as plugging the protocols P and Q into the context $C[\theta^\star(\circ)]$, so we have
$$\left|\Pr[\mathcal{A}(C[\theta^\star(P)])_{\mathcal{I}}] - \Pr[\mathcal{A}(C[\theta^\star(Q)])_{\mathcal{I}}]\right| \leq k \max\left(\varepsilon_1(p+l,q),\ldots,\varepsilon_n(p+l,q)\right)$$
as desired.
- CONG-COMP-LEFT Plugging the protocols $P \mid\mid Q$ and $P' \mid\mid Q$ into the context C is the same as plugging the protocols P and P' into the context $C[\circ \mid\mid Q]$, so we have
$$\left|\Pr[\mathcal{A}(C[P \mid\mid Q])_{\mathcal{I}}] - \Pr[\mathcal{A}(C[P' \mid\mid Q])_{\mathcal{I}}]\right| \leq k \max\left(\varepsilon_1(p+|Q|+l,q),\ldots,\varepsilon_n(p+|Q|+l,q)\right)$$
as desired.
- CONG-NEW Plugging the protocols new $o : A$ in P and new $o : A$ in P' into the context C is the same as plugging the protocols P and P' into the context $C[\mathsf{new}\ o : \tau\ \mathsf{in}\ \circ]$, so we have
$$\left|\Pr[\mathcal{A}(C[\mathsf{new}\ o : A\ \mathsf{in}\ P])_{\mathcal{I}}] - \Pr[\mathcal{A}(C[\mathsf{new}\ o : A\ \mathsf{in}\ P'])_{\mathcal{I}}]\right| \leq k \max\left(\varepsilon_1(p+l,q),\ldots,\varepsilon_n(p+l,q)\right)$$
as desired.

□

For *asymptotic* equivalence of IPDL protocols, lift the notion of approximate equivalence to protocol *families*, indexed by the security parameter $\lambda \in \mathsf{N}$. We assume a finite set $\mathbb{T}_{\mathsf{poly}}$ of ambient *axiom families* of the form $\{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda \in \mathsf{N}}$. The asymptotic equivalence of two *protocol families* $\{\Delta_\lambda \vdash P_\lambda : I_\lambda \to O_\lambda\}_{\lambda \in \mathsf{N}}$ and $\{\Delta_\lambda \vdash Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda \in \mathsf{N}}$ with identical typing judgements takes the form of the judgement $\mathbb{T}_{\mathsf{poly}} \Rightarrow \{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda \in \mathsf{N}}$. When we need to make the underlying exact theory $\mathbb{T}$ explicit, we write the judgement as $\mathbb{T};\ \mathbb{T}_{\mathsf{poly}} \Rightarrow \{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda \in \mathsf{N}}$.

Specifically, for any fixed λ, we obtain an approximate theory by selecting from each axiom family the particular axiom corresponding to this value of λ. Similarly, from each protocol family we select the protocol corresponding to the given λ, which gives us two concrete protocols. For the asymptotic judgement to hold, these λ-specific protocols must be approximately equivalent with respect to the λ-specific approximate theory we constructed earlier. We recall that an approximate judgement is tagged by a pair of parameters k and l. Ranging over all values of λ thus gives us two functions $k(\lambda)$ and $l(\lambda)$, and we require that these be bounded by a polynomial. Informally,

$$\boxed{\{\Delta^1_\lambda \vdash P^1_\lambda \approx Q^1_\lambda : I^1_\lambda \to O^1_\lambda\}_{\lambda\in\mathsf{N}}\}, \ldots, \{\Delta^n_\lambda \vdash P^n_\lambda \approx Q^n_\lambda : I^n_\lambda \to O^n_\lambda\}_{\lambda\in\mathsf{N}} \Rightarrow \{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathsf{N}}\}}$$

$$\frac{\begin{array}{c}\forall\lambda, \Delta^1_\lambda \vdash P^1_\lambda \approx Q^1_\lambda : I^1_\lambda \to O^1_\lambda, \ldots, \Delta^n_\lambda \vdash P^n_\lambda \approx Q^n_\lambda : I^n_\lambda \to O^n_\lambda \Rightarrow \Delta_\lambda \vdash P_\lambda \approx^{(k_\lambda, l_\lambda)} Q_\lambda : I_\lambda \to O_\lambda \\ k_\lambda = \mathrm{O}(\mathrm{poly}(\lambda)) \qquad l_\lambda = \mathrm{O}(\mathrm{poly}(\lambda))\end{array}}{\{\Delta^1_\lambda \vdash P^1_\lambda \approx Q^1_\lambda : I^1_\lambda \to O^1_\lambda\}_{\lambda\in\mathsf{N}}, \ldots, \{\Delta^n_\lambda \vdash P^n_\lambda \approx Q^n_\lambda : I^n_\lambda \to O^n_\lambda\}_{\lambda\in\mathsf{N}} \Rightarrow \{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathsf{N}}}$$

