# Concrete Security Bounds for Simulation-Based Proofs of Multi-Party Computation Protocols

KRISTINA SOJAKOVA, Vrije Universiteit Amsterdam, Netherlands
MIHAI CODESCU, Research Group of the NLNet Project IPDL, Romania
JOSHUA GANCHER, Northeastern University Boston, United States of America

The *concrete security* paradigm aims to give precise bounds on the probability that an adversary can subvert a cryptographic mechanism. This is in contrast to asymptotic security, where the probability of subversion may be *eventually* small, but large enough in practice to be insecure. Fully satisfactory concrete security bounds for Multi-Party Computation (MPC) protocols are difficult to attain, as they require reasoning about the *running time* of cryptographic adversaries and reductions.

In this paper we close this gap by introducing a new foundational approach that allows us to automatically compute concrete security bounds for MPC protocols. We take inspiration from the meta-theory of IPDL, a prior approach for formally verified distributed cryptography, to support reasoning about the runtime of protocols and adversarial advantage. For practical proof developments, we implement our approach in Maude, an extensible logic for equational rewriting.

We carry out four case studies of concrete security for simulation-based proofs. Most notably, we deliver the first formal verification of the GMW MPC protocol over $N$ parties. To our knowledge, this is the first time that formally verified concrete security bounds are computed for a proof of an MPC protocol in the style of Universal Composability. Our tool provides a layer of abstraction that allows the user to write proofs at a high level, which drastically simplifies the proof size. For comparison, a case study that in prior works required 2019 LoC only takes 567 LoC, thus reducing proof size by 72%.

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

Additional Key Words and Phrases: multi-party computation, equational reasoning, formal verification

**ACM Reference Format:**
Kristina Sojakova, Mihai Codescu, and Joshua Gancher. 2025. Concrete Security Bounds for Simulation-Based Proofs of Multi-Party Computation Protocols. 1, 1 (July 2025), 36 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Advanced distributed cryptographic protocols such as Multi-Party Computation (MPC) have the potential to enable new, privacy-preserving modes of computing. However, their inherent complexity introduces new risks, including the risk that the protocol itself (or protocol optimizations employed by implementations) is insecure. The commonly-used security definition for MPC is simulation-based security in the style of Universal Composability, or UC [Canetti 2000]. UC provides strong security guarantees that are robust under an embedding of the protocol into a larger distributed

Authors' addresses: Kristina Sojakova, k.sojakova@vu.nl, Vrije Universiteit Amsterdam, Netherlands; Mihai Codescu, mscodescu@gmail.com, Research Group of the NLNet Project IPDL, Romania; Joshua Gancher, j.gancher@northeastern.edu, Northeastern University Boston, United States of America.


XXXX-XXXX/2025/7-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

system. However, UC-style proofs for MPC protocols are generally very difficult, as they require complex bisimulation-based security arguments in conjunction with low-level runtime analysis.

While a number of prior approaches for verifying MPC protocols have been proposed [Almeida et al. 2017; Defrawy and Pereira 2019; Gancher et al. 2023b; Haagh et al. 2018; Stoughton and Varia 2017], most of these approaches deliver standalone security definitions not compatible with UC [Almeida et al. 2017; Defrawy and Pereira 2019; Haagh et al. 2018; Stoughton and Varia 2017], which harms composability. In contrast, the IPDL system [Gancher et al. 2023b] can deliver UC-style security results for MPC protocols via a convenient *equational* style of reasoning. Prior work has shown that IPDL can reason about realistic protocols, including a two-party variant of the GMW [Goldreich et al. 1987] MPC protocol, along with various Oblivious Transfer [Beaver 1995] protocols.

However, a number of security-critical caveats remain. None of the prior approaches for verifying MPC adequately reason about the *runtime* of adversaries and simulators constructed during the proof. Reasoning about runtime is essential for security, since nearly all cryptographic proofs contain *reductions* of the form "the probability that $A$ breaks protocol $P$ is bounded by the probability that the reduced adversary $R(A)$ breaks indistinguishability assumption $Q$". Here $R(A)$ is an adversary whose interaction with $Q$ mimics the interaction of the original adversary $A$ with $P$. If $R(A)$'s runtime is not adequately bounded, then the probability that $R(A)$ breaks the assumption $Q$ may be 1, rendering the security result essentially meaningless. This problem is even more pronounced for large protocols such as MPC, since the constructed simulators also become increasingly complex.

In this work, we address this issue for MPC protocols. Our method supports *concrete security* [Bellare et al. 1997], which gives precise bounds on the probability $\epsilon(t)$ that an attacker running in time $t$ violates the security guarantees of the system. Our novel strategy to obtain these bounds is to bound the size of the IPDL program context, and analyze the runtime of a Turing Machine that interprets IPDL programs. This strategy allows us to obtain practical bounds for MPC: indeed, *to our knowledge, there is no other formally verified proof of an MPC protocol that reasons about concrete security and runtime of adversaries/simulators.*

To enable fast and scalable formal proofs, we implemented our proof system in the equational rewrite tool Maude [Clavel et al. 2007] as an extension of SpeX [Ţuţu 2022] and equipped it with a Domain-Specific Language (DSL) for concise proofs that automatically compile down to lower-level Maude code. Using our implementation, we carry out four different case studies. To highlight the scalability of our approach, we present a new, fully mechanized proof of *simulation-based security* for the *Multi-Party GMW* MPC protocol defined over an arbitrary Boolean circuit and for arbitrarily many parties. Crucially, our tool *automates* the computation of concrete security bounds, which would otherwise be infeasible to carry out for a protocol of this size.

### 1.1 Contributions

In this work, we develop:

- a proof system that automatically computes concrete security bounds for composable simulation-based proofs of MPC protocols;
- a formally verified proof of the *GMW* MPC protocol with $N$ parties, which to our knowledge marks the first time that formally verified concrete security bounds have been computed for a UC-style proof of an MPC protocol;
- an accompanying Maude implementation that automatically computes the aforementioned bounds and a DSL for writing proofs that hides most low-level details from the user. This dramatically simplifies the proof size: *e.g.*, excluding definitions, the proof of the *Coin Flip* case study in [Gancher et al. 2023b] takes 1905 LoC, whereas we deliver it in 256 LoC. Our

proofs also significantly outperform their Coq equivalents from [Gancher et al. 2023b] in terms of runtime: *e.g.*, 5 seconds vs. a few minutes for the *Coin Flip* case study.

*Limitations of our system.* Since we build upon the process calculus in [Gancher et al. 2023b], we are only able to support protocols expressible in IPDL. Specifically, IPDL only considers protocols with static communication topologies and static security. Furthermore, it does not consider protocols that exhibit threshold behavior such as consensus protocols. As IPDL is targeted towards UC security, it also does not consider proofs that use rewinding.

*Limitations of our proof effort.* Our proof of the GMW protocol assumes that party $N$ is semi-honest and party $N + 1$ is honest. While this is not fully general, all other cases are either trivial (all parties honest/all parties semi-honest) or can be essentially reduced to the aforementioned case. We note that the GMW protocol is not secure against a malicious adversary.

*Structure of paper.* In Section 3 we briefly review simulation-based security, present IPDL, and illustrate our DSL on a simple example. In Section 4, we give the syntax and semantics of our cost-aware proof system for simulation-based security, and present our soundness results. In Section 5, we outline our proof of the *N-Party GMW* protocol and briefly describe the other case studies. We conclude by indicating some directions for future work.

## 2 RELATED WORK

This paper is part of a long line of formal verification efforts for MPC and similar protocols. Some works target domain-specific, standalone security notions for protocols [Almeida et al. 2017; Defrawy and Pereira 2019; Haagh et al. 2018; Stoughton and Varia 2017], while others [Barbosa et al. 2021; Canetti et al. 2019; Gancher et al. 2023b; Lochbihler et al. 2019] aim to construct general frameworks for *simulation-based security* in the style of UC. While runtime analysis is in a sense required for sound cryptographic reasoning, almost all of the above works (with the exception of [Barbosa et al. 2021] and [Gancher et al. 2023b]) declare reasoning about runtime out of scope, and instead defer to the reader to ensure that all relevant cryptographic reductions have a reasonable running time.

The line of work proposed in [Gancher et al. 2023b] shows that an equational proof strategy is useful for proving MPC and related protocols secure. However, in lieu of reasoning about runtime, [Gancher et al. 2023b] relies on symbolic bounds, which provide some measure of complexity for a syntactic simulation context. However, [Gancher et al. 2023b] does not give a low-level computational semantics to protocols, nor does it carry out any cryptographic reductions. It is therefore unclear what such a symbolic bound means for the runtime of adversaries and simulators. In this paper, we build upon the process calculus introduced in [Gancher et al. 2023b] with a new cost-aware proof system and semantics that reason explicitly about the runtime of cryptographic adversaries and simulators. In particular, we prove *concrete* security bounds of the form $\epsilon(t)$, where $t$ is the running time of the adversary.

The system by Barbosa et al. [Barbosa et al. 2021] extends EasyCrypt [Barthe et al. 2011] by a cost-aware Hoare logic and uses it to analyze a secure channel protocol. However, this runtime analysis only applies to sequential programs, while the process calculus of [Gancher et al. 2023b] natively handles concurrency. Thus, while [Barbosa et al. 2021] considers UC, their computational model assumes that the protocol follows a stack discipline, which is not a good fit for MPC.

Squirrel [Baelde et al. 2024] and CryptoVerif [Blanchet 2008] are two popular tools that deliver concrete security bounds for large classes of cryptographic protocols. However, the bulk of the proof for an MPC protocol consists of manipulations such as inlining the computation from one channel to another, or removing parts of the protocol that have become unused after simplification.

The process calculus of [Gancher et al. 2023b] was explicitly designed around such congruences, which do not have counterparts in Squirrel or CryptoVerif. This design choice enabled us to carry out the largest formally verified UC-style proof of an MPC protocol that we are aware of.

Overture [Skalka and Near 2024] is a recently-proposed system for proving security properties of MPC protocols via checking secrecy and integrity hyperproperties. While it offers automatic proofs, it does not prove UC-style properties, nor analyze the runtime of simulators. Owl [Gancher et al. 2023a] uses an information-flow type system to deliver modular proofs of protocols that use cryptographic mechanisms (rather than reason *about* them, which is required for MPC). Resource-aware session types (RAST) [Das et al. 2018] allow one to embed runtime guarantees into session-typed protocols. While RAST and our work both target concurrent process calculi, the protocols we consider do not naturally carry session types, and instead have a low-level computational interpretation in terms of Turing Machines. In particular, it is unclear how to embed MPC into RAST, or use session types in general to perform cryptographic security proofs. GAuV [Xie et al. 2024] uses graph transformations to automate proofs of semi-honest security for concrete instances of the BGW protocol (*i.e.*, for a fixed number of parties and a fixed circuit).

## 3 OVERVIEW: SIMULATION-BASED SECURITY AND IPDL

We now briefly describe simulation-based security, give an overview of the IPDL process calculus from [Gancher et al. 2023b], and illustrate our approach on a simple running example.

### 3.1 Simulation-Based Security

Simulation-based security relates the behavior of a protocol to that of an idealization, where cryptographic mechanisms are replaced by trusted resources secure by construction. In this sense, an idealization is a specification for the behavior that the real-world protocol aims to approximate with cryptographic methods. For example, a real-world protocol utilizes encryption to securely send a message from Alice to Bob over a public network. The idealization instead relies on a trusted third party that securely obtains the message from Alice and forwards it to Bob.

The simulation-based paradigm, as employed *e.g.* in UC and Constructive Cryptography [Maurer 2012] is very powerful and unifies various other security notions such as secrecy and integrity. In UC-style security, a protocol is an interactive system consisting of *parties*, *e.g.* Alice and Bob, and *functionalities*, *e.g.* a public network that forwards messages from Alice to Bob, or a key generation mechanism that randomly generates a secret key and securely delivers it to Alice and Bob.

Formal proofs in this setting amount to showing *observational equivalences* $P \approx Q$ between a "real" protocol $P$, and an "ideal" protocol $Q$. These proofs typically take the form of a sequence of exact ($=$) and approximate ($\approx$) equality steps:

$$P = P_1 \approx Q_1 = P_2 \approx Q_2 = \ldots = P_n \approx Q_n = Q$$

As observed in [Gancher et al. 2023b], a typical approximate equality step $Q_i \approx P_i$ consists of replacing the left-hand side of an *indistinguishability assumption* $G_i \approx H_i$ by its right-hand side in a common context $R_i$; *i.e.*, $P_i$ arises as $R_i[G_i]$ and $Q_i$ as $R_i[H_i]$. An example of such an assumption is IND-CPA, which states that encryptions of adversarially chosen messages are indistinguishable from encryptions of zeros (in the absence of a decryption oracle).

### 3.2 Background: IPDL

IPDL [Gancher et al. 2023b] is a process calculus for distributed cryptographic protocols (*e.g.*, MPC) that enables one to prove simulation-based security results similar to those analyzed in UC. IPDL *protocols* are composed of mutually interacting *reactions*, which are sequential monadic programs that probabilistically compute an *expression*. In the context of a protocol, a reaction operates on a

unique *channel* and may read from other channels, thereby utilizing computations coming from other reactions.

Aside from a formal process calculus, the original paper [Gancher et al. 2023b] defines two *equational* proof systems: an *exact* equational logic for proving perfect equivalences between protocols, and an *approximate* logic for proving computational indistinguishability results. The exact equational logic is proven sound in terms of *bisimulations*, while the approximate logic is proven sound in terms of *computational reductions*.

Following [Gancher et al. 2023b], we assume a user-defined *signature* that specifies the base types and the (probabilistic) functions we have at our disposal:

Definition 1 (Signature). *A signature* $\Sigma$ *consists of:*

- *type constants* $\mathsf{t}$,
- *function symbols* $\mathsf{f} : \sigma \to \tau$, *and*
- *distribution symbols* $\mathsf{d} : \sigma \to \tau$.

We summarize the syntax of IPDL in Figure 1. Data types and expressions are standard. Here 1 denotes the unit type and $\checkmark$ the canonical inhabitant of 1. The expression $\mathsf{app}_{\sigma\to\tau}\ \mathsf{f}\ e$ denotes the application of the function symbol $\mathsf{f} : \sigma \to \tau$ declared in the signature $\Sigma$ to an expression $e$. Similarly, the reaction $\mathsf{samp}_{\sigma\to\tau}\ \mathsf{d}\ e$ denotes the application of the distribution symbol $\mathsf{d} : \sigma \to \tau$ declared in the signature $\Sigma$ to an expression $e$.

The reaction $\mathsf{read}(c : \tau)$ denotes the read of a value of type $\tau$ from the channel $c$. We also have branching ($\mathsf{if}\ e\ \mathsf{then}\ R_1\ \mathsf{else}\ R_2$) and the standard monadic operations of return ($\mathsf{ret}\ e$) and bind ($x : \sigma \leftarrow R;\ S$). At the protocol level, we have the trivial protocol 0, the single-channel protocol $o := R$ that assigns a reaction $R$ to the channel $o$, the parallel composition $P \parallel Q$ of two protocols, and the spawning $\mathsf{new}\ o : \tau\ \mathsf{in}\ P$ of a new internal channel $o$ of type $\tau$ for use in $P$.

In our version of the IPDL syntax, references $\mathsf{var}(x : \tau)$ to variables and $\mathsf{read}(c : \tau)$ to channels include a typing annotation. We will need these later on when encoding an IPDL construct as a sequence of symbols on a Turing Machine tape; knowing the type $\tau$ will allow us to allocate the correct number of bits for the variable $x$ or the channel $c$.

| | | | |
|---|---|---|---|
| Variables | $x, y, z$ | | |
| Channels | $i, o, c$ | | |
| Channel Sets | $I, O$ | ::= | $\{c_1, \ldots, c_n\}$ |
| Data Types | $\tau, \sigma$ | ::= | $\mathsf{t} \mid 1 \mid \mathsf{Bool} \mid \tau_1 \times \tau_2$ |
| Expressions | $e$ | ::= | $\mathsf{var}(x : \tau) \mid \checkmark \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{app}_{\sigma\to\tau}\ \mathsf{f}\ e \mid (e_1, e_2)$ |
| | | $\mid$ | $\mathsf{fst}_{\sigma\times\tau}\ e \mid \mathsf{snd}_{\sigma\times\tau}\ e$ |
| Reactions | $R, S$ | ::= | $\mathsf{ret}\ e \mid \mathsf{samp}_{\sigma\to\tau}\ \mathsf{d}\ e \mid \mathsf{read}(c : \tau)$ |
| | | $\mid$ | $\mathsf{if}\ e\ \mathsf{then}\ R_1\ \mathsf{else}\ R_2 \mid x : \sigma \leftarrow R;\ S$ |
| Protocols | $P, Q$ | ::= | $0 \mid o := R \mid P \parallel Q \mid \mathsf{new}\ o : \tau\ \mathsf{in}\ P$ |
| Type Contexts | $\Gamma$ | ::= | $\cdot \mid \Gamma, x : \tau$ |
| Channel Contexts | $\Delta$ | ::= | $\cdot \mid \Delta, c : \tau$ |

Fig. 1. Syntax of IPDL.

