A PROBABILISTIC POLYNOMIAL-TIME CALCULUS FOR THE ANALYSIS OF CRYPTOGRAPHIC PROTOCOLS

JOHN C. MITCHELL, AJITH RAMANATHAN, ANDRE SCEDROV, AND VANESSA TEAGUE

Abstract. We prove properties of a process calculus that is designed for analyzing security protocols. Our long-term goal is to develop a form of protocol analysis, consistent with standard cryptographic assumptions, that provides a language for expressing probabilistic polynomial-time protocol steps, a specification method based on a compositional form of equivalence, and a logical basis for reasoning about equivalence.

The process calculus is a variant of CCS, with bounded replication and probabilistic polynomial-time expressions allowed in messages and boolean tests. To avoid inconsistency between security and nondeterminism, messages are scheduled probabilistically instead of nondeterministically. We prove that evaluation of any process expression halts in probabilistic polynomial time and define a form of asymptotic protocol equivalence that allows security properties to be expressed using *observational equivalence*, a standard relation from programming language theory that involves quantifying over possible environments that might interact with the protocol.

We develop a form of probabilistic bisimulation and use it to establish the soundness of an equational proof system based on observational equivalences. The proof system is illustrated by a formation derivation of the assertion, well-known in cryptography, that ElGamal encryption's semantic security is equivalent to the (computational) Decision Diffie-Hellman assumption. This example demonstrates the power of probabilistic bisimulation and equational reasoning for protocol security.

1. Introduction

There are a variety of methods used in the analysis of security protocols. The main systematic or formal approaches include specialized logics such as BAN logic [13, 19, 27], special-purpose tools designed for cryptographic protocol analysis [39], and theorem proving [55, 56] and model-checking techniques using several general purpose tools [43, 46, 52, 61, 63]. Although these approaches differ in significant ways, all reflect the same basic assumptions about the way an adversary may interact with the protocol or attempt to decrypt encrypted messages. This common model, largely derived from Dolev and Yao [26] and suggestions due to Needham and Schroeder [54], allows a protocol adversary to nondeterministically choose among

Date: December 14, 2003, and, in revised form, March 8 2004.
Key words and phrases. Process Algebra, Probabilistic Bisimulation, Security Analysis, Protocol Analysis.
The authors were supported by DoD MURI "Semantic Consistency in Information Exchange," ONR Grant N00014-97-1-0505 and OSD/ONR CIP/SW URI "Software Quality and Infrastructure Protection for Diffuse Computing," ONR Grant N00014-01-1-0795.
The third author was additionally supported by NSF Grants CCR-9800785 and CCR-0098096.
The fourth author was additionally supported by a Stanford Graduate Fellowship.

possible actions (see [19]). This convenient idealization is intended to give the adversary a chance to find an attack if one exists. In the presence of nondeterminism, however, the set of messages an adversary may use to interfere with a protocol must be restricted severely. Although Dolev-Yao-style assumptions make protocol analysis tractable, they also make it possible to "verify" protocols that are in fact susceptible to simple attacks that lie outside the adversary model (see *e.g.*, [55,62]). A further limitation of deterministic or nondeterministic settings is the inability to analyze probabilistic protocols.

In this paper we describe some technical properties of a process calculus that was proposed earlier [41, 42, 45, 51, 53] as the basis for a form of protocol analysis that is formal, yet close in foundations to the mathematical setting of modern cryptography. A recent conference paper [60] contains material excerpted from this paper. The framework relies on a language for defining communicating polynomial-time processes [51]. The reason we restrict processes to probabilistic polynomial time is so that we can reason about the security of protocols by quantifying over all "adversarial" processes definable in the language. In effect, establishing a bound on the running time of an adversary allows us to relax the simplifying assumptions on what the adversary might do. In particular, we can consider adversaries that might send randomly chosen messages, or perform sophisticated (yet probabilistic polynomial-time) computation to derive an attack from messages they overhear on the network. An important aspect of our framework is that we can analyze probabilistic as well as deterministic encryption functions and protocols. Without a probabilistic framework, it would not be possible to analyze an encryption function such as ElGamal [28], for which a single plaintext may have more than one ciphertext. A probabilistic setting is important also because the combination of nondeterminism and bit-level representation of encryption keys renders any encryption function insecure [41].