Fig. 19. Asymptotic equivalence for IPDL protocol families.

we can summarize the asymptotic judgement as saying that the protocol families are pointwise approximately equivalent, and the size of the derivation does not grow too fast in λ.

Definition C.3 (Computational Indistinguishability). Fix a signature Σ, a family of interpretations $\{\mathcal{I}_\lambda\}_{\lambda\in\mathsf{N}}$, and two families of protocols $\{\Delta_\lambda \vdash P_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathsf{N}}$ and $\{\Delta_\lambda \vdash Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathsf{N}}$ with identical typing judgments. Then we say that $\{P_\lambda\}$ and $\{Q_\lambda\}$ are *indistinguishable* under $\{\mathcal{I}_\lambda\}_{\lambda\in\mathsf{N}}$, written $\{\mathcal{I}_\lambda\}_{\lambda\in\mathsf{N}} \vDash \{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathsf{N}}$, if for any two polynomials $p(\lambda)$ and $q(\lambda)$ there exists a negligible function $\varepsilon(\lambda)$ such that for any $\lambda \in \mathsf{N}$, any program context $C : (\Delta_\lambda \vdash I_\lambda \to O_\lambda) \to (\Delta'_\lambda \vdash I'_\lambda \to O'_\lambda)$ bounded by $p(\lambda)$, and any distinguisher $\mathcal{A}$ for $\Delta'_\lambda \vdash I'_\lambda \to O'_\lambda$ bounded by $q(\lambda)$, we have

$$\left|\Pr[\mathcal{A}(C[P_\lambda])_{\mathcal{I}_\lambda}] - \Pr[\mathcal{A}(C[Q_\lambda])_{\mathcal{I}_\lambda}]\right| \leq \varepsilon(\lambda)$$

Definition C.4 (Sound asymptotic theory). Fix a signature Σ and a family of interpretations $\{\mathcal{I}_\lambda\}_{\lambda\in\mathsf{N}}$. An asymptotic theory $\{\Delta^1_\lambda \vdash P^1_\lambda \approx Q^1_\lambda : I^1_\lambda \to O^1_\lambda\}_{\lambda\in\mathsf{N}}\}, \ldots, \{\Delta^n_\lambda \vdash P^n_\lambda \approx Q^n_\lambda : I^n_\lambda \to O^n_\lambda\}_{\lambda\in\mathsf{N}}$ is *sound* under $\{\mathcal{I}_\lambda\}_{\lambda\in\mathsf{N}}$, written $\{\mathcal{I}_\lambda\}_{\lambda\in\mathsf{N}} \vDash \{\Delta^1_\lambda \vdash P^1_\lambda \approx Q^1_\lambda : I^1_\lambda \to O^1_\lambda\}_{\lambda\in\mathsf{N}}\}, \ldots, \{\Delta^n_\lambda \vdash P^n_\lambda \approx Q^n_\lambda : I^n_\lambda \to O^n_\lambda\}_{\lambda\in\mathsf{N}}$, if each axiom family is computationally indistinguishable under $\{\mathcal{I}_\lambda\}_{\lambda\in\mathsf{N}}$, that is $\{\mathcal{I}_\lambda\}_{\lambda\in\mathsf{N}} \vDash \{\Delta^i_\lambda \vdash P^i_\lambda \approx Q^i_\lambda : I^i_\lambda \to O^i_\lambda\}_{\lambda\in\mathsf{N}}$.