Typing of protocols in IPDL has the form $\Delta \vdash P : I \to O$, where $\Delta$ is a channel context assigning types to channel names, and $I, O$ are disjoint sets of *input* and *output* channels, respectively. Each output channel in $O$ must be assigned a reaction inside $P$. The exact equational logic of IPDL is parameterized by a finite set of axioms of the form $\Delta \vdash P_1 = P_2 : I \to O$, where $\Delta \vdash P_1 : I \to O$

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

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

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

$$\frac{\Delta;\ \cdot \vdash R : I \to \sigma \qquad \Delta;\ x : \sigma \vdash S : I \to \tau}{\Delta \vdash (\mathsf{new}\ c : \sigma\ \mathsf{in}\ o \coloneqq x \leftarrow \mathsf{read}\ c; S \parallel c \coloneqq R) = (o \coloneqq x \leftarrow R; S)}\ \textsc{fold-bind}$$

$$\frac{\Delta;\ \cdot \vdash (x \leftarrow R_1;\ y \leftarrow R_1;\ \mathsf{ret}\ (x, y)) = (x \leftarrow R_1;\ \mathsf{ret}\ (x, x))}{\Delta \vdash (o_1 \coloneqq R_1 \parallel o_2 \coloneqq x_1 \leftarrow \mathsf{read}\ o_1; R_2) = (o_1 \coloneqq R_1 \parallel o_2 \coloneqq x_1 \leftarrow R_1; R_2)}\ \textsc{subst}$$

$$\frac{\Delta;\ \cdot \vdash R_1 : I \to \tau_1 \qquad \Delta;\ \cdot \vdash R_2 : I \to \tau_2 \qquad \Delta;\ \cdot \vdash (x_1 \leftarrow R_1; R_2) = R_2 : I \to \tau_2}{\Delta \vdash (o_1 \coloneqq R_1 \parallel o_2 \coloneqq x_1 \leftarrow \mathsf{read}\ o_1; R_2) = (o_1 \coloneqq R_1 \parallel o_2 \coloneqq R_2)}\ \textsc{drop}$$

Fig. 2. Selected rules for exact equality of IPDL protocols.

and $\Delta \vdash P_2 : I \to O$. We use these axioms to express example-specific functional assumptions, *e.g.*, the correctness of an encryption/decryption scheme. Figure 2 shows a few illustrative rules of the exact fragment of IPDL.

Rule COMP-NEW allows us to pull a sub-protocol $P$ outside the scope of a channel declaration if the bound channel name does not appear in $P$. Rule ABSORB allows us to discard a sub-protocol that has become unused after simplification. Rule SUBST says that if the computation $R_1$ assigned to a channel $o_1$ is deterministic, then we may replace every occurrence of read $o_1$ by $R_1$. Rule FOLD-BIND states that if we only read from channel $c$ once in the context of a protocol, we can soundly replace read $c$ by the computation assigned to $c$, even if this computation is probabilistic. Finally, rule DROP allows us to drop a vacuous dependency on channel $o_1$ from channel $o_2$ if the computation $R_1$ assigned to $o_1$ does not introduce additional dependencies to $o_2$.

IPDL protocols come with a natural operational semantics. We slightly generalize the semantics given in [Gancher et al. 2023b] to support dynamic-length bitstrings. To this end, we use a special placeholder symbol $\bullet$ and by abuse of terminology we refer to strings $v \in \{0, 1, \bullet\}^\star$ as bitstrings.

Definition 2 (Interpretation). *An interpretation $[\![-]\!]$ for a signature $\Sigma$ associates to:*

- *each type symbol* $\mathsf{t}$ *a subset* $[\![\mathsf{t}]\!] \subseteq \{0, 1, \bullet\}^{|[\![\mathsf{t}]\!]|}$ *of bitstrings of length* $|[\![\mathsf{t}]\!]| \geq 0$*;*
- *each function symbol* $\mathsf{f} : \sigma \to \tau$ *a function* $[\![\mathsf{f}]\!]$ *from* $[\![\sigma]\!]$ *to* $[\![\tau]\!]$*;*
- *each distribution symbol* $\mathsf{d} : \sigma \to \tau$ *a function* $[\![\mathsf{d}]\!]$ *from* $[\![\sigma]\!]$ *to distributions on* $[\![\tau]\!]$.

In the above, we generalize the interpretation $[\![-]\!]$ to all types in the obvious way. To handle partial computations, we follow [Gancher et al. 2023b] and augment the syntax of IPDL protocols to contain intermediate bitstring values

$$\text{Protocols} \quad P, Q \quad ::= \quad o \coloneqq v \mid \ldots$$

We give semantics to IPDL protocols via two main small-step rules, see Figure 3, where we write $1[P]$ for the distribution with unit mass at the protocol $P$, and freely use a distribution in place of a

reaction or a protocol to indicate the obvious lifting of the corresponding construct to distributions on protocols. As reactions are sequential monadic programs, they admit a straightforward small-step semantics $R \longrightarrow \eta$, which we omit. Big-step operational semantics for protocols $P \Downarrow \eta$ performs output and internal steps in an arbitrary order until no more steps are possible, resulting in a unique distribution $\eta$ on protocols.

$$\boxed{P \overset{o \coloneqq v}{\longmapsto} Q}$$

$$\frac{P \overset{o \coloneqq v}{\longmapsto} P'}{P \parallel Q \overset{o \coloneqq v}{\longmapsto} P' \parallel Q[\mathsf{read}\ o \coloneqq \mathsf{val}\ v]} \qquad \frac{Q \overset{o \coloneqq v}{\longmapsto} Q'}{P \parallel Q \overset{o \coloneqq v}{\longmapsto} P[\mathsf{read}\ o \coloneqq \mathsf{val}\ v] \parallel Q'}$$

$$\frac{}{(o \coloneqq \mathsf{val}\ v) \overset{o \coloneqq v}{\longmapsto} (o \coloneqq v)} \qquad \frac{P \overset{o \coloneqq v}{\longmapsto} P' \qquad o \neq c}{(\mathsf{new}\ c : \tau\ \mathsf{in}\ P) \overset{o \coloneqq v}{\longmapsto} (\mathsf{new}\ c : \tau\ \mathsf{in}\ P')}$$

$$\boxed{P \longrightarrow \eta}$$

$$\frac{R \longrightarrow \eta}{(o \coloneqq R) \longrightarrow (o \coloneqq \eta)} \qquad \frac{P \longrightarrow \eta}{P \parallel Q \longrightarrow \eta \parallel Q} \qquad \frac{Q \longrightarrow \eta}{P \parallel Q \longrightarrow P \parallel \eta}$$

$$\frac{P \longrightarrow \eta}{(\mathsf{new}\ c : \tau\ \mathsf{in}\ P) \longrightarrow (\mathsf{new}\ c : \tau\ \mathsf{in}\ \eta)} \qquad \frac{P \overset{c \coloneqq v}{\longmapsto} P'}{(\mathsf{new}\ c : \tau\ \mathsf{in}\ P) \longrightarrow 1[\mathsf{new}\ c : \tau\ \mathsf{in}\ P']}$$

Fig. 3. Small-step operational semantics for IPDL protocols.

### 3.3 Example: Authenticated-To-Secure Channel

We now revisit the running example of [Gancher et al. 2023b]. Alice wants to securely communicate $n$ messages to Bob using an authenticated channel which leaks all messages to the adversary. To do so, we will assume they share a pre-shared key, which enables them to encrypt and decrypt all messages. We show how to encode this protocol and its proof in our DSL. First, we declare the number of sessions as a parameter $n$ to our case study, with the intended interpretation that $n$ is a function of the security parameter $\lambda$:

```
parameter n : nat .
```

*3.3.1 The Assumptions.* We use a version of the IND-CPA assumption, which states that encoding $n$ context-chosen messages with the same secret key is indistinguishable from encrypting zeros. For simplicity, we assume a type of messages with constant length, so that the constant `zeros` need not depend on the length. We express the assumption as an *axiom* about the equality between two protocols:

```
approx-assumption CPA :
  (fam In[i < n] :: msg)
  (fam Enc[i < n] :: ctxt)
```

```
inputs: fam In[i < n] |=
new Key : key in
  (Key ::= samp gen_key ||
  (family Enc[i < n] ::=
     m : msg <- read In[i] ;
     k : key <- read Key ;
     samp enc((m, k))))
 ~
new Key : key in
  (Key ::= samp gen_key ||
  (family Enc[i < n] ::=
     m : msg <- read In[i] ;
     k : key <- read Key ;
     samp enc((zeros, k)))) .
```

The CPA axiom is parameterized by two *families* of channels: `In[i < n]`, for `n` input channels carrying messages, and `Enc[i < n]`, for `n` output channels carrying ciphertexts. The left side of the CPA assumption samples a key on channel `Key` and, for each `i`, encrypts `In[i]` under the key. We do this by declaring a **`family`** of protocols — one for each `i` less than the parameter `n` — which reads from `In[i]`, reads from `Key`, and samples from the distribution of probabilistic encryptions under that key and message. We declare the `Key` channel as *internal* using **`new`** so that the outside context cannot read it. The right side of the CPA assumption is similar, but encrypts `zeros` (defined to be a constant) rather than the message. Importantly, the family `Enc[i < n]` on the right hand side still reads from `In[i]`, since the two protocols must have the same (logical) *timing* behaviors between the channels.

In addition to the CPA assumption, we also have the assumption about the encryption scheme's *correctness*: encrypting and decrypting must return the same message. We encode this in our DSL as an assumption similar to CPA, but since we assume that the encryption scheme is *perfectly* correct, with zero probability of error, we use the declaration **`protocol-assumption`** rather than **`approx-assumption`**:

```
protocol-assumption enc-dec-correctness :
  (chn In :: msg) (chn Key :: key)
  (chn Enc :: ctxt) (chn Dec :: msg)
  inputs: chn In, chn Key |=
  (Enc ::= m : msg <- read In;
           k : key <- read Key;
           samp enc((m, k))) ||
  (Dec ::= c : ctxt <- read Enc;
           k : key <- read Key;
           return dec((c, k)))
   =
  (Enc ::= m : msg <- read In;
           k : key <- read Key;
           samp enc((m, k))) ||
  (Dec ::= i : msg <- read In;
```

```
          return i).
```

*3.3.2 The Protocol.* The Authenticated-To-Secure Channel protocol `Real` now takes the following form:

```
protocol Real =
  new Key : key in
  newfamily Recv[i < n] : ctxt in
  newfamily Send[i < n] : ctxt in
   (Keygen || Alice || Channel || Bob)
    where Alice =
        (family Send[i < n] ::=
           m : msg <- read In[i] ;
           k : key <- read Key ;
           samp enc((m, k)))
    and Bob =
        (family Out[i < n] ::=
           c : ctxt <- read Recv[i] ;
           k : key <- read Key ;
           return dec((c, k)))
    and Channel =
        (family Leak[i < n] ::= read Send[i]) ||
        (family Recv[i < n] ::=
           c : ctxt <- read Send[i] ;
           ok : unit <- read Ok[i] ;
           return c)
    and Keygen =
        (Key ::= samp gen_key) .
```

The body of the protocol is a parallel composition of the key generating functionality `Keygen`, the two parties `Alice` and `Bob`, and the authenticated channel functionality `Channel`. Alice encrypts each input with the shared key stored on the internal channel `Key`, samples a ciphertext from the resulting distribution, and sends the result to the authenticated channel functionality on the channel `Send[i]`. Bob reads the ciphertext forwarded to him from the authenticated channel functionality on the channel `Recv[i]`, decrypts it with the shared key, and outputs the plaintext on the channel `Out[i]`. The channels `Send[i]` and `Recv[i]` are connected by the `Channel` functionality, which allows the adversary to read/schedule messages via `Leak[i]` and `Ok[i]`.

*Proving Protocols Secure.* Our protocol is named `Real` because we will compare it to an *ideal* version, where Alice and Bob communicate directly through a secure channel, without the need for encryption. As in UC, we do this by proving that the `Real` protocol is indistinguishable from the `Ideal` protocol composed with a *simulator* `Sim` that can emulate the `Leak[i]` messages without knowledge of the secret messages:

$$\texttt{Real} \approx \texttt{Ideal} \mid\mid \texttt{Sim}.$$

The key advantage of IPDL [Gancher et al. 2023b] is that it enables proofs through *equational reasoning* principles. Using basic identities of protocols (*e.g.*, inlining definitions of channels into

other channels) and assumptions (*e.g.*, CPA), one proves a protocol secure by progressively rewriting the real protocol into its idealization (plus the simulator).

We illustrate some key steps of the proof. The ideal functionality for our secure channel example has two output channels per session $i$: the adversarial output channel `LeakMsgRcvdIdAdv[i]` reads the message from `In[i]` and lets the adversary know that a message has been received – by passing a term of the `unit` type – but divulges nothing about the value of the message. The channel `Out[i]` first waits to receive a confirmation on the adversarial input channel `OkMsgAdvId[i]` that gives the green light to the functionality to process the message. It subsequently reads the message from `In[i]` on behalf of Alice, and outputs it on behalf of Bob:

```
Ideal =
  (family LeakMsgRcvdIdAdv[i < n] ::=
     m : msg <- read In[i];
     return ()) ||
  (family Out[i < n] ::=
     okMsg : unit <- read OkMsgAdvId[i];
     m : msg <- read In[i];
     return m)
```

The simulator turns the adversarial interface of the real protocol into the adversarial interface of the ideal functionality, thereby converting any adversary for the real protocol into an adversary for the functionality. In our example, the channels `LeakMsgRcvdIdAdv[i]` and `OkCtxtAdvNet[i]` are the inputs to the simulator, while the channels `LeakCtxtNetAdv[i]` and `OkMsgAdvId[i]` are the outputs.

Hence, upon receiving the information from the ideal functionality that a message has been received, the simulator must conjure up a ciphertext to leak to the adversary. This is accomplished by randomly generating a secret key and encrypting the chosen message `zeros` in each session. Upon receiving the approval from the adversary for the generated ciphertext, the simulator gives the approval to the ideal functionality to output the message:

```
Sim =
  new Key : key in
    (Key ::= samp gen_key(())) ||
    (family LeakCtxtNetAdv[i < n] ::=
       x : unit <- read LeakMsgRcvdIdAdv[i];
       k : key <- read Key;
       samp enc((zeros(()), k))) ||
    (family OkMsgAdvId[i < n] ::=
       okCtxt : unit <- read OkCtxtAdvNet[i];
       return okCtxt)
```

As the channel `OkMsgAdvId[i]` carries a deterministic computation, we want to inline the computation into `Out[i]` to yield the following:

```
family Out[i] i < n ::=
  okCtxt : unit <- read OkCtxtAdvNet[i];
  m : msg <- read In[i];
  return m
```

This form of substitution is justified by the rule SUBST from Figure 2, which says that if the computation $R_1$ assigned to a channel $o_1$ is deterministic, then we may replace every occurrence of read $o_1$ by $R_1$. However, for the rule SUBST to apply, we must first massage the protocol into a form where `OkMsgAdvId[i]` appears immediately next to `Out[i]`. In the implementation of [Gancher et al. 2023b], this required tedious manual transformations that, *e.g.*, permute channels inside a parallel composition or move channels in and out of scope of a new channel declaration (lines 87–104 of MultiChan.v). In our tool, all the necessary massaging is performed automatically, and we can simply write:

```
subst fam OkMsgAdvId into fam Out then
subst fam LeakMsgRcvdIdAdv into fam LeakCtxtNetAdv then
absorb fam LeakMsgRcvdIdAdv then
absorb fam OkMsgAdvId
```

A crucial step in simplifying the real protocol is to conceptually separate the encryption and decryption actions from the message-passing by introducing new internal channels `Enc[i]` and `Dec[i]` along with their definitions:

```
add internal family Enc i < n typed: ctxt
 assigned: m : msg <- read In[i];
           k : key <- read Key;
           samp enc((m, k)) then
add internal family Dec i < n typed: msg
 assigned: c : ctxt <- read Enc[i];
           k : key <- read Key;
           return dec((c, k))
```

We can now modify the channels `Send[i]` and `Out[i]` to read from `Enc[i]` and `Dec[i]` directly:

```
sym from change fam Send with
    e : ctxt <- read Enc[i];
    return e
    in currentProtocol(
       subst fam Enc into fam Send ) then
sym from change fam Out with
    okCtxt : unit <- read OkCtxtAdvNet[i];
    d : msg <- read Dec[i];
    return d
    in currentProtocol(
       subst fam Dec into fam Out )
```

We can now invoke the correctness assumption in each individual session to cancel the effect of encryption followed by decryption. In [Gancher et al. 2023b], the generalization to $n$ sessions was stated as a separate lemma with a nontrivial manual proof (lines 67 – 103 in CPA.v). In our code, the corresponding proof looks like this:

```
by induction on i with variable x (
  use assumption enc-dec-correctness
    on chn Dec[x], chn Enc[x] )
```

The above code snippet proves by induction on $i < n$ that if the channels `Dec(i)` with $i < x$ rewrite from the original formulation that performs the decryption to the new formulation that simply reads off the message `In[i]`, so does the channel `Dec[x]`.

On the other hand, our CPA assumption is applied just once across all sessions upon encountering the following protocol snippet:

```
new Key : key in
  ((Key ::= samp gen_key(())) ||
   (family Enc[i] i < n ::=
      m : msg <- read In[i];
      k : key <- read Key;
      samp enc((m, k))))
```

We invoke the approximate CPA assumption as shown below:

```
use approx assumption cpa
```

This yields the following protocol snippet:

```
new Key : key in
  ((Key ::= samp gen_key(())) ||
   (family Enc[i] i < n ::=
      m : msg <- read In[i];
      k : key <- read Key;
      samp enc((zeros(()), k))))
```

A final step in the proof *folds* the internal channels `Enc[i]` and `Dec[i]` that we introduced earlier into the rest of the protocol:

```
fold fam Enc into fam LeakCtxtNetAdv then
fold fam Dec into fam Out
```

This is justified by the rule FOLD-BIND in Figure 2, which states that if we only read from channel $c$ once, we can soundly replace read $c$ by the computation assigned to $c$, even if this computation is probabilistic.

*Concrete Security Bounds.* Such a proof could be carried out in the IPDL logic alone. However, that proof would only guarantee asymptotic security with respect to the custom symbolic bounds defined in [Gancher et al. 2023b]. In this work, we aim for *concrete* security with respect to a low-level Turing Machine semantics: a precise probabilistic bound for the difference in the probability that an adversary can distinguish `Real` from `Ideal || Sim`.