Some of the basic ideas of this work are outlined in [41], with the term language presented in [51] and further example protocols considered in [42]. Much of this paper is based on a preliminary report in [53]. The closest technical precursor is the Abadi and Gordon spi-calculus [2, 3] which uses observational equivalence and channel abstraction but does not involve probability or computational complexity bounds; subsequent related work is cited in [1], for example. Prior work on CSP and security protocols, *e.g.*, [61, 63], also uses process calculus and security specifications in the form of equivalence or related approximation orderings on processes. One important parallel effort with similar goals, the paradigm of "universally composable security", can be found in [14–18]. The relationship of this paradigm to our process calculus framework and its compositionality is discussed in [45]. The paper [45] does not deal with probabilistic bisimulation or the proof rules for our calculus. Another one based on I/O automata can be found in [7,8,57,58]. Previous literature on probabilistic process calculi includes, *e.g.*, [11, 40, 65]. However, asymptotic equivalence as used in security does not appear in any of these references. There are studies of asymptotic equivalence in the context of bisimulations though, including *e.g.*, [22, 23].

In this paper, security properties are specified as observational equivalences. Specifically, $\mathcal{P} \cong \mathcal{Q}$ means that for any context $\mathcal{C}[\ \ ]$, the behavior of process $\mathcal{C}[\mathcal{P}]$ is asymptotically computationally indistinguishable from the behavior of process $\mathcal{C}[\mathcal{Q}]$. If $\mathcal{P}$ is a protocol of interest, and $\mathcal{Q}$ is an idealized form of the process that uses private channels to guarantee authentication and secrecy, then $\mathcal{P} \cong \mathcal{Q}$ is a succinct way of asserting that $\mathcal{P}$ is secure. We have found this approach, also used in [14–18, 58], effective not only for specifying security properties of common network protocols, but also for stating common cryptographic assumptions. For this reason, we believe it is possible to prove protocol security from cryptographic assumptions using equational reasoning. The possibility is realized in this paper by proving security of El Gamal encryption from the standard Decision Diffie-Hellman assumption, and conversely.

Several advances over our previous efforts [41, 42, 45, 53] were needed to make these formal equational proofs possible. First, we have refined the operational semantics of our process calculus. Most importantly, we define protocol execution with respect to any probabilistic scheduler that runs in polynomial time and operates uniformly over certain kinds of choices (to avoid unrealistic collusion between the scheduler and a protocol attacker), and we give priority to private ("silent") actions by executing private actions simultaneously in parallel before public communication. Second, we develop a form of probabilistic bisimulation that, while not a complete characterization of asymptotic observational equivalence, gives a tractable approximation. Third, we present an equational proof system and prove its soundness using bisimulation. Finally, the material in Section 5 dealing with computational indistinguishability, semantic security, El Gamal encryption, and Decision Diffie-Hellman is entirely new.

Although our main long-term objective is to base protocol analysis on standard cryptographic assumptions, this framework may also shed new light on basic questions in cryptography. In particular, the characterization of "secure" encryption function, for use in protocols, does not appear to have been completely settled. While the definition of *semantic security* [34] appears to have been accepted, there are stronger notions such as *non-malleability* [25] that are more appropriate to protocol analysis. In a sense, the difference is that semantic security is natural for the single transmission of an encrypted message, while non-malleability accounts for vulnerabilities that may arise in more complex protocols. Our framework provides a setting for working backwards from security properties of a protocol to derive necessary properties of underlying encryption primitives. While we freely admit that much more needs to be done to produce a systematic analysis method, we believe that a foundational setting for protocol analysis that incorporates probability and complexity restrictions has much to offer in the future.

2. Preliminaries

In Section 2.1 we introduce a notion of probabilistic function tailored to our needs. Section 2.2 discusses multisets and extends the standard notion of an equivalence class over a set to an equivalence class over a multiset. Finally, the material in Section 2.3 on Turing machines recapitulates standard treatments and establishes notation and terminology.