Theorem C.5 (Soundness of asymptotic equivalence of protocols). *Fix a signature Σ, a PPT family of interpretations $\{\mathcal{I}_\lambda\}_{\lambda\in\mathsf{N}}$, an exact theory $\mathbb{T}$ such that $\mathcal{I}_\lambda \vDash \mathbb{T}$ for each $\lambda \in \mathsf{N}$, an asymptotic theory $\mathbb{T}_{\mathsf{poly}}$ such that $\{\mathcal{I}_\lambda\}_{\lambda\in\mathsf{N}} \vDash \mathbb{T}_{\mathsf{poly}}$, and two protocol families $\{\Delta_\lambda \vdash P_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathsf{N}}$ and $\{\Delta_\lambda \vdash Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathsf{N}}$ with identical typing judgments such that $|I_\lambda|$ and $|O_\lambda|$ are polynomial in λ. Then $\mathbb{T}$; $\mathbb{T}_{\mathsf{poly}} \Rightarrow \{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathsf{N}}$ implies $\{\mathcal{I}_\lambda\}_{\lambda\in\mathsf{N}} \vDash \{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathsf{N}}$.*

Proof. Let $\{\Delta^1_\lambda \vdash P^1_\lambda \approx Q^1_\lambda : I^1_\lambda \to O^1_\lambda\}_{\lambda\in\mathsf{N}}, \ldots, \{\Delta^n_\lambda \vdash P^n_\lambda \approx Q^n_\lambda : I^n_\lambda \to O^n_\lambda\}_{\lambda\in\mathsf{N}}$ be the axioms comprising the asymptotic theory $\mathbb{T}_{\mathsf{poly}}$. Since $\mathbb{T}_{\mathsf{poly}}$ is sound with respect to the family $\{\mathcal{I}_\lambda\}_{\lambda\in\mathsf{N}}$ of interpretations, each axiom family $\{\Delta^i_\lambda \vdash P^i_\lambda \approx Q^i_\lambda : I^i_\lambda \to O^i_\lambda\}_{\lambda\in\mathsf{N}}$ is computationally indistinguishable with respect to $\{\mathcal{I}_\lambda\}_{\lambda\in\mathsf{N}}$.

The top-level asymptotic equivalence judgement gives us two polynomials $k(\lambda)$ and $l(\lambda)$ that bind the derivation size. Fix polynomials $p(\lambda)$ and $q(\lambda)$. By assumption on the computational indistinguishability of each axiom family applied to the polynomials $p(\lambda) + l(\lambda)$ and $q(\lambda)$, there exist negligible functions $\epsilon^1(\lambda), \ldots, \epsilon^n(\lambda)$, one for each axiom family, such that

$$\left|\Pr[\mathcal{A}(C[P^i_\lambda])_{\mathcal{I}_\lambda}] - \Pr[\mathcal{A}(C[Q^i_\lambda])_{\mathcal{I}_\lambda}]\right| \leq \epsilon^i(\lambda)$$

Define

$$\epsilon(\lambda) \coloneqq k(\lambda) \max\left(\epsilon_1(\lambda), \ldots, \epsilon_n(\lambda)\right)$$

for any $\lambda \in \mathsf{N}$, program context $C : (\Delta^i_\lambda \vdash I^i_\lambda \to O^i_\lambda) \to (\Delta' \vdash I' \to O')$ bounded by $p(\lambda) + l(\lambda)$, and distinguisher $\mathcal{A}$ for $\Delta' \vdash I' \to O'$ bounded by $q(\lambda)$. Clearly $\epsilon(\lambda)$ is negligible. Define an error

$$\varepsilon^i(p, q) \coloneqq \sup\left\{\left|\Pr[\mathcal{A}(C[P^i_\lambda])_{\mathcal{I}_\lambda}] - \Pr[\mathcal{A}(C[Q^i_\lambda])_{\mathcal{I}_\lambda}]\right|\right\}$$

to be the supremum of the distinguishing advantages taken over all program contexts $C : (\Delta^i_\lambda \vdash I^i_\lambda \to O^i_\lambda) \to (\Delta' \vdash I' \to O')$ bounded by p, and all distinguishers $\mathcal{A}$ for $\Delta' \vdash I' \to O'$ bounded by q. Then by construction, $\mathcal{I}_\lambda \vDash \Delta^i_\lambda \vdash P^i_\lambda \approx_{\varepsilon^i} Q^i_\lambda : I^i_\lambda \to O^i_\lambda$.