After encoding the proof in the DSL, our tool computes the following bounds:

```
indistinguishability assumption cpa :
count: 1
context: n * |msg| * 6 + n * |ctxt| * 3 + n * 96 + 12
```

Here **count** denotes the number of times the CPA assumption was applied, and **context** bounds the maximal size of the program context in which it was applied. The expressions `|msg|` and `|ctxt|` denote the concrete length of bitstrings needed to represent the two types.

Given these two quantities, our main theorem allows us to derive the following concrete security bound on the distinguishing advantage for our protocol:

$$\left|\Pr\left[\mathsf{Adv} \rightleftharpoons \mathsf{Real} = 1\right] - \Pr\left[\mathsf{Adv} \rightleftharpoons \mathsf{Ideal} \mid\mid \mathsf{Sim} = 1\right]\right| \leq \varepsilon_{\mathsf{cpa}}.$$

Here, $\mathsf{Adv} \rightleftharpoons P$ denotes the interaction of the adversary Adv with a protocol $P$. The value $\varepsilon_{\mathsf{cpa}}$ is the maximal distinguishing advantage for the CPA assumption against any adversary with computational "cost" at most $\mathcal{P}(\mathbf{context})$. Here $\mathcal{P}$ is a fixed polynomial, and "cost" bounds the runtime of the adversary and the number of states in its Turing Machines (among other quantities). In other words, our theorem exactly bounds the security error in our protocol's proof by the error present in the IND-CPA game against the reduction.

Our novel strategy for computing concrete security bounds is to implement an *interpreter* for IPDL programs as a Turing Machine, and compute its cost as the aforementioned polynomial $\mathcal{P}$. The polynomial takes the size of the interpreted program as its argument, and bounds the number of Turing Machine states and transitions that the resulting interpretation will need.

### 3.4 The Main Theorem

Before giving technical details, we now discuss our main result informally. Roughly speaking, if $P$ is approximately equal to $Q$, then the advantage that an adversary has in distinguishing $P$ and $Q$ is a reasonable combination of the distinguishing advantages against each indistinguishability assumptions by an adversary whose computational resources are only slightly larger than those of the original adversary.

Theorem (Soundness of approximate equality of protocols, informal). *There exists a polynomial $\mathcal{P}(x, y, z)$ with the following property. Given:*

- *finitely many built-in functions that can be computed with cost at most $C_{\mathsf{sem}} \in \mathbb{N}$ and can be approximated by probabilistic Turing Machines with error at most $\eta_{\mathsf{sem}} \in \mathbb{Q}_{\geq 0}$;*
- *indistinguishability assumptions $\vdash P^1 \approx Q^1, \ldots, \vdash P^n \approx Q^n$;*
- *a proof of indistinguishability $\vdash P \approx Q$ with output bounds $\mathbf{count}_i$ and $\mathbf{context}_i$ for the $i$-th indistinguishability assumption;*
- *an adversary Adv for $P/Q$ that computes with cost at most $C_{\mathsf{adv}} \in \mathbb{N}$;*
- *axiom bounds $\varepsilon^1, \ldots, \varepsilon^n \in \mathbb{Q}_{\geq 0}$ with the property that for any adversary $\mathsf{Adv}^i$ for $P^i/Q^i$ such that $\mathsf{Adv}^i$ computes with cost at most $\mathcal{P}(C_{\mathsf{sem}}, C_{\mathsf{adv}}, \mathbf{context}_i)$, we have*

$$\left|\Pr\left[\mathsf{Adv}^i \overset{[\![-]\!]}{\rightleftharpoons} P^i = 1\right] - \Pr\left[\mathsf{Adv}^i \overset{[\![-]\!]}{\rightleftharpoons} Q^i = 1\right]\right| \leq \varepsilon^i,$$

*we have*

$$\left|\Pr\left[\mathsf{Adv} \overset{[\![-]\!]}{\rightleftharpoons} P = 1\right] - \Pr\left[\mathsf{Adv} \overset{[\![-]\!]}{\rightleftharpoons} Q = 1\right]\right| \leq \sum_{i=1}^{n} \mathbf{count}_i * \varepsilon^i.$$

In other words, we can bound the probability in distinguishing $P$ from $Q$ by the sum of the maximal probabilities of violating an indistinguishability axiom. The actual theorem we prove (Thm. 1) is slightly more general, as we allow the distribution symbols to be general distributions that are only *approximated* by Turing Machines, which requires an *error term* to appear in the theorem.

The probability $\varepsilon^i$ must work for *every* adversary with cost at most $\mathcal{P}(C_{\mathsf{sem}}, C_{\mathsf{adv}}, \mathbf{context}_i)$; thus, the larger the adversary cost, the looser the bound $\varepsilon^i$ must be. Indeed, if $\mathcal{P}(C_{\mathsf{sem}}, C_{\mathsf{adv}}, \mathbf{context}_i) = \mathbf{context}_i^{500}$, the bound is still polynomial but not particularly meaningful. Crucially, our proof is *constructive*: we are able to *compute* the polynomial as

$$\mathcal{P}(x, y, z) = y^2 + 8yz + 15z^2 + (|\Sigma_f| + |\Sigma_d| + 1)x + 34y + 47z + O(1).$$

This polynomial, where $|\Sigma_f|$ and $|\Sigma_d|$ are the number of function and distribution symbols in our signature, serves as a precise bound on the reduction overhead incurred when using a cryptographic assumption. We are able to achieve such a concrete bound precisely by using a low-level computational semantics for adversaries as Turing Machines. The exact polynomial, which we include in the appendix, is computed once and for all, and counts the precise number of steps that our TMs take, along with binding the TM's number of states, tapes, and symbols (to encode a protocol on a tape, we use additional symbols besides 0, 1). This is in contrast to almost all prior cryptographic work targeting MPC, which either reasons about runtime informally (without being able to compute the polynomial bound on reduction overhead) or subverts reasoning about runtime via error terms that contain reductions themselves [Brzuska et al. 2018]. A notable exception is Barbosa et al. [Barbosa et al. 2021], which adds a Hoare logic for running time to EasyCrypt, at the cost of being limited to imperative programs (and thus imperative encodings of protocols), and manual proof effort for each runtime bound.

*3.4.1 Overview of Results.* In this work, we build a new cost-aware proof system on top of the existing exact equational logic of IPDL. The rest of our framework diverges significantly. In particular:

- We carry out explicit cryptographic reductions instead of using symbolic adversaries. We represent an adversary (Section 4.3) as a tuple of (essentially arbitrary) probabilistic algorithms for scheduling interactions, updating the adversary's internal state, querying the protocol for output channel values, and assigning new input channel values. When absorbing a program context $Q$ into the adversary, we explicitly extend the adversary's state by the encoding of $Q$ as a sequence of symbols on the Turing Machine tape.
- We deliver *concrete security bounds* (Section 4.1 and 4.2) instead of symbolic ones. In particular, the bound induced by invoking an approximate congruence rule, which allows us to conclude $P \parallel Q \approx P' \parallel Q$ from $P \approx P'$, is the length $\|Q\|$ of the aforementioned Turing Machine encoding (plus some overhead).
- We give a natural definition of computational indistinguishability (Sec. 4.4) that does not involve reasoning about syntactic contexts (as in prior work [Gancher et al. 2023b]). Informally, we define two families of protocols to be indistinguishable if for any polynomial $p(\lambda)$ and negligible function $\eta(\lambda)$, there exists a negligible function $\varepsilon(\lambda)$ such that for any sufficiently large $\lambda$ and any adversary Adv with cost bounded by $p(\lambda)$, the distinguishing advantage of Adv is bounded by $\varepsilon(\lambda)$.
- We carry out an explicit analysis of errors induced by a probabilistic Turing Machine that ends up in a non-accepting state with a negligible probability. This probability makes an appearance in the concrete bounds we derive (Thm. 1).

## 4 COST-AWARE SYNTAX AND SEMANTICS FOR IPDL

In this section, we extend the IPDL logic [Gancher et al. 2023b] to handle *cost-aware* proofs; that is, proofs which guarantee precise concrete security bounds. Our main theorem for concrete security bounds assumes a sound ambient theory for the strict fragment of IPDL.

While the exact fragment of IPDL is exactly what is desired for cryptographic proofs, the approximate fragment is missing a crucial point of reasoning. In particular, its soundness proof is in terms of *symbolic* bounds, which abstract away the underlying cost semantics of IPDL protocols; in short, it does not reason about runtime. Because of this, the prior approximate logic of IPDL does not prove the same class of security results generally accepted by the cryptographic community.

For the rest of this section, we assume a fixed signature $\Sigma$ with type constants $\mathrm{t}_1, \ldots, \mathrm{t}_{|\Sigma_\mathrm{t}|}$.

### 4.1 Approximate Congruence

Our cost-aware equational theory consists of three layers. Firstly, we have *approximate conguence*, see Figure 4, which applies a program context to a single *approximate axiom*, resulting in the judgment $\Delta \vdash P \cong Q : I \to O$ ctxt $\psi$ for axiom # $k$. We assume a set of $n$ approximate axioms of the form $\Delta^k \vdash P^k \approx Q^k : I^k \to O^k$ for $1 \le k \le n$, where $\Delta^k \vdash P^k : I^k \to O^k$ and $\Delta^k \vdash Q^k : I^k \to O^k$. These axioms capture cryptographic assumptions on computational indistinguishability.

Here, the ctxt parameter tracks the increase in the adversary's resources incurred by the proof. A typical proof step in the exact fragment transforms the protocol into a form where an approximate axiom applies. We subsequently carry out an approximate congruence step, where we use the approximate axiom to replace a small protocol fragment nested inside an arbitrary program context by its computationally indistinguishable counterpart.

The program context is formally a part of the adversary, and as such it must be resource-bounded for the indistinguishability assumption to apply. Some nesting patterns do not effect any change on the adversary's resources: for example, a simple renaming of channels (rule EMBED); the formal addition of an unused channel $i$ to the protocol's inputs $I$ (rule INPUT-UNUSED), in which case any value assigned by the adversary to channel $i$ will leave the protocol unchanged; or the introduction of an internal channel $o : \tau$ (rule CONG-NEW), in which case the adversary will never query $o$ because internal channels are only visible in the scope of their declaration.

On the other hand, composing two approximately equal protocols $P \approx P'$ with another protocol $Q$ requires the adversary to simulate the interaction of the program context $Q$ with $P$ versus $P'$. In other words, the adversary *absorbs* $Q$ and the protocol becomes part of the new adversary's code. In particular, the number of symbols needed for encoding the adversary's code on a Turing Machine tape increases, and the parameter $\psi$ approximates this increase. As rule CONG-COMP shows, composition with protocol $Q$ incurs $\|Q\| + 3$ additional symbols: $\|Q\|$ symbols for encoding $Q$; a parallel composition symbol to combine the original code with the code for $Q$; and two parenthesis symbols "(", ")" for enclosing the composition. We emphasize that the exact numbers here are not crucial; what matters is that we eventually deliver a (reasonable) polynomial in $\lambda$.

We show how to compute the Turing Machine bound of an IPDL construct in Section A. This bound, and consequently the ctxt parameter $\psi$, is not a natural number but a function $\psi(t_1, \ldots, t_{|\Sigma_t|}) : \mathbb{N}^{|\Sigma_t|} \to \mathbb{N}$ that is *monotonically increasing in each argument*. When encoding a protocol $Q$ as a sequence of symbols on a Turing Machine tape, we invariably encounter variables $x$ of type $\tau$. At this point, we do not know how many bits we will need to encode values of type $\tau$, because the type constants $\mathsf{t} \in \Sigma$ are yet uninterpreted. Instead, we leave the size of each type constant as a variable to the function $\psi$, which will later be instantiated by the appropriate natural number according to $[\![-]\!]$.

### 4.2 Approximate and Asymptotic Equality

In the *approximate equality* of protocols, see Figure 5, we chain together a sequence of strict equalities and approximate congruence transformations to obtain the judgment $\Delta \vdash P \approx Q : I \to O$ count $\xi$ ctxt $\psi$. The parameter $\xi$ counts the number of axiom invocations for each of the $n$ axioms. One application of the $k$-th approximate axiom incurs a count that maps $k$ to 1 and all other axioms to 0. The use of transitivity requires us to add up the respective values of $\xi$ per each axiom (rule TRANS). Even though each individual axiom invocation introduces a negligible error, summing up exponentially many negligible errors might not be negligible, which is why we need to keep track of the number of times each axiom is applied. The parameter $\psi$ tracks the maximum size of a program context in which each axiom is applied.

$$\boxed{\Delta \vdash P \cong Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}$$

$$\frac{}{\Delta^k \vdash P^k \cong Q^k : I^k \to O^k \text{ ctxt } 0 \text{ for axiom \# } k}\ \textsc{axiom}$$

$$\frac{c \notin I \cup O \qquad \Delta \vdash P \cong Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}{\Delta \vdash P \cong Q : I \cup \{c\} \to O \text{ ctxt } \psi \text{ for axiom \# } k}\ \textsc{input-unused}$$

$$\frac{\phi : \Delta_1 \to \Delta_2 \qquad \Delta_2 \vdash P \cong Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}{\Delta_1 \vdash \phi^\star(P) \cong \phi^\star(Q) : \phi^\star(I) \to \phi^\star(O) \text{ ctxt } \psi \text{ for axiom \# } k}\ \textsc{embed}$$

$$\frac{\Delta \vdash P \cong P' : I \cup O_2 \to O_1 \text{ ctxt } \psi \text{ for axiom \# } k \qquad \Delta \vdash Q : I \cup O_1 \to O_2}{\Delta \vdash P \parallel Q \cong P' \parallel Q : I \to O_1 \cup O_2 \text{ ctxt } (\psi + \|Q\| + 3) \text{ for axiom \# } k}\ \textsc{cong-comp}$$

$$\frac{\Delta, o : \tau \vdash P \cong P' : I \to O \cup \{o\} \text{ ctxt } \psi \text{ for axiom \# } k}{\Delta \vdash \left(\text{new } o : \tau \text{ in } P\right) \cong \left(\text{new } o : \tau \text{ in } P'\right) : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}\ \textsc{cong-new}$$

Fig. 4. Approximate congruence of IPDL protocols.

$$\boxed{\Delta \vdash P \approx Q : I \to O \text{ count } \xi \text{ ctxt } \psi}$$

$$\frac{\Delta \vdash P = Q : I \to O}{\Delta \vdash P \approx Q : I \to O \text{ count } (i \mapsto 0) \text{ ctxt } (i \mapsto 0)}\ \textsc{strict}$$

$$\frac{\Delta \vdash P \cong Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}{\Delta \vdash P \approx Q : I \to O \text{ count } \left(k \mapsto 1, i \neq k \mapsto 0\right) \text{ ctxt } \left(k \mapsto \psi, i \neq k \mapsto 0\right)}\ \textsc{approx-cong}$$

$$\frac{\Delta \vdash P_1 \approx P_2 : I \to O \text{ count } \xi \text{ ctxt } \psi}{\Delta \vdash P_2 \approx P_1 : I \to O \text{ count } \xi \text{ ctxt } \psi}\ \textsc{sym}$$

$$\frac{\Delta \vdash P_1 \approx P_2 : I \to O \text{ count } \xi_1 \text{ ctxt } \psi_1 \qquad \Delta \vdash P_2 \approx P_3 : I \to O \text{ count } \xi_2 \text{ ctxt } \psi_2}{\Delta \vdash P_1 \approx P_3 : I \to O \text{ count } \left(i \mapsto \xi_1(i) + \xi_2(i)\right) \text{ ctxt } \left(i \mapsto \max(\psi_1(i), \psi_2(i))\right)}\ \textsc{trans}$$

Fig. 5. Approximate equality for IPDL protocols.

Finally, analogously to [Gancher et al. 2023b], we define the *asymptotic equality* of two protocol families, see Figure 6, as functions of the security parameter $\lambda \in \mathbb{N}$. Informally speaking, if two protocol families are asymptotically equal, then any *resource-bounded* adversary cannot distinguish them with greater than negligible error. Formally, we assume a finite set of *approximate axiom families* of the form $\left\{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\right\}_{\lambda \in \mathbb{N}}$, where $\{P_\lambda\}, \{Q_\lambda\}$ are two protocol families with pointwise-identical typing judgments. If the axiom families comprising our asymptotic theory are clear from the context, we will omit them from the asymptotic equality judgment.

$$\boxed{\left\{\Delta_\lambda^1 \vdash P_\lambda^1 \approx Q_\lambda^1 : I_\lambda^1 \to O_\lambda^1\right\}_\lambda, \ldots, \left\{\Delta_\lambda^n \vdash P_\lambda^n \approx Q_\lambda^n : I_\lambda^n \to O_\lambda^n\right\}_\lambda \vdash \left\{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\right\}_\lambda}$$

$$\frac{\begin{array}{c}\forall \lambda, \Delta_\lambda^i \vdash P_\lambda^i \approx Q_\lambda^i : I_\lambda^i \to O_\lambda^i, i = 1, \ldots, n \vdash \Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda \text{ count } \xi_\lambda \text{ ctxt } \psi_\lambda \\ \forall i, \xi_{(\cdot)}^i = \mathrm{O}(\mathrm{poly}(\lambda)) \qquad \forall i, \psi_{(\cdot)}^i = \mathrm{O}\big(\mathrm{poly}(\lambda, t_1, \ldots, t_{|\Sigma_\mathrm{t}|})\big)\end{array}}{\left\{\Delta_\lambda^1 \vdash P_\lambda^1 \approx Q_\lambda^1 : I_\lambda^1 \to O_\lambda^1\right\}_\lambda, \ldots, \left\{\Delta_\lambda^n \vdash P_\lambda^n \approx Q_\lambda^n : I_\lambda^n \to O_\lambda^n\right\}_\lambda \vdash \left\{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\right\}_\lambda}$$

Fig. 6. Asymptotic equality for IPDL protocol families.