2.1. **Probabilistic Functions.**

Definition 2.1. A *probabilistic function F from X to Y* is a function $X \times Y \to [0, 1]$ that satisfies the following two conditions:

(1) $\forall x \in X\colon \sum_{y \in Y} F(x, y) \leq 1$,
(2) For each x in X, the size of the set $\{y |\, y \in Y, F(x, y) > 0\}$ is finite.

For some $x \in X, y \in Y$, we write $\mathrm{Prob}\big[F(x) = y\big] = p$ and say F *maps x to y with probability p* just when $F(x, y) = p$.

Definition 2.2. Let $F\colon X \times Y \to [0,1]$ be a probabilistic function. We will refer to X as the *domain of F*, Y as the *codomain of F*, and

$$\bigcup_{x \in X} \{y \mid y \in Y, \mathrm{Prob}\big[F(x) = y\big] > 0\}$$

as the *range of F*.

We will say that a probabilistic function $F\colon X \times Y \to [0,1]$ is *stochastic* just when $\forall x \in X\colon \sum_{y \in Y} \mathrm{Prob}\big[F(x) = y\big] = 1$.

2.1.1. *Composition of Probabilistic Functions.*

Definition 2.3. We define the *composition* $F_2 \circ F_1\colon X \times Y \to [0,1]$ of two probabilistic functions $F_1\colon X \times Y \to [0,1]$ and $F_2\colon X \times Y \to [0,1]$ as the function satisfying the following condition:

$$\forall x \in X. \forall z \in Z\colon F(x, z) = \sum_{y \in Y} F_1(x, y) \cdot F_2(y, z)$$

Lemma 2.4. *The composition of two probabilistic functions $F_1\colon X \times Y \to [0,1]$ and $F_2\colon Y \times Z \to [0,1]$ is a probabilistic function from X to Z.*

Proof. It is easy to see that $F_2 \circ F_1$ satisfies condition 2 of Defn. 2.1. So it is sufficient to show that $F_2 \circ F_1$ satisfies the condition $\forall x \in X\colon \sum_{z \in Z} F(x, z) \leq 1$.

For any fixed $x \in X$:

$$\begin{array}{rcll} \sum_{z \in Z} F(x,z) & = & \sum_{z \in Z} \sum_{y \in Y} F_1(x,y) \cdot F_2(y,z) & \text{by Defn. 2.3} \\ & = & \sum_{y \in Y} \Big(F_1(x,y) \cdot \sum_{z \in Z} F_2(y,z) \Big) & \\ & \leq & \sum_{y \in Y} F_1(x,y) & \text{by Defn. 2.1} \\ & \leq & 1 & \text{by Defn. 2.1} \end{array}$$

Hence, composition is a probabilistic function. □

2.2. **Multisets and Quotients of Multisets.**

Definition 2.5. A *multiset* over a set X is a function $A\colon X \to \mathbb{N}$.

Note that any subset of X may be viewed, through its characteristic function, as a multiset over X. We write $x \in A$ iff $A(x) \geq 1$. For two multisets, A and B over X, we write $A \subseteq B$ iff $\forall x \in X\colon A(x) \leq B(x)$. We will use $\emptyset$ to denote the *empty multiset* defined as $\emptyset(x) = 0$ for all $x \in X$.

The *union* of two multisets over X, A and B, is given by $(A \uplus B)(x) = A(x) + B(x)$ for all $x \in X$. The *intersection* of two multisets over X, A and B, is given by $(A \Cap B)(x) = \min\{A(x), B(x)\}$ for all $x \in X$. The *difference* $A \setminus B$ of two multisets over X is given by $(A \setminus B)(x) = \max\{0, A(x) - B(x)\}$. The *cardinality* of the multiset A over X is $\sum_{x \in X} A(x)$. Finally, from a mapping $f\colon X \to Y$ and a multiset A over X, we can define the multiset f_A over Y as $f_A(y) = \sum_{\{x \in X \mid f(x) = y\}} A(x)$.

We may explicitly define a multiset A by enumerating its elements between the $\{\!|\cdots|\!\}$ brackets. For example, given an underlying set X, $\{\!|a, a, b|\!\}$ denotes the multiset A over X such that $A(a) = 2$, $A(b) = 1$, and $\forall z \in X \setminus \{x, y\}\colon A(z) = 0$.