For a fixed λ, program context $C : (\Delta_\lambda \vdash I_\lambda \to O_\lambda) \to (\Delta'_\lambda \vdash I'_\lambda \to O'_\lambda)$ bounded by $p(\lambda)$, and distinguisher $\mathcal{A}$ for $\Delta'_\lambda \vdash I'_\lambda \to O'_\lambda$ bounded by $q(\lambda)$, we therefore have

$$\begin{aligned}\left|\Pr[\mathcal{A}(C[P_\lambda])_{\mathcal{I}_\lambda}] - \Pr[\mathcal{A}(C[Q_\lambda])_{\mathcal{I}_\lambda}]\right| &\leq k(\lambda) \max\left(\varepsilon^1(p(\lambda)+l(\lambda), q(\lambda)), \ldots, \varepsilon^n(p(\lambda)+l(\lambda), q(\lambda))\right) \\ &\leq k(\lambda) \max\left(\epsilon_1(\lambda), \ldots, \epsilon_n(\lambda)\right) \\ &= \epsilon(\lambda)\end{aligned}$$

where the first inequality follows from the soundness theorem for approximate equivalence, and the second from the definition of supremum.

□

REFERENCES

Martin Abadi and Phillip Rogaway. 2002. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of cryptology* 15, 2 (2002), 103–127.

Michael Backes, Ankit Malik, and Dominique Unruh. 2012. Computational Soundness without Protocol Restrictions. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (Raleigh, North Carolina, USA) *(CCS '12)*. Association for Computing Machinery, New York, NY, USA, 699–711. https://doi.org/10.1145/2382196.2382270

Michael Backes, Birgit Pfitzmann, and Michael Waidner. 2007. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation* 205, 12 (2007), 1685–1720. https://doi.org/10.1016/j.ic.2007.05.002

David Baelde, Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos, and Solène Moreau. 2021. An Interactive Prover for Protocol Verification in the Computational Model. In *SP 2021 - 42nd IEEE Symposium on Security and Privacy*. San Fransisco / Virtual, United States. https://hal.archives-ouvertes.fr/hal-03172119

Gergei Bana and Hubert Comon-Lundh. 2014. A Computationally Complete Symbolic Attacker for Equivalence Properties. *Proceedings of the ACM Conference on Computer and Communications Security* (11 2014). https://doi.org/10.1145/2660267.2660276

M. Barbosa, G. Barthe, K. Bhargavan, B. Blanchet, C. Cremers, K. Liao, and B. Parno. 2021a. SoK: Computer-Aided Cryptography. In *2021 2021 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 777–795. https://doi.org/10.1109/SP40001.2021.00008

Manuel Barbosa, Gilles Barthe, Benjamin Grégoire, Adrien Koutsos, and Pierre-Yves Strub. 2021b. Mechanized Proofs of Adversarial Complexity and Application to Universal Composability. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2541–2563. https://doi.org/10.1145/3460120.3484548

G. Barthe, B. Grégoire, S. Heraud, and Santiago Zanella Béguelin. 2011. Computer-Aided Security Proofs for the Working Cryptographer. In *CRYPTO*.

Gilles Barthe, Benjamin Grégoire, and Benedikt Schmidt. 2015. Automated proofs of pairing-based cryptography. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1156–1168.

Donald Beaver. 1995. Precomputing oblivious transfer. In *Annual International Cryptology Conference*. Springer, 97–109.

Karthikeyan Bhargavan, Abhishek Bichhawat, Quoc Huy Do, Pedram Hosseyni, Ralf Küsters, Guido Schmitz, and Tim Würtele. 2021. DY* : A Modular Symbolic Verification Framework for Executable Cryptographic Protocol Code. In *EuroS&P 2021 - 6th IEEE European Symposium on Security and Privacy*. Virtual, Austria. https://hal.inria.fr/hal-03178425

Bruno Blanchet. 2006. A Computationally Sound Mechanized Prover for Security Protocols. 140–154. https://doi.org/10.1109/SP.2006.1

Bruno Blanchet. 2013. Automatic verification of security protocols in the symbolic model: The verifier proverif. In *Foundations of security analysis and design VII*. Springer, 54–87.

Manuel Blum. 1983. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News* 15, 1 (1983), 23–27.

David Butler, David Aspinall, and Adrià Gascón. 2020. Formalising Oblivious Transfer in the Semi-Honest and Malicious Model in CryptHOL. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs* (New Orleans, LA, USA) *(CPP 2020)*. Association for Computing Machinery, New York, NY, USA, 229–243. https://doi.org/10.1145/3372885.3373815

Ran Canetti. 2000. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Cryptology ePrint Archive, Report 2000/067. https://ia.cr/2000/067.