For any fixed $\lambda$, we obtain an approximate theory by selecting from each axiom family the particular axiom corresponding to $\lambda$. Similarly, from each of the two protocol families we select the protocol corresponding to $\lambda$, which gives us two concrete protocols to equate approximately. We recall that an approximate equality judgment is tagged by a pair of parameters $\xi$ and $\psi$, and for each axiom $1 \leq i \leq n$ we have $\xi^i \in \mathbb{N}$ and $\psi^i : \mathbb{N}^{|\Sigma_\mathrm{t}|} \to \mathbb{N}$, where $|\Sigma_\mathrm{t}|$ is the number of type constants declared in our ambient signature $\Sigma$. Fixing the axiom $i$ and letting the security parameter $\lambda$ vary thus gives us two functions $\mathbb{N} \to \mathbb{N}$ and $\mathbb{N}^{|\Sigma_\mathrm{t}|+1} \to \mathbb{N}$, and we require that these be bounded by polynomials in the appropriate number of variables. Unlike [Gancher et al. 2023b], we do not impose a bound on the channel context $\Delta$, which can be particularly burdensome to check in a formal tool. Instead, our definition of an adversary ensures that it only interacts with the protocol via a bounded number of channels.

### 4.3 Adversaries for IPDL Protocols

To seamlessly account for the possible renaming of channel names, an adversary for protocols of type $\Delta \vdash I \to O$ is allowed to operate in a larger context $\Delta'$ that subsumes the original context $\Delta$ via an *embedding* $\phi : \Delta' \to \Delta$. Here $\phi$ is an injective, type-preserving mapping that specifies how to rename channels in $\Delta$ to fit in the larger context $\Delta'$. In this larger context, we specify the adversarial input channels $I'$ that the adversary will query for a value, and the adversarial output channels $O'$ that the adversary will assign values to. The adversarial inputs $I'$ will be a subset of the protocol outputs $O$, appropriately translated along $\phi$. Dually, the protocol inputs $I$, appropriately translated along $\phi$, will be a subset of the adversarial outputs $O'$. In the interaction between the adversary and the protocol, every query for a value of a channel $o \in I'$ will extract the value of the channel $o \in \phi^\star(O)$ as computed by the the protocol, and pass it on to the adversary. Conversely, an input on channel $i \in \phi^\star(I)$ to the protocol occurs after the adversary computes the value of the channel $i \in O'$.

Since our adversaries will be resource-bounded, we need to bind the number of interactions or *rounds* between the adversary and the protocol. In each round, the adversary examines its internal state to determine the type of interaction to perform next, and steps to a new state. This transition function is a partial probabilistic function of type $\mathrm{St} \rightharpoonup \big(\{\bot\} \cup I' \cup O'\big) \times \mathrm{St}$. That is, for any internal state $s$ the adversary probabilistically decides among: *1)* no interaction, coupled with stepping to a new state $s'$; *2)* querying a channel $o \in I'$, coupled with stepping to a new state $s'$; *3)* an assignment to a channel $i \in O'$, coupled with stepping to a new state $s'$; or *4)* halting, in which case the game between the adversary and the protocol ends without a decision Boolean. We use this last option to capture probabilistic computations that only succeed up to a negligible error.

If the adversary queries channel $o \in I'$ and receives a value $v$ as a response to the query, it updates its internal state according to an input assignment function of type $[\![\tau]\!] \times \mathrm{St} \to \mathrm{St}$, where $\tau$ is the type of the channel $o$ in $\Delta'$. That is, for any value $v \in [\![\tau]\!]$ and any state $s$, the adversary

steps to a new state $s'$ that records the value $v$ as a result of the query. If the adversary chooses a value assignment to a channel $i \in O'$, the value $v$ – if any – is determined by an output valuation function of type $\mathsf{St} \to [\![\tau]\!] \cup \{\bot\}$, where $\tau$ is the type of the channel $i$ in $\Delta'$. After completing the designated number of rounds, the adversary converts its internal state to a final decision Boolean according to a function of type $\mathsf{St} \to \{0,1\}$.

To bind the complexity of the aforementioned operations, we implement them as Turing Machines[1]. For convenience, we allow TMs with multiple tapes. As is standard, in the initial configuration all tapes except the first are fully blank. The internal state of the adversary is typically encoded as a bitstring, containing *e.g.*, register values together with the sequence of instructions to be executed, if we wish to view the adversary as an essentially arbitrary probabilistic program. For our purposes, it is convenient to allow additional symbols besides $0, 1$ on our TM tapes: when justifying the COMP-CONG rule of our theory, we suitably compose the adversary with the common context $Q$. The protocol $Q$ thus becomes integrated into the new adversary's code. Instead of encoding protocols as bitstrings, we will suitably enrich our baseline set of symbols so that we can faithfully capture IPDL code.

DEFINITION 3 (ADVERSARY). *Fix an interpretation $[\![-]\!]$ for $\Sigma$. An* adversary *for protocols $\Delta \vdash I \to O$ is a tuple $\left(\Delta', I', O', \phi, \#_{\mathsf{rnd}}, \#_{\mathsf{tape}}, \mathsf{Symb}, \mathsf{St}, s_\star, \mathsf{T}, \{\mathsf{I}_o\}_{o \in I'}, \{\mathsf{O}_i\}_{i \in O'}, \mathsf{D}\right)$, where*

- *$\Delta'$ is a channel context;*
- *$I' \subseteq \Delta'$ is a set of channels that the adversary can query for a value;*
- *$O' \subseteq \Delta'$ is a set of channels to which the adversary can assign a value;*
- *$\phi : \Delta' \to \Delta$ is an embedding of $\Delta$ into $\Delta'$;*
- *$\#_{\mathsf{rnd}} \geq 1$ is the number of rounds the adversary will perform;*
- *$\#_{\mathsf{tape}} \geq 1$ is the number of TM tapes at our disposal;*
- *$\mathsf{Symb}$ is a finite set of additional symbols that will be used to encode the adversary's internal state;*
- *$\mathsf{St} \subseteq \left(\{0,1\} \sqcup \mathsf{Symb}\right)^l$ is a set of strings of a fixed length $l \geq 1$ consisting of symbols drawn from the disjoint union of the sets $\{0,1\}$ and $\mathsf{Symb}$;*
- *$s_\star \in \mathsf{St}$ is the initial state;*
- *$\mathsf{T}$ is a probabilistic TM that computes a partial function $\mathsf{St} \rightharpoonup \left(\{\bot\} \cup I' \cup O'\right) \times \mathsf{St}$, with $\#_{\mathsf{tape}}$-many tapes using symbols from the set $\{0,1\} \sqcup \{\bot\} \sqcup (I' \cup O') \sqcup \mathsf{Symb}$,*
- *$\mathsf{I}_o$ where $o : \tau \in \Delta$ is a TM that computes a function $\mathsf{St} \times [\![\tau]\!] \to \mathsf{St}$, with $\#_{\mathsf{tape}}$-many tapes using symbols from the set $\{0,1\} \sqcup \mathsf{Symb}$,*
- *$\mathsf{O}_i$ where $i : \tau \in \Delta$ is a TM that computes a function $\mathsf{St} \to [\![\tau]\!] \cup \{\bot\}$, with $\#_{\mathsf{tape}}$-many tapes using symbols from the set $\{0,1\} \sqcup \{\bot\} \sqcup \mathsf{Symb}$,*
- *$\mathsf{D}$ is a TM that computes a function $\mathsf{St} \to \{0,1\}$, with $\#_{\mathsf{tape}}$-many tapes using symbols from the set $\{0,1\} \sqcup \mathsf{Symb}$.*

*We furthermore require that $I' \subseteq \phi^\star(O)$, $\phi^\star(I) \subseteq O'$, and $\phi^\star(O) \cap O' = \emptyset$.*

The probabilistic TM $\mathsf{T}$ that computes the transition function can terminate in a non-accepting state with probability $> 0$. We will be interested in families of adversaries where this *error* as a function of the security parameter is negligible.

DEFINITION 4 (ADVERSARIAL ERROR). *We say that an adversary* Adv has error up to $\varepsilon \in \mathbb{Q}_{\geq 0}$, *written* $\mathsf{err}(\mathsf{Adv}) \leq \varepsilon$, *if for any state $s \in \mathsf{St}$ the transition function $\mathsf{T}(s)$ is undefined with probability*

[1]When using Turing Machines to compute (probabilistic) functions, we only consider (probabilistic) Turing Machines that have a finite runtime $N \in \mathbb{N}$; *i.e.*, for every input in the domain, after $N$ (probabilistic) transitions the TM ends up in a configuration where no further transitions are possible. That is, the TM has either reached an accepting state or it has halted after reading a symbol for which no transition is possible in the current state.

$\leq \varepsilon$. *In other words, when* $\mathsf{T}$ *is run with the initial tape contents* $s$*, it halts in a non-accepting state with probability* $\leq \varepsilon$.

To ensure that the adversary does not have access to computationally expensive functions such as the discrete logarithm, we need to impose a bound on its computational resources. We will be interested in families of adversaries where the bound as the function of the security parameter is polynomial.

Definition 5 (Resource-bounded adversaries). *We say that an adversary* Adv *has* cost at most $K \in \mathbb{N}$*, written* $|\mathsf{Adv}| \leq K$*, if:*

- $\#_{\mathsf{rnd}}, \#_{\mathsf{tape}} \leq K$;
- $|I'| \leq K$ *and for each* $o \in I'$ *with* $o : \tau \in \Delta'$*, we have* $|\tau| \leq K$,
- $|O'| \leq K$ *and for each* $i \in O'$ *with* $i : \tau \in \Delta'$*, we have* $|\tau| \leq K$,
- $|\mathsf{Symb}| \leq K$;
- *the length* $k$ *of a state* $s \in \mathsf{St}$ *is* $\leq K$;
- *the number of states of each*[2] *TM* $\mathsf{T}, \mathsf{I}_o, \mathsf{O}_i, \mathsf{D}$ *is* $\leq K$;
- *the runtime of each TM* $\mathsf{T}, \mathsf{I}_o, \mathsf{O}_i, \mathsf{D}$ *is* $\leq K$.

Definition 6 (Interaction). *Fix an interpretation* $[\![-]\!]$ *for* $\Sigma$*. Let* Adv *be an adversary for protocols* $\Delta \vdash I \longrightarrow O$ *and let* $\Delta \vdash P : I \longrightarrow O$*. We define* $\mathsf{Adv} \overset{[\![-]\!]}{\rightleftharpoons} P$ *to be the probability sub-distribution on Booleans induced by the algorithm in Figure 7.*

Algorithm $\mathsf{Adv} \overset{[\![-]\!]}{\rightleftharpoons} P$:

$s := s_\star; P := \phi^\star(P)$
for $\#_{\mathsf{rnd}}$ rounds
  $P' \leftarrow P\Downarrow; (q, s') \leftarrow \mathsf{T}(s)$
  if $q = \bot$ then $s := s'; P := P'$
  else if $q = i \in O'$ then
    if $\mathsf{O}_i(s') = v$ *for some* $v$ then
      $s := s'; P := P'[\mathsf{read}\ i := \mathsf{val}\ v]$
    else $s := s'; P := P'$
  else if $q = o \in I'$ then
    if $(o := v) \in P'$ *for some* $v$ then $s := \mathsf{I}_o(v, s'); P := P'$
    else $s := s'; P := P'$
return $\mathsf{D}(s)$

Fig. 7. Interaction of an an adversary Adv with an IPDL protocol $P$.

In Figure 7, the adversary interacts with the protocol through the specified number of rounds. The algorithm maintains a state variable $s$ along with a protocol variable $P$, which we respectively

[2]Instead of having a separate Turing Machine $\mathsf{I}_o$ for each channel $o \in I'$ we could have required a single Turing Machine that performs the computation across all channels in $I'$, and analogously for $\mathsf{O}_i$. However, this is unnecessary as the number of channels in both $I'$ and $O'$ is $\mathsf{O}(\mathsf{poly}(\lambda))$, and the current formulation is more convenient for our purposes.

initialize to the initial state $s_\star$ and the original protocol $P$, appropriately embedded in $\Delta'$. In each round, the protocol $P$ probabilistically evolves to a new protocol $P'$. Independently, the adversary probabilistically computes the type of interaction $q \in \{\bot\} \cup I' \cup O'$ together with a new state $s'$ according to $\mathsf{T}(s)$. If $q = \bot$, in which case no interaction takes place, the state and the protocol are updated to $s'$ and $P'$. If $q = i$ for some $i \in O'$, we compute $\mathsf{O}_i(s')$ to see if in the adversary's current state $s'$ the channel $i$ carries a value $v$. If this is the case, we update the state to $s'$ while computing a new protocol $P'[\text{read } i := \text{val } v]$. Otherwise we update the state and the protocol to $s'$ and $P'$. Finally, if $q = o$ for some $o \in I'$, we examine the protocol $P'$ to see if the output channel $o$ carries a value $v$. If this is the case, we compute a new adversary state $\mathsf{I}_o(v, s')$, while updating the protocol to $P'$. Otherwise we update the state and the protocol to $s'$ and $P'$. After completing the prescribed number of rounds, we obtain a decision Boolean[3] $\mathsf{D}(s)$ based on the adversary's current state.

### 4.4 Computational Indistinguishability

A family of interpretations is PPT (probabilistic polynomial-time) if it assigns polynomial lengths to type symbols t, and PPT-computable functions to function symbols f and distribution symbols d. A small caveat is that a random distribution on a subset $S \subseteq \{0,1\}^n$ of bitstrings is in general computable by a probabilistic Turing Machine only up to a small error $\varepsilon$, which is the probability that the TM does not end up in an accepting state. In effect, the TM computes a distribution $\mu$ on $S \cup \{\bot\}$ with $\mu(\bot) = \varepsilon$. To relate $\mu$ to our original distribution on $S$, we introduce the following:

Definition 7 (Approximating distributions). *Let $S \subseteq \{0,1\}^n$ be a subset of bitstrings of a fixed length. We say that a distribution $\mu_1$ on $S \cup \{\bot\}$* approximates *a distribution $\mu_2$ on $S$ with error $0 \le \varepsilon \le 1$ if there are distributions $\eta_1, \eta_2$ on $S$ such that $\mu_1 = (1-\varepsilon)\eta_1 + \varepsilon 1[\bot]$ and $\mu_2 = (1-\varepsilon)\eta_1 + \varepsilon\eta_2$.*

A function $\varepsilon : \mathbb{N} \to \mathbb{Q}_{\ge 0}$ is negligible if it is eventually smaller than the inverse of any polynomial: *for any $n \in \mathbb{N}$, there exists $N \in \mathbb{N}$ such that for all $\lambda \ge N$ we have $\varepsilon(\lambda) \le \frac{1}{\lambda^n}$.*

Definition 8 (PPT family of interpretations). *We say that a family $\{[\![-]\!]_\lambda\}_{\lambda\in\mathbb{N}}$ of interpretations for $\Sigma$ is* PPT *if there is a polynomial $p(\lambda)$, a negligible function $\eta(\lambda)$, and a natural number $N \in \mathbb{N}$ such that the following holds:*

- *For all type symbols* t, $|\mathsf{t}|_\lambda \le p(\lambda)$ *if* $\lambda \ge N$.
- *For all function symbols* $\mathsf{f} : \sigma \to \tau$, *the function* $[\![\mathsf{f}]\!]_\lambda$ *from bitstrings* $[\![\sigma]\!]_\lambda$ *to bitstrings* $[\![\tau]\!]_\lambda$ *is computable by a deterministic Turing Machine* $\mathsf{TM}_\lambda$ *with symbols* 0, 1. *Both the number of states and the runtime of* $\mathsf{TM}_\lambda$ *are* $\le p(\lambda)$ *if* $\lambda \ge N$.
- *For all distribution symbols* $\mathsf{d} : \sigma \to \tau$, *the function* $[\![\mathsf{d}]\!]_\lambda$ *from bitstrings* $[\![\sigma]\!]_\lambda$ *to distributions on bitstrings* $[\![\tau]\!]_\lambda$ *is computable up to an error* $\eta(\lambda)$ *by a probabilistic Turing Machine* $\mathsf{TM}_\lambda$ *with symbols* 0, 1. *Specifically, for every* $v \in [\![\sigma]\!]_\lambda$, *the distribution* $\mathsf{TM}_\lambda(v)$ *on* $[\![\tau]\!]_\lambda \cup \{\bot\}$ *approximates* $[\![\mathsf{d}]\!]_\lambda(v)$ *with error* $\le \eta(\lambda)$. *Both the number of states and the runtime of* $\mathsf{TM}_\lambda$ *are* $\le p(\lambda)$ *if* $\lambda \ge N$.

We are now ready to give the definition of computational indistinguishability. We will be only interested in indistinguishability with respect to a PPT family of interpretations.

Definition 9 (Computational Indistinguishability). *Consider a family $\{[\![-]\!]_\lambda\}_{\lambda\in\mathbb{N}}$ of interpretations for $\Sigma$. Let $\{\Delta_\lambda \vdash P_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathbb{N}}$ and $\{\Delta_\lambda \vdash Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathbb{N}}$ be two protocol families with identical typing judgments. We say that $\{P_\lambda\}$ and $\{Q_\lambda\}$ are* indistinguishable *under $\{[\![-]\!]_\lambda\}$,*

[3] Strictly speaking, the interaction $\text{Adv} \overset{[\![-]\!]}{\rightleftharpoons} P$ is only a *sub*-distribution on Booleans, since $\mathsf{T}(s)$ may halt without a result. As the probability of this happening will be negligible, we gloss over this technical point here.