For any set X, element $x \in X$, and equivalence relation $R \subseteq X \times X$, $[x]_R$ is the equivalence class of x with respect to R and $X/_R$ is the set of equivalence classes of X induced by R. Let $E \in X/_R$ be an equivalence class of X under the relation R. We write $\operatorname{rep} E$ for a *representative element* of E *i.e.*, any $x \in E$.

We extend the notion of an equivalence class to multisets. Let A be a multiset and X its underlying set. Let R be an equivalence relation over X. Given R and an element $x \in X$, we define the *equivalence class* $[x]_R$ *over* A as the multiset

$$[x]_R(y) = \begin{cases} A(y) & \text{if } xRy \text{, and} \\ 0 & \text{otherwise.} \end{cases}$$

That is to say, the equivalence class of x with respect to R over the multiset A is a multiset consisting of those elements $y \in X$ related to x under R, each element given the same multiplicities as in the multiset A. The *set of equivalence classes of* A *induced by* R, written $A/_R$, is the set $\{[x]_R \mid x \in A\}$.

2.3. **Probabilistic Turing Machines.** The following definitions are standard (see *e.g.*, [6]).

Definition 2.6. An *oracle Turing machine* is a Turing machine with an extra oracle tape and three extra states $q_{\text{query}}, q_{\text{yes}}$, and q_{no}. When the machine enters state q_{query} control passes to the state q_{yes} if the contents on the oracle tape are in the oracle set; otherwise, control passes to the state q_{no}. Given an oracle Turing machine M, we will write $M(\varphi, \vec{a})$ for the result of M on input $\vec{a}$ using oracle φ.

We only consider binary oracles *i.e.*, oracles that produce either a '1' or a '0' when queried. A binary oracle φ determines a set $A_\varphi = \{a \mid \varphi(a)\}$, and querying φ at a asks whether $a \in A_\varphi$.

Definition 2.7. An oracle Turing machine M runs in *oracle polynomial time* if there exists a polynomial $q(\vec{x})$ such that for all oracles φ, $M(\varphi, \vec{a})$ halts in time $q(|\vec{a}|)$ where $\vec{a} = \langle a_1, \ldots, a_k \rangle$ and $|\vec{a}| = |a_1| + \cdots + |a_k|$.

If M runs in oracle polynomial time, then $M(\vec{a})$ queries the oracle set on at most the first $q(|\vec{a}|)$ elements of the oracle set.

Definition 2.8 (Probabilistic poly-time Turing machine)**.** Let M be an oracle poly-time Turing machine. We can view M as a *probabilistic poly-time Turing machine* (ppTM) if we randomly choose an oracle from the space of oracles that can be queried in the time bound of M. More precisely, let M be an oracle machine running in time bounded by the polynomial $q(\vec{x})$. Since $M(\vec{a})$ can only query an oracle with at most $q(\vec{a})$ bits, we have a finite set $\mathcal{Q}$ of oracles on which M runs in time bounded by $q(x)$. Then, we can view M as a probabilistic poly-time Turing machine where we say that $M(\vec{a}) = b$ with probability p iff, choosing an oracle φ uniformly at random from the finite set $\mathcal{Q}$, the probability that $M(\varphi, \vec{a}) = b$ is p.

Definition 2.9 (Probabilistic poly-time computable)**.** We say that a probabilistic poly-time Turing machine M *computes* a probabilistic function F if for all inputs $\vec{a}$ and for all outputs b we have $\operatorname{Prob}\big[F(\vec{a}) = b\big] = \operatorname{Prob}\big[M(\vec{a}) = b\big]$. A probabilistic function F is *poly-time* if it is computed by a probabilistic poly-time Turing machine.

We note that every function computed by a probabilistic polynomial-time Turing machine satisfies condition 2 of the definition of probabilistic functions (Defn. 2.1) and is, therefore, a probabilistic function.

3. The Probabilistic Process Calculus

We present a probabilistic process calculus for analyzing security protocols in which protocol adversaries may be arbitrary probabilistic polynomial-time processes. The language consists of a set of *terms* that do not perform any communications, *expressions* that can communicate with other expressions, and, *channels* that are the (logical) medium through which expressions communicate which each other.