Ran Canetti, Alley Stoughton, and Mayank Varia. 2019. EasyUC: Using EasyCrypt to Mechanize Proofs of Universally Composable Security. In *32nd IEEE Computer Security Foundations Symposium*. https://eprint.iacr.org/2019/582.

Miguel Castro, Barbara Liskov, et al. 1999. Practical byzantine fault tolerance. In *OsDI*, Vol. 99. 173–186.

Veronique Cortier and Bogdan Warinschi. 2011. A Composable Computational Soundness Notion. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (Chicago, Illinois, USA) *(CCS '11)*. Association for Computing Machinery, New York, NY, USA, 63–74. https://doi.org/10.1145/2046707.2046717

Cas J. F. Cremers. 2008. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In *Computer Aided Verification*, Aarti Gupta and Sharad Malik (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 414–418.

Karim M. El Defrawy and Vitor Pereira. 2019. A High-Assurance Evaluator for Machine-Checked Secure Multiparty Computation. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (2019).

Danny Dolev and Andrew Yao. 1983. On the security of public key protocols. *IEEE Transactions on information theory* 29, 2 (1983), 198–208.

Denis Firsov and Dominique Unruh. 2022. Reflection, Rewinding, and Coin-Toss in EasyCrypt. In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs* (Philadelphia, PA, USA) *(CPP 2022)*. Association for Computing Machinery, New York, NY, USA, 166–179. https://doi.org/10.1145/3497775.3503693

Joshua Gancher, Kristina Sojakova, Xiong Fan, Elaine Shi, and Greg Morrisett. 2022. A Core Calculus for Equational Proofs of Distributed Cryptographic Protocols: Supplemental Material. https://github.com/ipdl/ipdl.

Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 218–229.

Andrew K. Hirsch and Deepak Garg. 2022. Pirouette: Higher-Order Typed Functional Choreographies. *Proc. ACM Program. Lang.* 6, POPL, Article 23 (jan 2022), 27 pages. https://doi.org/10.1145/3498684

Kevin Liao, Matthew A. Hammer, and Andrew Miller. 2019. ILC: A Calculus for Composable, Computational Cryptography. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. Association for Computing Machinery, New York, NY, USA, 640–654. https://doi.org/10.1145/3314221.3314607

Yehuda Lindell. 2020. Secure Multiparty Computation. *Commun. ACM* 64, 1 (dec 2020), 86–96. https://doi.org/10.1145/3387108

Andreas Lochbihler and S. Reza Sefidgar. 2018. A tutorial introduction to CryptHOL. Cryptology ePrint Archive, Report 2018/941. https://ia.cr/2018/941.

Andreas Lochbihler, S. Reza Sefidgar, David Basin, and Ueli Maurer. 2019. Formalizing Constructive Cryptography using CryptHOL. In *32nd IEEE Computer Security Foundations Symposium*. http://www.andreas-lochbihler.de/pub/lochbihler2019csf.pdf.

Gavin Lowe. 1996. Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 147–166.

Assia Mahboubi and Enrico Tassi. 2021. *Mathematical Components*. Zenodo. https://doi.org/10.5281/zenodo.4457887

Ueli Maurer. 2012. Constructive Cryptography – A New Paradigm for Security Definitions and Proofs. In *Theory of Security and Applications*, Sebastian Mödersheim and Catuscia Palamidessi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 33–56.

Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Computer Aided Verification*, Natasha Sharygina and Helmut Veith (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 696–701.

Robin Milner, Joachim Parrow, and David Walker. 1992. A calculus of mobile processes, I. *Information and Computation* 100, 1 (1992), 1–40. https://doi.org/10.1016/0890-5401(92)90008-4

Moni Naor and Benny Pinkas. 1999. Oblivious transfer and polynomial evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. 245–254.

Adam Petcher and Greg Morrisett. 2015. The Foundational Cryptography Framework. In *Principles of Security and Trust*, Riccardo Focardi and Andrew Myers (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 53–72.

S. Schneider. 1996. Security properties and CSP. In *Proceedings 1996 IEEE Symposium on Security and Privacy*. 174–187. https://doi.org/10.1109/SECPRI.1996.502680

Received 2022-07-07; accepted 2022-11-07