*written*

$$\left\{[\![-]\!]_\lambda\right\}_{\lambda\in\mathbb{N}} \vDash \left\{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\right\}_{\lambda\in\mathbb{N}}$$

*if for any polynomial $p(\lambda)$ and negligible function $\eta(\lambda)$, there exists a negligible function $\varepsilon(\lambda)$ and an $N \in \mathbb{N}$ such that for any $\lambda \geq N$ and any adversary* Adv *for protocols $\Delta_\lambda \vdash I_\lambda \to O_\lambda$ with respect to the interpretation $[\![-]\!]_\lambda$ such that $|\mathsf{Adv}| \leq p(\lambda)$ and $\mathsf{err}(\mathsf{Adv}) \leq \eta(\lambda)$, we have*[4]

$$\left|\Pr\left[\mathsf{Adv} \overset{[\![-]\!]_\lambda}{\rightleftharpoons} P_\lambda = 1\right] - \Pr\left[\mathsf{Adv} \overset{[\![-]\!]_\lambda}{\rightleftharpoons} Q_\lambda = 1\right]\right| \leq \varepsilon(\lambda).$$

If the family of interpretations is clear from the context, we may omit it from the computational indistinguishability judgment.

### 4.5 Concrete and Asymptotic Security

We can now present our main result in full:

Theorem 1 (Soundness of approximate equality of protocols). *There exists a polynomial $\mathcal{P}(x, y, z)$ such that for any*

- *interpretation $[\![-]\!]$ for $\Sigma$ for which there are $C_{\mathsf{sem}} \in \mathbb{N}$ and $\eta_{\mathsf{sem}} \in \mathbb{Q}_{\geq 0}$ such that*
  - *for all type symbols* t, $|\mathsf{t}| \leq C_{\mathsf{sem}}$*;*
  - *for all function symbols* f, $[\![\mathsf{f}]\!]$ *is computable by a TM with symbols* 0, 1 *such that the number of states and the runtime are $\leq C_{\mathsf{sem}}$; and*
  - *for all distribution symbols* d, $[\![\mathsf{d}]\!]_\lambda$ *is computable up to an error $\eta_{\mathsf{sem}}$ by a probabilistic TM with symbols* 0, 1 *such that the number of states and the runtime are $\leq C_{\mathsf{sem}}$;*
- *approximate axioms $\Delta^1 \vdash P^1 \approx Q^1 : I^1 \to O^1, \ldots, \Delta^n \vdash P^n \approx Q^n : I^n \to O^n$;*
- *derivation $\Delta \vdash P \approx Q : I \to O$* count $\xi$ ctxt $\psi$*;*
- *adversary* Adv *for protocols of type $\Delta \vdash I \to O$ such that $|\mathsf{Adv}| \leq C_{\mathsf{adv}}$ and $\mathsf{err}(\mathsf{Adv}) \leq \eta_{\mathsf{adv}}$ for some $C_{\mathsf{adv}} \in \mathbb{N}$ and $\eta_{\mathsf{adv}} \in \mathbb{Q}_{\geq 0}$,*
- *context bounds $C^1_{\mathsf{ctxt}}, \ldots, C^n_{\mathsf{ctxt}} \in \mathbb{N}$ such that $\psi^i(|\mathsf{t}_1|, \ldots, |\mathsf{t}_{|\Sigma_\mathsf{t}|}|) \leq C^i_{\mathsf{ctxt}}$; and*
- *axiom bounds $\varepsilon^1, \ldots, \varepsilon^n \in \mathbb{Q}_{\geq 0}$ with the property that for any adversary $\mathsf{Adv}^i$ for protocols $\Delta^i \vdash I^i \to O^i$ such that*

$$|\mathsf{Adv}^i| \leq \mathcal{P}(C_{\mathsf{sem}}, C_{\mathsf{adv}}, C^i_{\mathsf{ctxt}})$$

*and $\mathsf{err}(\mathsf{Adv}^i) \leq \max(\eta_{\mathsf{sem}}, \eta_{\mathsf{adv}})$, we have*

$$\left|\Pr\left[\mathsf{Adv}^i \overset{[\![-]\!]}{\rightleftharpoons} P^i = 1\right] - \Pr\left[\mathsf{Adv}^i \overset{[\![-]\!]}{\rightleftharpoons} Q^i = 1\right]\right| \leq \varepsilon^i,$$

*we have*

$$\left|\Pr\left[\mathsf{Adv} \overset{[\![-]\!]}{\rightleftharpoons} P = 1\right] - \Pr\left[\mathsf{Adv} \overset{[\![-]\!]}{\rightleftharpoons} Q = 1\right]\right| \leq \sum_{i=1}^{n} \xi^i * \left(\varepsilon^i + 2 * C^i_{\mathsf{ctxt}} * \eta_{\mathsf{sem}}\right).$$

We briefly explain where the error term $2 * C^i_{\mathsf{ctxt}} * \eta_{\mathsf{sem}}$ comes from. The original error $\eta_{\mathsf{sem}}$ is multiplied by $C^i_{\mathsf{ctxt}}$ because $C^i_{\mathsf{ctxt}}$ provides an upper bound on how many times we perform probabilistic samplings when executing the absorbed protocol, and hence serves as a proxy for how much the total error accumulates. Finally, we multiply the overall error term by two because computing the distinguishing advantage of an adversary $A$ for protocols $P_1 \parallel Q$ versus $P_2 \parallel Q$, we accumulate errors twice: once when comparing $A \rightleftharpoons (P_1 \parallel Q)$ against $B \rightleftharpoons P_1$ (where $B$ is the reduced adversary obtained from $A$ by absorbing the program context $Q$), and then again when

[4]Instead of comparing probabilities for 1, we could have likewise used 0: the probability $\Pr[b = 0]$ that the decision Boolean $b$ is 0 is only negligibly different from $1 - \Pr[b = 1]$, since the probability that the game ends without a decision Boolean is negligible. This follows from the fact that the error is negligible and the number of rounds is $\mathrm{O}(\mathrm{poly}(\lambda))$.

comparing $B \rightleftharpoons P_2$ against $A \rightleftharpoons (P_2 \parallel Q)$. Our proof of Theorem 1 relies on two intermediate results. The first one says that strict equality of protocols implies perfect indistinguishability against any adversary (not just a resource-bounded one); for proof see Section D.

Lemma 1 (Perfect indistinguishability). *For any interpretation* $[\![-]\!]$ *for* $\Sigma$*, derivation* $\Delta \vdash P = Q : I \to O$*, and adversary* $\mathsf{Adv}$ *for protocols* $\Delta \vdash I \to O$*, we have*

$$\left|\Pr\left[\mathsf{Adv} \overset{[\![-]\!]}{\rightleftharpoons} P = 1\right] - \Pr\left[\mathsf{Adv} \overset{[\![-]\!]}{\rightleftharpoons} Q = 1\right]\right| = 0.$$

The next result is the *absorption lemma*, the proof of which we sketch in Section F, which allows us to absorb a protocol into an adversary at the cost of correspondingly increasing its cost and error. This is precisely where the polynomial $\mathcal{P}(x, y, z)$ in Theorem 1 comes from:

Lemma 2 (Absorption). *There exists a polynomial* $\mathcal{P}(x, y, z) \geq y$ *such that for any*

- *interpretation* $[\![-]\!]$ *for* $\Sigma$ *for which there are* $C_{\mathsf{sem}} \in \mathbb{N}$, $\eta_{\mathsf{sem}} \in \mathbb{Q}_{\geq 0}$ *such that*
  - *for all type symbols* $\mathsf{t}$, $|\mathsf{t}| \leq C_{\mathsf{sem}}$,
  - *for all function symbols* $\mathsf{f}$, $[\![\mathsf{f}]\!]$ *is computable by a TM with symbols* $0, 1$ *such that the number of states and the runtime are* $\leq C_{\mathsf{sem}}$*, and*
  - *for all distribution symbols* $\mathsf{d}$, $[\![\mathsf{d}]\!]$ *is computable up to error* $\eta_{\mathsf{sem}}$ *by a probabilistic TM with symbols* $0, 1$ *such that the number of states and the runtime are* $\leq C_{\mathsf{sem}}$*,*
- *adversary* $\mathsf{Adv}$ *for protocols of type* $\Delta \vdash I \to O_1 \cup O_2$ *such that* $|\mathsf{Adv}| \leq C_{\mathsf{adv}}$ *and* $\mathsf{err}(\mathsf{Adv}) \leq \eta_{\mathsf{adv}}$ *for some* $C_{\mathsf{adv}} \in \mathbb{N}$ *and* $\eta_{\mathsf{adv}} \in \mathbb{Q}_{\geq 0}$*,*
- *protocol* $\Delta \vdash Q : I \cup O_1 \to O_2$*,*

*we have an adversary* $\mathsf{Adv}_{\mathcal{R}}$ *for protocols* $\Delta \vdash I \cup O_2 \to O_1$ *with*

$$|\mathsf{Adv}_{\mathcal{R}}| \leq \mathcal{P}\left(C_{\mathsf{sem}}, C_{\mathsf{adv}}, \|Q\|(|\mathsf{t}_1|, \ldots, |\mathsf{t}_{|\Sigma_{\mathsf{t}}|}|)\right)$$

*and* $\mathsf{err}(\mathsf{Adv}_{\mathcal{R}}) \leq \max(\eta_{\mathsf{sem}}, \eta_{\mathsf{adv}})$ *such that for any protocol* $\Delta \vdash P : I \cup O_2 \to O_1$ *we have*

$$\left|\Pr\left[\mathsf{Adv} \overset{[\![-]\!]}{\rightleftharpoons} P \parallel Q = 1\right] - \Pr\left[\mathsf{Adv}_{\mathcal{R}} \overset{[\![-]\!]}{\rightleftharpoons} P = 1\right]\right| \leq \|Q\|(|\mathsf{t}_1|, \ldots, |\mathsf{t}_{|\Sigma_{\mathsf{t}}|}|) * \eta_{\mathsf{sem}}.$$

The high-level idea behind the proof of Theorem 1 is to restructure the derivations of approximate equality so that all invocations of the rule EMBED are carried out first, followed by applications of the rule INPUT-UNUSED, which are in turn followed by invocations of the rule CONG-COMP, and finally by applications of the rule CONG-NEW. The new layered form of our approximate judgments is shown in Figures 9 and 10.

Crucially, we collapse a sequence of applications of the CONG-COMP rule with common contexts $Q_1, \ldots, Q_n$ into a single application with the combined common context $Q_1 \parallel \ldots \parallel Q_n$. This is necessary because every time we absorb a protocol into the adversary, we increase the adversary's resources: *e.g.*, the number of rounds that $\mathsf{Adv}_{\mathcal{R}}$ takes is more than double the original number of rounds. Thus, if we carried out this process $\lambda$-many times, we would see an exponential increase in the adversary's resources.

Our second result (Theorem 2; for proof see Section E) concerns asymptotic security and serves as a sanity check for the concrete bounds we derived. Our asymptotic theory is said to be sound if each of its axioms is sound:

Definition 10. *We say that an approximate axiom family* $\left\{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\right\}_{\lambda \in \mathbb{N}}$ *is* sound *with respect to a family of interpretations* $\left\{[\![-]\!]_\lambda\right\}_{\lambda \in \mathbb{N}}$ *if* $\left\{[\![-]\!]_\lambda\right\}_{\lambda \in \mathbb{N}} \vDash \left\{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\right\}_{\lambda \in \mathbb{N}}$.

Theorem 2 (Soundness of asymptotic equality of protocols). *For any*

- *protocol families* $\{\Delta_\lambda \vdash P_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathbb{N}}$ *and* $\{\Delta_\lambda \vdash Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathbb{N}}$ *with identical typing judgments;*
- *PPT family of interpretations* $\{[\![-]\!]_\lambda\}_{\lambda\in\mathbb{N}}$ *for* $\Sigma$*; and*
- *an asymptotic theory that is sound with respect to* $\{[\![-]\!]_\lambda\}_{\lambda\in\mathbb{N}}$*;*

*we have that*

$$\vdash \{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathbb{N}}$$

*implies*

$$\vDash \{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathbb{N}}.$$

## 5 CASE STUDIES

We now briefly describe our four case studies, focusing on the Multi-Party GWM protocol. In Fig. 8, we summarize our case studies by lines of code (second number in columns 2 and 3) and runtime to verify (column 4). Whenever applicable, we compare against the corresponding case study from [Gancher et al. 2023b] implemented in Coq (first number in columns 2 and 3). Our largest case study currently takes 12 minutes to verify. This number can likely be reduced with performance optimizations and/or by splitting the proof across multiple files that can be verified in parallel[5].

| Case Study | Proofs (LoC) | | Defs (LoC) | | Runtime |
|---|---|---|---|---|---|
| | Coq | DSL | Coq | DSL | DSL |
| Auth-To-Secure Channel [Maurer 2012] | 128 | 61 | 97 | 143 | 1.38s |
| DHKE-OTP [Barbosa et al. 2021] | 532 | 164 | 183 | 371 | 3.3s |
| Multi-Party Coin Flip [Blum 1983a] | 1905 | 256 | 114 | 311 | 5.1s |
| Multi-Party GMW [Goldreich et al. 1987] | - | 9102 | - | 5738 | 12.08m |

Fig. 8. Case Studies: size of proofs and definitions, and runtime to verify.

*Authenticated-To-Secure Channel.* Our smallest case study constructs a secure channel from an authenticated one. Alice wants to communicate $q$ messages to Bob using an authenticated channel. The authenticated channel is not secure: it leaks each message to the adversary, and waits to receive an *ok* message back from the adversary before delivering the in-flight message. Thus, the adversary cannot modify any of the messages but can read and delay them for any amount of time. To transmit information securely, Alice sends encryptions of her messages, which Bob decrypts using a shared key not known to the adversary. The indistinguishability assumption states that the encryption scheme is CPA-secure.

*One-Time Pad (OTP) From Diffie-Hellman Key Exchange.* In symmetric-key encryption, the sender (Alice) and the receiver (Bob) need to agree on a shared secret key. One such key-agreement protocol is the Diffie-Hellman Key Exchange (DHKE), which assumes a cyclic group $G$ of a prime order with generator $g$. The indistinguishability assumption is the *decisional Diffie-Hellman (DDH)* assumption: as long as the exponents $k, l$ are generated uniformly, even if the adversary knows the values $g^k$ and $g^l$, they will be unable to distinguish $(g^k)^l$ from a uniformly generated element of $G$.

We subsequently use the DHKE protocol to turn an authenticated channel into a one-time pad (OTP) that delivers a single secret message from Alice to Bob. Our proof is modular: in the first step, we establish that the DHKE protocol can be replaced with its idealization. In the second step, we prove that the resulting OTP protocol itself reduces to an idealization.

[5]The case studies and the sources can be found at https://github.com/concrete-bounds-for-mpc-proofs/concrete-bounds-for-mpc-simulation-proofs.

*Multi-Party Coin Flip.* We implement a protocol (originally due to [Blum 1983b]) where $N+2$ parties labeled $0, \ldots, N-1$ reach a Boolean consensus. At the start of the protocol, each party commits to a randomly-generated Boolean. After all parties have committed, each party opens its commit. The consensus is the Boolean sum of all commits. We prove the protocol secure against a malicious attacker in the case when party $N$ is corrupt, party $N+1$ is honest, and any other party is arbitrarily honest or corrupt. As this protocol is perfectly secure, there are no indistinguishability assumptions.

*Multi-Party GMW Protocol.* In the multi-party GMW protocol [Goldreich et al. 1987], $N+2$ parties securely compute the value of a given Boolean circuit built out of *xor*-, *and*-, and *not* gates. The inputs to the circuit are divided among the parties, and no party has access to the inputs of any other. Each party maintains its share of the value $v$ computed by each gate, and summing up the shares of all parties yields $v$. We prove the protocol secure in the case when party $N$ is semi-honest, party $N+1$ is honest, and any other party is arbitrarily honest or semi-honest.

The protocol consists of the $N+2$ parties, plus an instance of the 1-Out-Of-4 Oblivious Transfer (OT) protocol for each gate and each pair of parties $n < m$, where $n$ is the sender and $m$ is the receiver. The code for each party is separated into three parts: in the initial phase, each party computes and distributes everyone's shares for each of its inputs. In the inductive phase, each party computes their share of each gate by induction on the ambient circuit. At last, in the final phase, parties send their shares of each output wire to one another and add them up to compute the result.

The shares of each gate are computed as follows. In the case of an *input* gate, the parties use the corresponding input share from the initial phase. In the case of a *not* gate, parties $0, \ldots, N$ simply copy their share of the incoming wire, whereas party $N+1$ negates its share. If the gate is an *xor* gate, the resulting share is the sum of the shares of the incoming two wires. The case of an *and* gate is the most complex. The sum of everybody's shares must equal $(x_0 \oplus \ldots \oplus x_{N+1}) * (y_0 \oplus \ldots \oplus y_{N+1})$, where $x_n, y_n$ are the respective shares of party $n$ on the incoming two wires. We have

$$(x_0 \oplus \ldots \oplus x_{N+1}) * (y_0 \oplus \ldots \oplus y_{N+1}) = \bigoplus_i \bigoplus_j x_i * y_j$$

Parties $n$ and $m$ engage in a 1-Out-Of-4 OT exchange to compute $(x_n * y_m) \oplus (x_m * y_n)$. There are four possible combinations of values that $x_m, y_m$ can take, and party $n$ computes the value of $(x_n * y_m) \oplus (x_m * y_n)$ for each. This offers party $m$ four messages to choose from, and he selects the one corresponding to the actual values of $x_m, y_m$. A small caveat: in the exchange as described above, party $m$ would still be able to infer the value of party $n$'s shares in certain cases: *e.g.*, if $x_m = 0$ and $y_m = 1$, party $m$ gets the share $x_n$ as the result of the exchange. To prevent this, party $n$ encodes her messages by masking them with a random Boolean $b$ that only she knows. To offset for the presence of this Boolean, she includes it in her own share $b \oplus (x_n * y_n)$.