In what follows we assume a countable set of variable names Var, a countable set of channel names $Channel$, and a set of positive polynomials in one variable $Poly = \{q\colon \mathbb{N} \to \mathbb{N} | \forall a \in \mathbb{N}\colon q(a) > 0\}$. Finally, we have a distinguished constant $\underline{\text{N}}$, the security parameter, that we will discuss in Section 3.2.

3.1. Terms.

We assume the existence of a class of *basic terms* Θ for probabilistic poly-time numeric functions of arbitrary arity, and a probabilistic function $eval\colon \Theta \times \mathbb{N}^* \to \mathbb{N}$ called *basic term evaluation*, such that:

(1) If θ is a basic term with $k = arg(\theta)$ arguments, then there exists a probabilistic Turing machine M_θ with k inputs and a polynomial $q_\theta(x_1, \ldots, x_k)$ such that:
   (a) The Turing machine $M_\theta(a_1, \ldots, a_k)$ returns a with probability p iff $eval(\theta, \langle a_1, \ldots, a_k\rangle) = a$ with probability p; and,
   (b) For any choice of $a_1, \ldots, a_k$, the machine $M_\theta(a_1, \ldots, a_k)$ halts in time at most $q_\theta(|a_1|, \ldots, |a_k|)$.

(2) For each probabilistic poly-time function $f\colon \mathbb{N}^m \to \mathbb{N}$, there exists a basic term θ of m arguments such that M_θ computes f.

The first condition states that all basic terms are computable in polynomial time, while the second condition guarantees that any probabilistic poly-time function of type $\mathbb{N}^m \to \mathbb{N}$ can be expressed by some basic term. One example of such a set of terms is based on a term calculus called OSLR studied in Mitchell, Mitchell, and Scedrov [51] (based in turn on work by Bellantoni and Hofmann [9,37]). The closed OSLR terms of type $\mathbb{N}^m \to \mathbb{N}$ satisfy properties 1 and 2.

For our purposes, we simply identify the probabilistic poly-time functions and basic terms. Thus, if f is a probabilistic poly-time function, then we will also use f to refer to the basic term given by condition 2.

Definition 3.1 (Terms)**.** Letting θ range over basic terms and x range over Var, the set $Term$ of *terms* is given by the grammar:

$$T \quad ::= \quad x \mid \underline{\text{N}} \mid (\theta)\ T_1, \ldots, T_k \qquad \text{where } \theta \text{ is a basic term of } k \text{ arguments}$$

Given a term T with no free variables, we define its *reduction* inductively:

(1) The term $\underline{\text{N}}$ reduces to the value chosen for the security parameter with probability 1.
(2) The term $[a/x]x$ reduces to the value a with probability 1.
(3) The term $(\theta)\ T_1, \ldots, T_k$ is reduced by first reducing $T_1, \ldots, T_k$ yielding the values $a_1, \ldots, a_k$ and then computing $eval(\theta, \langle a_1, \ldots, a_k\rangle)$.

Given a term T we write $T(a_1, \ldots, a_k) \overset{p}{\hookrightarrow} a$ just when T reduces to a on inputs $a_1, \ldots, a_k$ with probability p. A term T is an *atom* just when $\exists a \in \mathbb{N}\colon T \equiv a$. Given a term T all of whose free variables are among $x_1, \ldots, x_k$, it is easy to construct a Turing machine M_T with k inputs such that $T(a_1, \ldots, a_k) = a$ with probability p just when $[a_1, \ldots, a_k/x_1, \ldots, x_k]T$ reduces to a with probability p.

We note that $(id)\ a \overset{1}{\hookrightarrow} a$ *i.e.*, the term $(id)\ a$ reduces to a with probability one. In future we will simply write a for the term $(id)\ a$. Additionally, we draw the reader's attention to the fact that the distinguished constant $\underline{\mathrm{N}}$ can appear as a term. Finally, since every basic term has an associated Turing machine and since the set of Turing machines are countable, $Term$ is countable.

Example 3.2. Here are some sample terms:

(1) If M is a ppTM that generates key-pairs of a given length for some fixed encryption scheme, then $\theta_M\ x$ is the term that generates key-pairs of length x.
(2) If E is a ppTM that, given a message and a public key, produces the RSA ciphertext of that message, then $\theta_E\ m\ k$ is the term that encrypts messages m under public key k.

3.2. **The Process Calculus.** *Expressions* of the probabilistic process calculus (PPC) are given by the following grammar:

$$\begin{array}{rcll}
\mathcal{P} & ::= & \oslash & \text{(termination)}\\
& & \nu_c(\mathcal{P}) & \text{(private channel)}\\
& & \mathtt{in}\langle c, x\rangle.(\mathcal{P}) & \text{(input)}\\
& & \mathtt{out}\langle c, T\rangle.(\mathcal{P}) & \text{(output)}\\
& & [T_1 = T_2].(\mathcal{P}) & \text{(match)}\\
& & (\mathcal{P} \mid \mathcal{P}) & \text{(parallel composition)}\\
& & !_{q(\underline{\mathrm{N}})}.(\mathcal{P}) & \text{(bounded replication)}
\end{array}$$

Intuitively $\oslash$ is the *empy process* that takes no further action. The private channel operator, ν, forces the channel name c bound to it to act as a private channel in the scope denoted by the enclosing parentheses. A channel name is public if it is not bound by a ν-operator. For convenience we will α-rename channel names so that they are all distinct. This is especially useful for separating public and private channel names.

In the input expression $\mathtt{in}\langle c, x\rangle.\mathcal{P}$, the input operator $\mathtt{in}\langle c, x\rangle$ binds the free variable x in $\mathcal{P}$. The expression $\mathtt{in}\langle c, x\rangle.\mathcal{P}$ waits until it receives a value a on the channel c, and then substitutes a for the free occurences of x in $\mathcal{P}$.

In contrast, the output operator in the output expression $\mathtt{out}\langle c, T\rangle.\mathcal{P}$ is not a binding operator. The output operator $\mathtt{out}\langle c, T\rangle$ simply reduces T to a value a, and then transmits a on the channel c before proceeding with $\mathcal{P}$.

In the match expression $[T_1 = T_2].(\mathcal{P})$, the match operator $[T_1 = T_2]$ acts as a guard on the expression $\mathcal{P}$. If the match terms T_1 and T_2 both reduce to the same value, then the evaluation of the expression $\mathcal{P}$ may continue. Otherwise, then entire match expression evaluates to the empty process.

We evaluate the parallel composition $\mathcal{P} \mid \mathcal{Q}$ by evaluating $\mathcal{P}$ and $\mathcal{Q}$ simultaneously. We will assume that the parallel composition operator $\mid$ associates to the left. The bounded replication operator has bound determined by the polynomial $q \in Poly$ affixed as a subscript. The expression $!_{q(\underline{\mathrm{N}})}.(\mathcal{P})$ is evaluated by rewriting it as the $q(\underline{\mathrm{N}})$-fold parallel composition

$$\overbrace{\mathcal{P} \mid \cdots \mid \mathcal{P}}^{q(\underline{\mathrm{N}})\text{ times}}$$

and evaluating the resultant expression. Finally, given expressions $\mathcal{P}$ and $\mathcal{Q}$, we write $\mathcal{P} \equiv \mathcal{Q}$ just when $\mathcal{P}$ and $\mathcal{Q}$ are syntactically identical.

An expression $\mathcal{P}$ generated by this grammar may contain the distinguished constant $\underline{\mathtt{N}}$. Substituting a value drawn from the natural numbers for $\underline{\mathtt{N}}$ gives rise to *processes*. In particular if $\mathcal{P}$ is an expression, then the process obtained by substituting i for all occurrences of $\underline{\mathtt{N}}$ in $\mathcal{P}$ is denoted $P^{\underline{\mathtt{N}}\leftarrow i}$ *i.e.*, $P^{\underline{\mathtt{N}}\leftarrow i} \equiv [i/\underline{\mathtt{N}}]\mathcal{P}$. An expression $\mathcal{P}$ can be thought to define the set of processes $\{P^{\underline{\mathtt{N}}\leftarrow i} | \, i \in \mathbb{N}\}$. If we wish to write a process without making the value of the security parameter explicit, we will just drop the superscript and write P for a process obtained from the expression $\mathcal{P}$. The security parameter can appear in three places in an expression:

(1) In terms,
(2) In the polynomials bounding the replication operator, and,
(3) In *bandwidth polynomials*. We will associate a polynomial with each channel name using the function $\sigma\colon \mathit{Channel} \to \mathit{Poly}$. We will refer to σ as the *bandwidth map*. The maximum number of bits that a channel $c \in \mathit{Channel}$ can transmit in one message is fixed by $\sigma(c)(\underline{\mathtt{N}})$.