We assume four opaque protocols representing four implementations of the 1-Out-Of-4 Oblivious Transfer, one for each of the possible combinations of an honest/semi-honest sender and receiver. We also assume four cryptographic hardness axioms, stating that each OT implementation is approximately equal to an ideal 1-Out-Of-4 OT functionality. In the first step of our proof, we replace each OT implementation by its the corresponding functionality. The rest of the proof is carried out in the exact fragment of our proof system.

Since the last party is by assumption honest, the simulator does not have access to its inputs. Therefore, any computation that depends on the value of the inputs belonging to the last party must be eliminated. In particular, all shares of the last party must be eliminated. Instead, the simulator only computes shares for parties $0, \ldots, N$ in the inductive part, and replaces every mention of the

last party's share in its leakage by the quantity $x \oplus \left(\bigoplus_{i \leq N} x_i\right)$. Here $x$ is the value carried by the gate as computed by the ideal functionality, and leaked to the simulator by a semi-honest party.

In this case study, the main challenge for formal verification is how to effectively carry out proofs by induction nested three levels deep: one for the arbitrary Boolean circuit, and two for each of the arbitrarily many parties communicating with another party. Additionally, we have to account for an essentially arbitrary combination of honest/semi-honest parties, as all but the last two parties can be honest or semi-honest. Every one of these factors reflects into the actual code of the protocol, and has to be taken into account when computing concrete security bounds. The final concrete bound computed by our tool is as follows:

```
indistinguishability assumption HH2HH :
count: |{(n, m, k) s.t. n < N + 2, m < N + 2, k < K,
  when isHonest(n) and isHonest(m)}|
context:
  N * K * max(N * 434 + 990, N * 434 + 1011, N * 607 + 1329, N * 434 + 983) +
  N * N * K * max(|1OutOf4OTReal-Honest-Honest|, 543) + N * N * K * 218 +
  N * K * max(|1OutOf4OTReal-Honest-Honest|, 543) * 4 + N * K * 932 +
  K * max(N * 434 + 990, N * 434 + 1011, N * 607 + 1329, N * 434 + 983) * 2 +
  N * N * maxValue(I, N + 2) * 300 + N * N * 68 + K * 992 + N * 352 +
  K * max(|1OutOf4OTReal-Honest-Honest|, 543) * 3 +
  (K - 1) * max(|1OutOf4OTReal-Honest-Honest|, 543) +
  N * maxValue(I, N + 2) * 1169 + maxValue(I, N + 2) * 1138 + 509

indistinguishability assumption SHH2SHH : ...
indistinguishability assumption HSH2HSH : ...
indistinguishability assumption SHSH2SHSH : ...
```

Here `K` is the number of wires in the circuit, `HH2HH` is the indistinguishability assumption stating that the 1-Out-Of-4 Oblivious Transfer protocol when both the sender and the receiver are honest can be soundly replaced by the corresponding functionality, and `maxValue(I, N + 2)` selects the maximum of `I(0)`, ..., `I(N + 1)`, where `I(n)` denotes the number of inputs to party `n`. We omit the bounds for the other indistinguishability assumptions, as these are entirely analogous. As expected, the bound **`context`** is `O(N * N * K)`, since we have one instance of an OT protocol for each of the $K$ wires and each pair of parties. As we can see from the the bound **`count`**, the total number of times we invoke an indistinguishability assumption is `(N + 2) * (N + 2) * K`, since we replace each of the OT protocols with its idealization. The indistinguishability assumption applicable to a particular OT instance is conditional on whether the sender and receiver are honest or semi-honest, respectively.

*DSL Implementation.* Our DSL serves as a layer of abstraction over the implementation internals, hiding low-level proof details from the user. For each rule, we have language construct in the DSL that handles its application. For example, an application of the rule SUBST in Fig. 2 is written in the DSL as **`subst`** `o1` **`into`** `o2`. Internally, this translates into a call of a Maude strategy, which applies the substitution rule over a protocol without explicitly writing applications of congruence or exchange rules. Moreover, in many cases these strategies are generated on the fly, instead of requiring the user to manually specify them.

## 6 FUTURE WORK

We present a promising formalism for developing verified cryptographic proofs for MPC protocols. We compute concrete security bounds by bounding the size of the program context in which indistinguishability assumptions are applied, and analyzing the cost of the interpreter that executes the program context. Currently, our main source of over-approximation is the runtime of this interpreter, which is currently quadratic due to using linear scans for inputs (see Section F); however, reducing the overhead to near-linear is possible using standard techniques.

Our tool can model arbitrary (static) corruption scenarios, including protocols that tolerate $k/N$ corruptions (either semi-honest or malicious); formalizing a protocol that demonstrates this capability is one avenue for future work. In in addition to verifying even more sophisticated MPC protocols (*e.g.*, garbled circuits), we want to develop new algorithms for automatically *synthesizing* cryptographic simulators, rather than requiring them to be manually encoded. Finally, we aim to extend our system to handle more expressive classes of protocols, such as those exhibiting threshold behavior (*e.g.*, consensus protocols).

## REFERENCES

J. Almeida, M. Barbosa, G. Barthe, François Dupressoir, B. Grégoire, Vincent Laporte, and Vitor Pereira. 2017. A Fast and Verified Software Stack for Secure Function Evaluation. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017).

David Baelde, Caroline Fontaine, Adrien Koutsos, Guillaume Scerri, and Theo Vignon. 2024. A Probabilistic Logic for Concrete Security . In *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*. IEEE Computer Society, Los Alamitos, CA, USA, 324–339. https://doi.org/10.1109/CSF61375.2024.00046

Manuel Barbosa, Gilles Barthe, Benjamin Grégoire, Adrien Koutsos, and Pierre-Yves Strub. 2021. 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*.

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

Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. 1997. A Concrete Security Treatment of Symmetric Encryption. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*. IEEE, 394–403.

Bruno Blanchet. 2008. A Computationally Sound Mechanized Prover for Security Protocols. *IEEE Trans. Dependable Secur. Comput.* 5, 4 (2008), 193–207. https://doi.org/10.1109/TDSC.2007.1005

Manuel Blum. 1983a. Coin Flipping by Telephone a Protocol for Solving Impossible Problems. *SIGACT News* 15, 1 (1983), 23–27. https://doi.org/10.1145/1008908.1008911

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

Chris Brzuska, Antoine Delignat-Lavaud, Cédric Fournet, Konrad Kohbrok, and Markulf Kohlweiss. 2018. State separation for code-based game-playing proofs. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III 24*. Springer, 222–249.

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.

Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott (Eds.). 2007. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*. Lecture Notes in Computer Science, Vol. 4350. Springer. https://doi.org/10.1007/978-3-540-71999-1

Ankush Das, Jan Hoffmann, and Frank Pfenning. 2018. Work Analysis with Resource-Aware Session Types. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 305–314.

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).

Joshua Gancher, Sydney Gibson, Pratap Singh, Samvid Dharanikota, and Bryan Parno. 2023a. Owl: Compositional Verification of Security Protocols via an Information-Flow Type System. In *44th IEEE Symposium on Security and Privacy, SP 2023,*

*San Francisco, CA, USA, May 21-25, 2023*. IEEE, 1130–1147. https://doi.org/10.1109/SP46215.2023.10179477

Joshua Gancher, Kristina Sojakova, Xiong Fan, Elaine Shi, and Greg Morrisett. 2023b. A Core Calculus for Equational Proofs of Cryptographic Protocols. *Proc. ACM Program. Lang.* 7, POPL (2023), 866–892. https://doi.org/10.1145/3571223

Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play Any Mental Game. In *Proceedings of the 19th annual ACM symposium on Theory of Computing*. ACM, 218–229.

Helene Haagh, Aleksandr Karbyshev, Sabine Oechsner, Bas Spitters, and Pierre-Yves Strub. 2018. Computer-Aided Proofs for Multiparty Computation with Active Security. *2018 IEEE 31st Computer Security Foundations Symposium (CSF)* (2018), 119–131.

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.

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.

Christian Skalka and Joseph Near. 2024. Language-Based Security for Low-Level MPC. In *Proceedings of the 26th International Symposium on Principles and Practice of Declarative Programming*. 1–14.

Alley Stoughton and Mayank Varia. 2017. Mechanizing the Proof of Adaptive, Information-Theoretic Security of Cryptographic Protocols in the Random Oracle Model. *2017 IEEE 30th Computer Security Foundations Symposium (CSF)* (2017), 83–99.

Ionut Ţuţu. 2022. SpeX: A Rewriting-Based Formal Specification Environment. In *Recent Trends in Algebraic Development Techniques - 26th IFIP WG 1.3 International Workshop, WADT 2022, Aveiro, Portugal, June 28-30, 2022, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 13710)*, Alexandre Madeira and Manuel A. Martins (Eds.). Springer, 163–178. https://doi.org/10.1007/978-3-031-43345-0_8

Xingyu Xie, Yifei Li, Wei Zhang, Tuowei Wang, Shizhen Xu, Jun Zhu, and Yifan Song. 2024. GAuV: A Graph-Based Automated Verification Framework for Perfect Semi-Honest Security of Multiparty Computation Protocols. *IACR Cryptol. ePrint Arch.* (2024).

## A TURING MACHINE BOUNDS

The Turing Machine bound of a type $\tau$ is straightforward:

$$\begin{aligned} \|\mathsf{t}_i\| &:= t_i \\ \|1\| &:= 0 \\ \|\mathsf{Bool}\| &:= 1 \\ \|\tau_1 \times \tau_2\| &:= \|\tau_1\| + \|\tau_2\| \end{aligned}$$

The encoding of variables $x$ of type $\tau$ uses the symbols "(", "var", ":", ")" in addition to the de Bruijn index of the variable $x$, encoded as a single symbol, and the encoding of the type annotation $\tau$. For expressions ✓, true, false, we use the corresponding symbols "✓", "true", "false" and the two parenthesis symbols "(", ")". For an application f $e$ of a function of type $\sigma \to \tau$, we use the symbols "(", "app", "→", ")" in addition to the function symbol f, encoded as a single symbol, and the encodings of the two type annotations $\sigma, \tau$ and the expression $e$. To encode a pair $(e_1, e_2)$, we will only need the encodings of the two expressions $e_1$ and $e_2$. Finally, to encode first and second projections, we will use the symbols "(", "fst", "snd", "×", "of", ) in addition to the encodings of

the two type annotations $\sigma, \tau$ and the expression $e$.

$$\begin{aligned}
\|\mathsf{var}(x : \tau)\| &:= \|\tau\| + 5 \\
\|\checkmark\| &:= 3 \\
\|\mathsf{true}\| &:= 3 \\
\|\mathsf{false}\| &:= 3 \\
\|\mathsf{app}_{\sigma\to\tau}\ \mathsf{f}\ e\| &:= \|\sigma\| + \|\tau\| + \|e\| + 5 \\
\|(e_1, e_2)\| &:= \|e_1\| + \|e_2\| \\
\|\mathsf{fst}_{\sigma\times\tau}\ e\| &:= \|\sigma\| + \|\tau\| + \|e\| + 5 \\
\|\mathsf{snd}_{\sigma\times\tau}\ e\| &:= \|\sigma\| + \|\tau\| + \|e\| + 5
\end{aligned}$$

For a ret $e$, we use the symbols "(", "ret", )" in addition to the encoding of the expression $e$. For a sampling samp d $e$ from a distribution of type $\sigma \to \tau$, we use the symbols "(", "samp", "↠", ")" in addition to the distribution symbol d, encoded as a single symbol, and the encodings of the two type annotations $\sigma, \tau$ and the expression $e$. For a read $c$ from a channel of type $\tau$, we use the symbols "(", "read", ":", ")" in addition to the de Bruijn index of the channel $c$, encoded as a single symbol, and the encoding of the type annotation $\tau$. Furthermore, we will need one extra symbol: one of "input-to-query", "input-queried", "input-not-to-query". When encoding a protocol $Q : I \cup O_1 \to O_2$ coming from the COMP-CONG rule, we use "input-to-query" or "input-queried" if we are reading from a channel $o_1 \in O_1$, according to whether we have already queried the channel $o_1$, and "input-not-to-query" otherwise.

For a conditional if $e$ then $R_1$ else $R_2$, we use the symbols "(", "if", "then", "else", ")" in addition to the encodings of the expression $e$ and the two reactions $R_1, R_2$. Finally, to encode a bind, we use the symbols "{", "_", ":", "←", ";", "}" in addition to the encodings of the type annotation $\sigma$ and the two reactions $R$ and $S$. The symbol "_" is used in lieu of the bound variable name $x$ and stands for de Bruijn index 0.

$$\begin{aligned}
\|\mathsf{ret}\ e\| &:= \|e\| + 3 \\
\|\mathsf{samp}_{\sigma\to\tau}\ \mathsf{d}\ e\| &:= \|\sigma\| + \|\tau\| + \|e\| + 5 \\
\|\mathsf{read}(c : \tau)\| &:= \|\tau\| + 6 \\
\|\mathsf{if}\ e\ \mathsf{then}\ R_1\ \mathsf{else}\ R_2\| &:= \|e\| + \|R_1\| + \|R_2\| + 5 \\
\|x : \sigma \leftarrow R;\ S\| &:= \|\sigma\| + \|R\| + \|S\| + 6
\end{aligned}$$

To encode the zero protocol 0, we use the single symbol "0". For an assignment $o := R$, we use the symbols "[", ":=", "react", "]" in addition to the de Bruijn index of the channel $c$, encoded as a single symbol, and the encoding of the reaction $R$. For a parallel composition $P \parallel Q$, we use the symbols "(", "‖", ")" in addition to the encodings of the two protocols $P$ and $Q$. Finally, for the declaration of a new channel new $o : \tau$ in $P$, we use the symbols "new", "_", ":", "in", "wen" in addition to the encodings of the typing annotation $\tau$ and the protocol $P$. The symbol "_" is used in lieu of the bound channel name $c$ and stands for de Bruijn index 0. The symbol "wen" indicates the end of the binding scope.

$$\begin{aligned}
\|0\| &:= 1 \\
\|o := R\| &:= \|R\| + 5 \\
\|P \parallel Q\| &:= \|P\| + \|Q\| + 3 \\
\|\mathsf{new}\ c : \tau\ \mathsf{in}\ P\| &:= \|\tau\| + \|P\| + 5
\end{aligned}$$

## B ENCODING PROTOCOLS ON A TURING MACHINE TAPE

To encode protocols on a Turing Machine tape, we make use of the following sets of symbols:

- Punc with symbols “⟨”,“⟩”, “(”, “)”, “{”, “}”, “[”, “]”, “_”, “:”, “·”, “;”, “→”, “↠”, “←”, “×”, “:=”, “||”, “●”,
- KeyWords with symbols “var”, “✓”, “true”, “false”, “app”, “fst”, “snd”, “of”, “ret”, “samp”, “read”, “if”, “then”, “else”, “0”, “new”, “in”, “wen”, “input-to-query”, “input-queried”, “input-not-to-query”.

We also need a finite set of de Bruijn indices in lieu of channel and variable names. To derive an upper bound on how many indices we will need, we statically count the maximum depth of variable and channel declarations in the protocol, giving us a *variable-index bound* and a *channel-index bound*, respectively.

To avoid an infinite loop, an adversary executing the absorbed protocol will need to keep track of which channels have already been queried for a value. We store this information inside the protocol in the form of an annotation: for each channel read $\mathsf{read}(c : \tau)$, we denote whether the channel $c$ has already been queried for a value, if applicable. By erasing the annotations from a query-annotated reaction or protocol, we obtain the underlying IPDL construct.

Given an ambient interpretation $[\![-]\!]$ for the signature $\Sigma$, we now show how to encode IPDL constructs as a sequence of symbols on a Turing Machine tape. For types, the encoding $\mathsf{Enc}[\tau]$ consists of the symbol “·” repeated $|\tau|$-many times. For expressions, we have the encoding below, where + denotes string concatenation. We recall that each variable name is represented as a de Bruijn index, and is in particular a natural number.

$$
\begin{aligned}
\mathsf{Enc}[v] &:= v \\
\mathsf{Enc}[\mathsf{var}(x : \tau)] &:= \text{“(”} + \text{“var”} + x + \text{“:”} + \mathsf{Enc}[\tau] + \text{“)”} \\
\mathsf{Enc}[\checkmark] &:= \text{“(”} + \text{“✓”} + \text{“)”} \\
\mathsf{Enc}[\mathsf{true}] &:= \text{“(”} + \text{“true”} + \text{“)”} \\
\mathsf{Enc}[\mathsf{false}] &:= \text{“(”} + \text{“false”} + \text{“)”} \\
\mathsf{Enc}[\mathsf{app}_{\sigma\to\tau}\ \mathsf{f}\ e] &:= \text{“(”} + \text{“app”} + \mathsf{Enc}[\sigma] + \text{“→”} + \mathsf{Enc}[\tau] + \mathsf{f} + \mathsf{Enc}[e] + \text{“)”} \\
\mathsf{Enc}[(e_1, e_2)] &:= \mathsf{Enc}[e_1] + \mathsf{Enc}[e_2] \\
\mathsf{Enc}[\mathsf{fst}_{\sigma\times\tau}\ e] &:= \text{“(”} + \text{“fst”} + \mathsf{Enc}[\sigma] + \text{“×”} + \mathsf{Enc}[\tau] + \text{“of”} + \mathsf{Enc}[e] + \text{“)”} \\
\mathsf{Enc}[\mathsf{snd}_{\sigma\times\tau}\ e] &:= \text{“(”} + \text{“snd”} + \mathsf{Enc}[\sigma] + \text{“×”} + \mathsf{Enc}[\tau] + \text{“of”} + \mathsf{Enc}[e] + \text{“)”}
\end{aligned}
$$

The encoding $\mathsf{Enc}[a]$ of an annotation is the corresponding symbol. For reactions, we have the following encoding, where we recall that each channel name is represented as a de Bruijn index, and is in particular a natural number.

$$
\begin{aligned}
\mathsf{Enc}[\mathsf{val}\ v] &:= \text{“⟨”} + v + \text{“⟩”} \\
\mathsf{Enc}[\mathsf{ret}\ e] &:= \text{“(”} + \text{“ret”} + \mathsf{Enc}[e] + \text{“)”} \\
\mathsf{Enc}[\mathsf{samp}_{\sigma\to\tau}\ \mathsf{d}\ e] &:= \text{“(”} + \text{“samp”} + \mathsf{Enc}[\sigma] + \text{“↠”} + \mathsf{Enc}[\tau] + \mathsf{d} + \mathsf{Enc}[e] + \text{“)”} \\
\mathsf{Enc}[\mathsf{read}[a](c : \tau)] &:= \text{“(”} + \text{“read”} + \mathsf{Enc}[a] + c + \text{“:”} + \mathsf{Enc}[\tau] + \text{“)”} \\
\mathsf{Enc}[\mathsf{if}\ e\ \mathsf{then}\ R_1\ \mathsf{else}\ R_2] &:= \text{“(”} + \text{“if”} + \mathsf{Enc}[e] + \text{“then”} + \mathsf{Enc}[R_1] + \text{“else”} + \mathsf{Enc}[R_2] + \text{“)”} \\
\mathsf{Enc}[x : \sigma \leftarrow R;\ S] &:= \text{“{”} + \text{“_”} + \text{“:”} + \mathsf{Enc}[\sigma] + \text{“←”} + \mathsf{Enc}[R] + \text{“;”} + \mathsf{Enc}[S] + \text{“}\}\text{”}
\end{aligned}
$$

Finally, for protocols we have the encoding below.

$$\begin{aligned}
\mathsf{Enc}[0] &\coloneqq \text{“0”} \\
\mathsf{Enc}[o := v] &\coloneqq \text{“[”} + o + \text{“:=”} + v + \text{“]”} \\
\mathsf{Enc}[o := R] &\coloneqq \text{“(”} + o + \text{“:=”} + \text{“react”} + \mathsf{Enc}[R] + \text{“)”} \\
\mathsf{Enc}[P \parallel Q] &\coloneqq \text{“(”} + \mathsf{Enc}[P] + \text{“||”} + \mathsf{Enc}[Q] + \text{“)”} \\
\mathsf{Enc}[\text{new } c : \tau \text{ in } P] &\coloneqq \text{“new”} + \text{“_”} + \text{“:”} + \mathsf{Enc}[\tau] + \text{“in”} + \mathsf{Enc}[P] + \text{“wen”}
\end{aligned}$$

To avoid having to shift the tape contents when executing IPDL protocols on a Turing Machine tape, we will make use of the white-space symbol “ ”, which we consider as distinct from the symbol *blank*. The former will be used as a placeholder so that our protocol encoding remains at a constant length throughout the execution. For this reason, we extend our notion of encoding to allow extra white-spaces around the encoding of an expression $e$ or a query-annotated reaction $R$ occurring inside a query-annotated protocol $P$.

## C LAYERED APPROXIMATE JUDGEMENTS

The layered form of our approximate judgments is shown in Figures 9 and 10.

Lemma 3. *We have* $\Delta \vdash P \approx Q : I \to O$ count $k$ ctxt $l$ *iff* $\Delta \vdash P \approx_5 Q : I \to O$ count $k$ ctxt $l$ *for any protocols* $\Delta \vdash P : I \to O$ *and* $\Delta \vdash Q : I \to O$.

## D PERFECT INDISTINGUISHABILITY: PROOF OF LEMMA 1

Lemma (Perfect indistinguishability). *For any interpretation* $[\![-]\!]$ *for* $\Sigma$, *derivation* $\Delta \vdash P = Q : I \to O$, *and adversary* Adv *for protocols* $\Delta \vdash I \to O$, *we have*

$$\left|\Pr\left[\mathsf{Adv} \overset{[\![-]\!]}{\rightleftharpoons} P = 1\right] - \Pr\left[\mathsf{Adv} \overset{[\![-]\!]}{\rightleftharpoons} Q = 1\right]\right| = 0.$$

Proof. Fix an adversary Adv as in Definition 3. By assumption, we have a proof $\Delta \vdash P = Q : I \to O$, which means we also have a proof that $\Delta' \vdash \phi^\star(P) = \phi^\star(Q) : \phi^\star(I) \to \phi^\star(O)$. The soundness theorem for strict equality of protocols applied to this proof gives us a bisimulation $\sim$ such that $1[\phi^\star(P)] \sim 1[\phi^\star(Q)]$. Now let $\sim_{\mathsf{adv}}$ be a binary relation on sub-distributions on pairs where the first element is an adversary state and the second is a protocol of type $\Delta \vdash I \to O$, defined as follows:

- $(s, \eta) \sim_{\mathsf{adv}} (s, \varepsilon)$ if $s \in \mathsf{St}$ and $\eta \sim \varepsilon$, where we use a distribution in place of a protocol to indicate the obvious lifting to sub-distributions on pairs of the the aforementioned form, and
- $1[\bot] \sim_{\mathsf{adv}} 1[\bot]$, where $\bot$ indicates that the security game between the adversary and the protocol halted without a decision Boolean.

Let $\mathcal{L}_{\sim_{\mathsf{adv}}}$ be the closure of $\sim_{\mathsf{adv}}$ under joint convex combinations. Explicitly, $\mathcal{L}_{\sim_{\mathsf{adv}}}$ is defined by

$$\left(\sum_i c_i\, \eta_i\right) \mathcal{L}_{\sim_{\mathsf{adv}}} \left(\sum_i c_i\, \varepsilon_i\right)$$

for coefficients $c_i > 0$ with $\sum_i c_i = 1$ and distributions $\eta_i \sim_{\mathsf{adv}} \varepsilon_i$. We now establish a loop invariant for the algorithm in Figure 7. Before starting the first round, the initial distributions are suitably related: by assumption, we have $1[\phi^\star(P)] \sim 1[\phi^\star(Q)]$, which means that

$$1\left[(s_\star, \phi^\star(P))\right] \mathcal{L}_{\sim_{\mathsf{adv}}} 1\left[(s_\star, \phi^\star(Q))\right]$$

as the two distributions are already related under $\sim_{\mathsf{adv}}$. Now assume that we have two sub-distributions related by $\mathcal{L}_{\sim_{\mathsf{adv}}}$. We prove that performing a single round yields sub-distributions that are again related by $\mathcal{L}_{\sim_{\mathsf{adv}}}$. It suffices to show this for the case $(s, \eta) \sim_{\mathsf{adv}} (s, \varepsilon)$, where $s \in \mathsf{St}$

$$\boxed{\Delta \vdash P \cong_0 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}$$

$$\frac{\Delta^k \vdash P^k \approx Q^k : I^k \to O^k}{\Delta^k \vdash P^k \cong_0 Q^k : I^k \to O^k \text{ ctxt } 0 \text{ for axiom \# } k}\ \textsc{axiom}$$

$$\boxed{\Delta \vdash P \cong_1 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}$$

$$\frac{\Delta \vdash P \cong_0 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}{\Delta \vdash P \cong_1 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}\ \textsc{sub}$$

$$\frac{\phi : \Delta_1 \to \Delta_2 \qquad \Delta_2 \vdash P \cong_0 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}{\Delta_1 \vdash \phi^\star(P) \cong_1 \phi^\star(Q) : \phi^\star(I) \to \phi^\star(O) \text{ ctxt } \psi \text{ for axiom \# } k}\ \textsc{embed}$$

$$\boxed{\Delta \vdash P \cong_2 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}$$

$$\frac{\Delta \vdash P \cong_1 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}{\Delta \vdash P \cong_2 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}\ \textsc{sub}$$

$$\frac{i \notin I \cup O \qquad \Delta \vdash P \cong_2 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}{\Delta \vdash P \cong_2 Q : I \cup \{i\} \to O \text{ ctxt } \psi \text{ for axiom \# } k}\ \textsc{input-unused}$$

$$\boxed{\Delta \vdash P \cong_3 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}$$

$$\frac{\Delta \vdash P \cong_2 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}{\Delta \vdash P \cong_3 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}\ \textsc{sub}$$

$$\frac{\Delta \vdash P \cong_2 P' : I \cup O_2 \to O_1 \text{ ctxt } \psi \text{ for axiom \# } k \qquad \Delta \vdash Q : I \cup O_1 \to O_2}{\Delta \vdash P \parallel Q \cong_3 P' \parallel Q : I \to O_1 \cup O_2 \text{ ctxt } \psi + \|Q\| + 3 \text{ for axiom \# } k}\ \textsc{cong-comp-left}$$

$$\boxed{\Delta \vdash P \cong_4 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}$$

$$\frac{\Delta \vdash P \cong_3 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}{\Delta \vdash P \cong_4 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}\ \textsc{sub}$$

$$\frac{\Delta, o : \tau \vdash P \cong_4 P' : I \to O \cup \{o\} \text{ ctxt } \psi \text{ for axiom \# } k}{\Delta \vdash \left(\text{new } o : \tau \text{ in } P\right) \cong_4 \left(\text{new } o : \tau \text{ in } P'\right) : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}\ \textsc{cong-new}$$

Fig. 9. Layered approximate judgements for protocols.

$$\boxed{\Delta \vdash P \cong_5 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}$$

$$\frac{\Delta \vdash P = P' : I \to O \qquad \Delta \vdash P' \cong_4 Q' : I \to O \text{ ctxt } \psi \text{ for axiom \# } k \qquad \Delta \vdash Q' = Q : I \to O}{\Delta \vdash P \cong_5 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}\ \textsc{sub}$$

$$\boxed{\Delta \vdash P \approx_5 Q : I \to O \text{ count } \xi \text{ ctxt } \psi}$$

$$\frac{\Delta \vdash P = Q : I \to O}{\Delta \vdash P \approx_5 Q : I \to O \text{ count } (i \mapsto 0) \text{ ctxt } (i \mapsto 0)}\ \textsc{strict}$$

$$\frac{\Delta \vdash P \cong_5 Q : I \to O \text{ ctxt } \psi \text{ for axiom \# } k}{\Delta \vdash P \approx_5 Q : I \to O \text{ count } \big(k \mapsto 1, i \neq k \mapsto 0\big) \text{ ctxt } \big(k \mapsto \psi, i \neq k \mapsto 0\big)}\ \textsc{approx-cong}$$

$$\frac{\Delta \vdash P_1 \approx_5 P_2 : I \to O \text{ count } \xi \text{ ctxt } \psi}{\Delta \vdash P_2 \approx_5 P_1 : I \to O \text{ count } \xi \text{ ctxt } \psi}\ \textsc{sym}$$

$$\frac{\Delta \vdash P_1 \approx_5 P_2 : I \to O \text{ count } \xi_1 \text{ ctxt } \psi_1 \qquad \Delta \vdash P_2 \approx_5 P_3 : I \to O \text{ count } \xi_2 \text{ ctxt } \psi_2}{\Delta \vdash P_1 \approx_5 P_3 : I \to O \text{ count } \big(i \mapsto \xi_1(i) + \xi_2(i)\big) \text{ ctxt } \big(i \mapsto \max(\psi_1(i), \psi_2(i))\big)}\ \textsc{trans}$$

Fig. 10. Layered approximate judgements for protocols, continued.

and $\eta \sim \varepsilon$. We first compute the distributions $\eta' := \eta\Downarrow$ and $\varepsilon' := \varepsilon\Downarrow$. By definition of $\sim$ we have $\eta' \sim \varepsilon'$. Independently, we probabilistically compute the type of interaction to perform together with a new adversary state $s'$. If no interaction has been chosen, the resulting distributions are $(s', \eta')$ and $(s', \varepsilon')$. We have

$$(s', \eta') \ \mathcal{L}_{\sim_{\text{adv}}}\ (s', \varepsilon')$$

as desired, as the two distributions are already related under $\sim_{\text{adv}}$. If the interaction is an input on channel $i$, we compute $\mathrm{O}_i(s')$ to see if in the adversary's current state $s'$ the channel $i$ carries a value. If this is not the case, the resulting distributions are $(s', \eta')$ and $(s', \varepsilon')$. Here we again have $(s', \eta') \ \mathcal{L}_{\sim_{\text{adv}}}\ (s', \varepsilon')$, as desired. On the other hand, if the channel $i$ carries a value $v$, the resulting distributions are $\big(s', \eta'[\text{read } i := \text{val } v]\big)$ and $\big(s', \varepsilon'[\text{read } i := \text{val } v]\big)$. Now because $\eta' \sim \varepsilon'$, by definition of $\sim$ we have $\eta'[\text{read } i := \text{val } v] \sim \varepsilon'[\text{read } i := \text{val } v]$. Thus we have

$$\big(s', \eta'[\text{read } i := \text{val } v]\big) \ \mathcal{L}_{\sim_{\text{adv}}}\ \big(s', \varepsilon'[\text{read } i := \text{val } v]\big)$$

as desired, as the two distributions are already related under $\sim_{\text{adv}}$. Finally, if the interaction is a query for an output channel $o$, we recall that the valuation property of the bisimulation $\sim$ allows us to jointly partition the distributions $\eta' \sim \varepsilon'$ into a joint convex combination

$$\eta' = \sum_i c_i\, \eta'_i \ \sim\ \sum_i c_i\, \varepsilon'_i = \varepsilon'$$

with $c_i > 0$ and $\sum_i c_i = 1$ such that

- the respective components $\eta'_i \sim \varepsilon'_i$ are again related, and
- $\eta'_i|_{\text{val}(o)} = v_\perp = \varepsilon'_i|_{\text{val}(o)}$ for the same $v_\perp \in \{\perp\} \cup [\![\tau]\!]$ where $o : \tau$ in $\Delta'$.

Therefore, it suffices to consider the respective components $\eta_i' \sim \varepsilon_i'$ with the same $v_\perp$. If $v_\perp$ is $\perp$, then the resulting distributions are $(s', \eta_i')$ and $(s', \varepsilon_i')$. Here we again have $(s', \eta_i') \; \mathcal{L}_{\sim_{\mathsf{adv}}} \; (s', \varepsilon_i')$, as desired. On the other hand, if $v_\perp$ is a value $v$, then the resulting distributions are $\big(\mathsf{I}_o(v, s'), \eta_i'\big)$ and $\big(\mathsf{I}_o(v, s'), \varepsilon_i'\big)$. Thus we have

$$\big(\mathsf{I}_o(v, s'), \eta_i'\big) \; \mathcal{L}_{\sim_{\mathsf{adv}}} \; \big(\mathsf{I}_o(v, s'), \varepsilon_i'\big)$$

as desired, as the two distributions are already related under $\sim_{\mathsf{adv}}$. This proves that after completing the required number of rounds, we end up with two sub-distributions related by $\mathcal{L}_{\sim_{\mathsf{adv}}}$. It is now easy to see that they induce the same sub-distribution on decision Booleans. It suffices to prove this for the case $(s, \eta) \sim_{\mathsf{adv}} (s, \varepsilon)$, where $s \in \mathrm{St}$ and $\eta \sim \varepsilon$. But the state $s$ is the same for both distributions, so the resulting distribution on decision Booleans is $1[\mathsf{D}(s)]$. This finishes the proof. □

## E SOUNDNESS OF ASYMPTOTIC EQUALITY: PROOF OF THEOREM 2

Theorem (Soundness of asymptotic equality of protocols). *For any*

- *protocol families* $\{\Delta_\lambda \vdash P_\lambda : I_\lambda \to O_\lambda\}_{\lambda \in \mathbb{N}}$ *and* $\{\Delta_\lambda \vdash Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda \in \mathbb{N}}$ *with identical typing judgments;*
- *PPT family of interpretations* $\{[\![-]\!]_\lambda\}_{\lambda \in \mathbb{N}}$ *for* $\Sigma$*; and*
- *an asymptotic theory that is sound with respect to* $\{[\![-]\!]_\lambda\}_{\lambda \in \mathbb{N}}$*;*

*we have that*

$$\vdash \{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda \in \mathbb{N}}$$

*implies*

$$\vDash \{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda \in \mathbb{N}}.$$

Proof. Given $n$ approximate axioms, the top-level asymptotic equality judgement

$$\vdash \{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda \in \mathbb{N}}$$

gives us functions

$$\xi : \mathbb{N} \to \{0, \ldots, n-1\} \to \mathbb{N}$$
$$\psi : \mathbb{N} \to \{0, \ldots, n-1\} \to \mathbb{N}^{|\Sigma_{\mathrm{t}}|+1} \to \mathbb{N}$$

such that for each $\lambda \in \mathbb{N}$, we have

$$\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda \; \mathsf{count} \; \xi_\lambda \; \mathsf{ctxt} \; \psi_\lambda \qquad (\star)$$

and for each $1 \le i \le n$, we have

$$\xi^i_{(\cdot)} = \mathsf{O}(\mathsf{poly}(\lambda))$$
$$\psi^i_{(\cdot)} = \mathsf{O}\big(\mathsf{poly}(\lambda, t_1, \ldots, t_{|\Sigma_{\mathrm{t}}|})\big)$$

In particular, there are polynomials $p^i_{\mathsf{count}}(\lambda)$ with $N^i_{\mathsf{count}} \in \mathbb{N}$ such that $\xi^i_\lambda \le p^i_{\mathsf{count}}(\lambda)$ if $\lambda \ge N^i_{\mathsf{count}}$, and polynomials $p^i_{\mathsf{ctxt}}(\lambda, t_1, \ldots, t_{|\Sigma_{\mathrm{t}}|})$ with $N^i_{\mathsf{ctxt}} \in \mathbb{N}$ such that

$$\psi^i_\lambda(t_1, \ldots, t_{|\Sigma_{\mathrm{t}}|}) \le p^i_{\mathsf{ctxt}}(\lambda, t_1, \ldots, t_{|\Sigma_{\mathrm{t}}|})$$

if $\lambda \ge N^i_{\mathsf{ctxt}}$ and $t_1, \ldots, t_{|\Sigma_{\mathrm{t}}|} \ge N^i_{\mathsf{ctxt}}$.