Henceforth in this paper, unless otherwise specified, we will assume that every process written without an explicit value for the security parameter has the same value i chosen for the security parameter.

We let *Expr* be the set of process expressions and *Proc* the set of all processes (expressions without $\underline{\mathtt{N}}$).

Example 3.3. Here are some sample expressions. In each we assume that the channel bandwidths are large enough to accommodate the messages generated by the terms.

(1) We assume that we have terms `rsa-params` that generates the public values of an RSA cryptosystem parameterized by $\underline{\mathtt{N}}$ and `rand-msg` that generates a random message of length determined by $\underline{\mathtt{N}}$.

$$\mathtt{in}\langle c_1, x\rangle.\mathtt{in}\langle c_2, y\rangle.\mathtt{out}\langle d, \mathtt{rsa}(x,y)\rangle.\oslash \mid \\ \mathtt{out}\langle c_1, \mathtt{rsa\text{-}params}\rangle.\mathtt{out}\langle c_2, \mathtt{rand\text{-}msg}\rangle.\oslash$$

This expression computes the RSA ciphertext of the message y in the RSA cryptosystem determined by x.

(2) $!_{2\underline{\mathtt{N}}}.\big(\mathtt{out}\langle c, \mathtt{rand}\rangle.\oslash\big)$. The expression can potentially transmit (given a suitable number of inputs on the channel c) up to $2\underline{\mathtt{N}}$ random bits.
(3) $\mathtt{out}\langle c, \mathtt{rand}\rangle \mid !_{\underline{\mathtt{N}}-1}.\big(\mathtt{in}\langle c, x\rangle.\mathtt{out}\langle c, x \parallel \mathtt{rand}\rangle.\oslash\big)$. We can use this expression to guess a key of length $\underline{\mathtt{N}}$.

If $\mathcal{P}$ is an expression with free variables (*i.e.*, variables not bound by an input) we say that $\mathcal{P}$ is *open*; otherwise $\mathcal{P}$ is *closed*. We will denote the set of all variable-closed processes by *ClosedProc*. Let $\mathcal{P}$ be a open PPC expression and ξ a valuation of the free variables of $\mathcal{P}$ in $\mathbb{N}$. Then $\mathcal{P}(\xi)$ denotes the result of substituting, for each free variable x, the value $\xi(x)$ for all free occurrences of x in $\mathcal{P}$. We extend the notions of open, closed, and substitutions to processes in the usual way.

3.2.1. *Contexts.* A context is an expression with numbered "holes" (indicated by empty square brackets $[\ ]_k$ with $k \in \mathbb{N}$). The numerical subscripts serve to uniquely identify the holes. The notion of a context will prove very useful in reasoning about security. If we express protocols as expressions, then we can use contexts to express adversarial environments in which the protocols execute.

In what follows we will assume that the subscripts numbering holes are drawn from an index set $K = \{k_1, \ldots, k_m \,|\, k_i \in \mathbb{N}\}$. We define *contexts* inductively:

$$\begin{array}{rcll} \mathcal{C}[\ \ ]_K & ::= & \mathcal{P} & \\ & & [\ \ ]_{k_i} & (k_i \in K) \\ & & \nu_c(\mathcal{C}[\ \ ]_K) & \\ & & \mathtt{in}\langle c, x\rangle.(\mathcal{C}[\ \ ]_K) & \\ & & \mathtt{out}\langle c, T\rangle.(\mathcal{C}[\ \ ]_K) & \\ & & [T_1 = T_2].(\mathcal{C}[\ \ ]_K) & \\ & & (\mathcal{C}[\ \ ]_{K_1} \mid \mathcal{C}[\ \ ]_{K_2}) & (K_1 \cap K_2 = \emptyset, K_1 \cup K_2 \subseteq K) \\ & & !_{q(\underline{\mathrm{N}})}.(\mathcal{C}[\ \ ]_K) & \end{array}$$

This definition of contexts allows *zero-holed contexts*, which are just expressions. If $\mathcal{C}[\ \ ]_K$ is a one-holed context with the hole indexed by $k \in K$ we will simply write $\mathcal{C}[\ \ ]_k$.