Since $\{[\![-]\!]_\lambda\}_{\lambda \in \mathbb{N}}$ is PPT, we have a polynomial $C_{\mathsf{sem}}(\lambda)$, a negligible function $\eta_{\mathsf{sem}}(\lambda)$, and a $N_{\mathsf{sem}} \in \mathbb{N}$ such that:

- *for all type symbols* t, $|\mathrm{t}|_\lambda \le C_{\mathsf{sem}}(\lambda)$ *if* $\lambda \ge N_{\mathsf{sem}}$,
- *for all function symbols* f, $[\![\mathrm{f}]\!]_\lambda$ *is computable by a deterministic Turing Machine* $\mathsf{TM}_\lambda$ *with symbols* 0, 1, *and both the number of states and the runtime of* $\mathsf{TM}_\lambda$ *are* $\le C_{\mathsf{sem}}(\lambda)$ *if* $\lambda \ge N_{\mathsf{sem}}$, *and*

– *for all distribution symbols* $\mathsf{d}$, $[\![\mathsf{d}]\!]_\lambda$ *is computable up to an error* $\eta_{\mathsf{sem}}(\lambda)$ *by a probabilistic Turing Machine* $\mathsf{TM}_\lambda$ *with symbols* $0, 1$*, and both the number of states and the runtime of* $\mathsf{TM}_\lambda$ *are* $\leq C_{\mathsf{sem}}(\lambda)$ *if* $\lambda \geq N_{\mathsf{sem}}$.

To prove

$$\{[\![-]\!]_\lambda\}_{\lambda\in\mathbb{N}} \vDash \{\Delta_\lambda \vdash P_\lambda \approx Q_\lambda : I_\lambda \to O_\lambda\}_{\lambda\in\mathbb{N}},$$

assume a polynomial $C_{\mathsf{adv}}(\lambda)$ and a negligible function $\eta_{\mathsf{adv}}(\lambda)$. Since the ambient asymptotic theory is sound under the ambient family of interpretations, we have the computational indistinguishability assumption

$$\{[\![-]\!]_\lambda\}_{\lambda\in\mathbb{N}} \vDash \{\Delta^i_\lambda \vdash P^i_\lambda \approx Q^i_\lambda : I^i_\lambda \to O^i_\lambda\}_{\lambda\in\mathbb{N}}.$$

Define

$$C^i_{\mathsf{ctxt}}(\lambda) \coloneqq p^i_{\mathsf{ctxt}}\big(\lambda, C_{\mathsf{sem}}(\lambda) + N^i_{\mathsf{ctxt}}, \ldots, C_{\mathsf{sem}}(\lambda) + N^i_{\mathsf{ctxt}}\big),$$

and apply the computational indistinguishability assumption above to the polynomial

$$\mathcal{P}\big(C_{\mathsf{sem}}(\lambda), C_{\mathsf{adv}}(\lambda), C^i_{\mathsf{ctxt}}(\lambda)\big)$$

and the negligible function $\max\big(\eta_{\mathsf{sem}}(\lambda), \eta_{\mathsf{adv}}(\lambda)\big)$.

This yields a negligible function $\varepsilon^i(\lambda)$ with an $N^i \in \mathbb{N}$ such that for any $\lambda \geq N^i$ and any adversary $\mathsf{Adv}^i$ for protocols $\Delta^i_\lambda \vdash I^i_\lambda \to O^i_\lambda$ under the interpretation $[\![-]\!]_\lambda$ with the property that

$$|\mathsf{Adv}^i| \leq \mathcal{P}\big(C_{\mathsf{sem}}(\lambda), C_{\mathsf{adv}}(\lambda), C^i_{\mathsf{ctxt}}(\lambda)\big)$$

and $\mathsf{err}(\mathsf{Adv}^i) \leq \max\big(\eta_{\mathsf{sem}}(\lambda), \eta_{\mathsf{adv}}(\lambda)\big)$, we have

$$\Big|\Pr\big[\mathsf{Adv} \overset{[\![-]\!]_\lambda}{\rightleftharpoons} P^i_\lambda = 1\big] - \Pr\big[\mathsf{Adv} \overset{[\![-]\!]_\lambda}{\rightleftharpoons} Q^i_\lambda = 1\big]\Big| \leq \varepsilon^i(\lambda).$$

We can now define our desired negligible function as

$$\varepsilon(\lambda) \coloneqq \sum_{i=1}^{n} \xi^i_\lambda * \big(\varepsilon^i(\lambda) + 2 * C^i_{\mathsf{ctxt}}(\lambda) * \eta_{\mathsf{sem}}(\lambda)\big)$$

The negligibility of $\varepsilon(\lambda)$ follows easily: if $\lambda \geq N^i_{\mathsf{count}}$, then

$$\xi^i_\lambda * \big(\varepsilon^i(\lambda) + 2 * C^i_{\mathsf{ctxt}}(\lambda) * \eta_{\mathsf{sem}}(\lambda)\big) \leq p^i_{\mathsf{count}}(\lambda) * \big(\varepsilon^i(\lambda) + 2 * C^i_{\mathsf{ctxt}}(\lambda) * \eta_{\mathsf{sem}}(\lambda)\big),$$

thus it suffices to show that this latter function is negligible. But this is immediate from the negligibility of $\varepsilon^i(\lambda)$, the negligibility of $\eta_{\mathsf{sem}}(\lambda)$, and the fact that $p^i_{\mathsf{count}}(\lambda)$ and $C^i_{\mathsf{ctxt}}(\lambda)$ are polynomials. Define

$$N \coloneqq \max\big(N_{\mathsf{sem}}, N^1_{\mathsf{ctxt}}, \ldots, N^n_{\mathsf{ctxt}}, N^1, \ldots, N^n\big).$$

Now assume $\lambda \geq N$ and take any adversary $\mathsf{Adv}$ for protocols $\Delta_\lambda \vdash I_\lambda \to O_\lambda$ under the interpretation $[\![-]\!]_\lambda$, such that $|\mathsf{Adv}| \leq C_{\mathsf{adv}}(\lambda)$ and $\mathsf{err}(\mathsf{Adv}) \leq \eta_{\mathsf{adv}}(\lambda)$. We aim to show that

$$\Big|\Pr\big[\mathsf{Adv} \overset{[\![-]\!]_\lambda}{\rightleftharpoons} P_\lambda = 1\big] - \Pr\big[\mathsf{Adv} \overset{[\![-]\!]_\lambda}{\rightleftharpoons} Q_\lambda = 1\big]\Big| \leq \sum_{i=1}^{n} \xi^i_\lambda * \big(\varepsilon^i(\lambda) + 2 * C^i_{\mathsf{ctxt}}(\lambda) * \eta_{\mathsf{sem}}(\lambda)\big)$$

But this is precisely the conclusion of Theorem 1 applied to the derivation ($\star$). It thus suffices to prove the hypotheses of Theorem 1. Among these, the only non-trivial assumption is

$$\psi^i_\lambda\big(|\mathsf{t}_1|, \ldots, |\mathsf{t}_{|\Sigma_\mathsf{t}|}|\big) \leq C^i_{\mathsf{ctxt}}(\lambda).$$

We show this via the following sequence of inequalities:

$$\begin{aligned}
&\psi^i_\lambda\big(|\mathsf{t}_1|_\lambda, \ldots, |\mathsf{t}_{|\Sigma_\mathsf{t}|}|_\lambda\big) \leq \\
&\psi^i_\lambda\big(C_\mathsf{sem}(\lambda), \ldots, C_\mathsf{sem}(\lambda)\big) \leq \\
&\psi^i_\lambda\big(C_\mathsf{sem}(\lambda) + N^i_\mathsf{ctxt}, \ldots, C_\mathsf{sem}(\lambda) + N^i_\mathsf{ctxt}\big) \leq \\
&p^i_\mathsf{ctxt}\big(\lambda, C_\mathsf{sem}(\lambda) + N^i_\mathsf{ctxt}, \ldots, C_\mathsf{sem}(\lambda) + N^i_\mathsf{ctxt}\big) = C^i_\mathsf{ctxt}(\lambda).
\end{aligned}$$

The first inequality follows since the function $\psi^i_\lambda : \mathbb{N}^{|\Sigma_\mathsf{t}|} \to \mathbb{N}$ is monotonically increasing in each argument and $|\mathsf{t}_1|_\lambda, \ldots, |\mathsf{t}_{|\Sigma_\mathsf{t}|}|_\lambda \leq C_\mathsf{sem}(\lambda)$ by assumption since $\lambda \geq N_\mathsf{sem}$. The second inequality is again monotonicity of $\psi^i_\lambda$, and the third follows from the definition of $p^i_\mathsf{ctxt}$ since $\lambda \geq N^i_\mathsf{ctxt}$ and $C_\mathsf{sem}(\lambda) + N^i_\mathsf{ctxt} \geq N^i_\mathsf{ctxt}$. □

## F ABSORPTION: PROOF SKETCH FOR LEMMA 2

Lemma (Absorption). *There exists a polynomial $\mathcal{P}(x, y, z) \geq y$ such that for any*

- *interpretation $[\![-]\!]$ for $\Sigma$ for which there are $C_\mathsf{sem} \in \mathbb{N}$, $\eta_\mathsf{sem} \in \mathbb{Q}_{\geq 0}$ such that*
  - *for all type symbols* t, $|\mathsf{t}| \leq C_\mathsf{sem}$,
  - *for all function symbols* f, $[\![\mathsf{f}]\!]$ *is computable by a TM with symbols* 0, 1 *such that the number of states and the runtime are* $\leq C_\mathsf{sem}$, *and*
  - *for all distribution symbols* d, $[\![\mathsf{d}]\!]$ *is computable up to error $\eta_\mathsf{sem}$ by a probabilistic TM with symbols* 0, 1 *such that the number of states and the runtime are* $\leq C_\mathsf{sem}$,
- *adversary* Adv *for protocols of type $\Delta \vdash I \to O_1 \cup O_2$ such that* $|\mathsf{Adv}| \leq C_\mathsf{adv}$ *and* $\mathsf{err}(\mathsf{Adv}) \leq \eta_\mathsf{adv}$ *for some $C_\mathsf{adv} \in \mathbb{N}$ and $\eta_\mathsf{adv} \in \mathbb{Q}_{\geq 0}$,*
- *protocol $\Delta \vdash Q : I \cup O_1 \to O_2$,*

*we have an adversary* $\mathsf{Adv}_\mathcal{R}$ *for protocols $\Delta \vdash I \cup O_2 \to O_1$ with*

$$|\mathsf{Adv}_\mathcal{R}| \leq \mathcal{P}\big(C_\mathsf{sem}, C_\mathsf{adv}, \|Q\|(|\mathsf{t}_1|, \ldots, |\mathsf{t}_{|\Sigma_\mathsf{t}|}|)\big)$$

*and* $\mathsf{err}(\mathsf{Adv}_\mathcal{R}) \leq \max(\eta_\mathsf{sem}, \eta_\mathsf{adv})$ *such that for any protocol $\Delta \vdash P : I \cup O_2 \to O_1$ we have*

$$\Big|\Pr\big[\mathsf{Adv} \overset{[\![-]\!]}{\rightleftharpoons} P \parallel Q = 1\big] - \Pr\big[\mathsf{Adv}_\mathcal{R} \overset{[\![-]\!]}{\rightleftharpoons} P = 1\big]\Big| \leq \|Q\|(|\mathsf{t}_1|, \ldots, |\mathsf{t}_{|\Sigma_\mathsf{t}|}|) * \eta_\mathsf{sem}.$$

Sketch. Let context $:= \|Q\|(|\mathsf{t}_1|, \ldots, |\mathsf{t}_{|\Sigma_\mathsf{t}|}|)$, and let $O_1^\mathsf{min}$ be the minimal set of query inputs to $Q$ from among $O_1$. In other words, $O_1^\mathsf{min}$ contains precisely those channels of $O_1$ that $Q$ reads from. The reason for replacing $O_1$ with $O_1^\mathsf{min}$ is that we do not have a bound on the size of the former, but the size of the latter is bounded by the number of occurrences of the query annotation "input-to-query", and this is in turn bounded by context. Let

$$\mathsf{Adv} := \big(\Delta', I', O', \phi, \#_\mathsf{rnd}, \#_\mathsf{tape}, \mathsf{Symb}, \mathsf{St}, s_\star, \mathsf{T}, \{\mathsf{I}_o\}_{o \in I'}, \{\mathsf{O}_i\}_{i \in O'}, \mathsf{D}\big)$$

be the adversary for protocols of type $\Delta \vdash I \to O_1 \cup O_2$. Define the reduced adversary $\mathsf{Adv}_\mathcal{R}$ for protocols of type $\Delta \vdash I \cup O_2 \to O_1$ as follows:

$$\mathsf{Adv}_\mathcal{R} := \big(\Delta', I'_\mathcal{R}, O'_\mathcal{R}, \phi, \#^\mathcal{R}_\mathsf{rnd}, \#^\mathcal{R}_\mathsf{tape}, \mathsf{Symb}^\mathcal{R}, \mathsf{St}^\mathcal{R}, s^\mathcal{R}_\star, \mathsf{T}^\mathcal{R}, \{\mathsf{I}^\mathcal{R}_o\}_{o \in I'_\mathcal{R}}, \{\mathsf{O}^\mathcal{R}_i\}_{i \in O'_\mathcal{R}}, \mathsf{D}^\mathcal{R}\big)$$

Here:

- the set of *inputs* is $I'_\mathcal{R} := (I' \cup \phi^\star(O_1)) \cup \phi^\star(O_1^\mathsf{min})$;
- the set of *outputs* is $O'_\mathcal{R} := O' \cup \phi^\star(O_2)$;
- the number of *rounds* is $\#^\mathcal{R}_\mathsf{rnd} := \#_\mathsf{rnd} * \mathsf{context} + \mathsf{context}^2 + 2 * \#_\mathsf{rnd} + \mathsf{context} + 1$;
- the number of *tapes* is $\#^\mathcal{R}_\mathsf{tape} := \#_\mathsf{tape} + 1$;

- the set of *symbols* is

$$\mathsf{Symb}^{\mathcal{R}} := \{r \mid 0 \le r \le \#_{\mathsf{rnd}}\} \bigsqcup \mathsf{ProtEncSymb} \bigsqcup \{\text{“}\rightleftharpoons\text{”}, \text{“\#”}\} \bigsqcup \mathsf{Symb},$$

  where ProtEncSymb is the disjoint union of the following sets: {“ ”}, Punc, KeyWords, the set $\Sigma_{\mathsf{f}}$ of function symbols declared in $\Sigma$, the set $\Sigma_{\mathsf{d}}$ of distribution symbols declared in $\Sigma$, the set

$$\{n \mid 0 \le n < \textit{variable-index bound}(Q)\},$$

  and the set

$$\{n \mid 0 \le n < \textit{channel-index bound}(Q)\} \bigcup \{m + n \mid m \in \phi^{\star}(I \cup O_1^{\mathsf{min}} \cup O_2) \textit{ and } 0 \le n \le \textit{channel-index bound}(Q)\};$$

- the set of *states* is

$$\mathsf{St}^{\mathcal{R}} := \{\text{“(”} + r + b + \mathsf{prot} + \text{“ }\rightleftharpoons\text{ ”} + \mathsf{adv} + \text{“)”}\},$$

  where
  - the round counter $0 \le r \le \#_{\mathsf{rnd}}$ denotes the number of rounds remaining,
  - the Boolean $b \in \{0, 1\}$ indicates whether we are processing the adversary code (0) or the absorbed protocol code (1),
  - adv $\in$ St is the original adversary code, and
  - prot is the absorbed protocol code,
- the *initial state* $s_{\star}^{\mathcal{R}}$ sets:
  - $r := \#_{\mathsf{rnd}}$,
  - $b := 1$,
  - adv $:= s_{\star}$, and
  - prot is the encoding of the protocol $Q$ with every channel read from $O_1^{\mathsf{min}}$ annotated with “input-to-query”.

The TM $\mathsf{T}^{\mathcal{R}}$ executes the encoded protocol code by first searching for a channel read annotated with “input-to-query”; if it finds one, it updates the annotation to “input-queried” and performs the query. Otherwise all inputs from $O_1^{\mathsf{min}}$ have been queried. We thus enter a second phase, where we search for a reaction that computes; if we find one, we perform the computation. Otherwise the protocol computation terminates and we set the bit $b$ to 0. Our proof yields the following polynomial:

$$P(x, y, z) := y^2 + 8yz + 15z^2 + (|\Sigma_f| + |\Sigma_d| + 1)x + 34y + 47z + (|\mathsf{Punc}| + |\mathsf{KeyWords}| + |\Sigma_f| + |\Sigma_d| + 161)$$

□

## ACKNOWLEDGMENTS

This work is partially supported by the NGI0 Core Fund (https://nlnet.nl/core), a fund established by NLnet (https://nlnet.nl) with financial support from the European Commission's Next Generation Internet programme (https://ngi.eu), under the aegis of DG Communications Networks, Content and Technology under grant agreement Nr. 101092990.