It is useful to have a condition on the sets of indices ensuring that, after substituting one context into another, the holes in the resultant context are all uniquely-indexed. Let $\mathcal{C}[\ \ ]_K$ be a context with $K = \{k_1, \ldots, k_m\}$. Let $K_1, \ldots, K_{m'}$ $(m' \leq m)$ be sets of indices such that $\forall i, j \in [1..m']\colon K_i \cap K_j = \emptyset$ and $\forall i \in [1..m']\colon K \cap K_i = \emptyset$. Then $\mathcal{C}[\mathcal{C}_1[\ \ ]_{K_1}, \ldots, \mathcal{C}_{m'}[\ \ ]_{K_{m'}}]$ is the expression obtained by substituting $\mathcal{C}_i[\ \ ]_{K_i}$ for the hole indexed by $k_i \in K$.

We note that if we substitute $\mathcal{D}[\ \ ]_K$ into the hole in the context $\mathtt{in}\langle c, x\rangle.[\ \ ]_k$, the input operator $\mathtt{in}\langle c, x\rangle$ will bind all free occurrences of the variable x in $\mathcal{D}[\ \ ]_K$. Similarly, substituting $\mathcal{D}[\ \ ]_K$ into the hole in the context $\nu_c([\ \ ]_k)$ results in the binding of any occurence of channel c in $\mathcal{D}[\ \ ]_K$ to the ν-operator. Additionally, if $\mathcal{C}[\ \ ]_k$ is a zero-holed context, then $\mathcal{C}[\mathcal{D}[\ \ ]_K]_k \equiv \mathcal{C}[\ \ ]_k$ for any context $\mathcal{D}[\ \ ]_K$. As was the case for expressions, it follows from the presence of the security parameter $\underline{\mathrm{N}}$ in a context $\mathcal{C}[\ \ ]_K$ that we can view $\mathcal{C}[\ \ ]_K$ as defining the set $\{C^{\underline{\mathrm{N}} \leftarrow i}[\ \ ]_K \,|\, i \in \mathbb{N}\}$.

We now extend syntactic identity to contexts. We will write that $\mathcal{C}[\ \ ] \equiv \mathcal{D}[\ \ ]$ just when writing $\mathcal{C}[\ \ ]$ and $\mathcal{D}[\ \ ]$ without any indices yields syntactically identical expressions.

Definition 3.4. We define *Con* to be the set of all contexts. We will write *PCon* for the set of contexts without $\underline{\mathrm{N}}$. The set of one-holed contexts will be an important one and so we denote it by Con_1.

Unless specifically stated or made contextually clear, we will assume that contexts are one-holed.

Definition 3.5. Let $\mathcal{C}[\ \ ]$ and $\mathcal{D}[\ \ ]$ be contexts. Then we say that $\mathcal{D}[\ \ ]$ *is a subexpression of* $\mathcal{C}[\ \ ]$ exactly when there exists a one-holed context $\mathcal{E}[\ \ ]$ such that $\mathcal{C}[\ ] \equiv \mathcal{E}[\mathcal{D}[\ \ ]]$. We say that $\mathcal{D}$ is a *proper* subexpression of $\mathcal{C}$ if $\mathcal{E}[\ ] \not\equiv [\ ]$. In the case that $\mathcal{D}[\ \ ]$ is not a zero-holed context, we will say that $\mathcal{D}[\ \ ]$ is a *subcontext* of $\mathcal{C}[\ \ ]$. We will extend the definitions of (proper) subexpressions to (proper) *subprocesses.*

For brevity, instead of writing $\mathtt{out}\langle c, T\rangle.(\oslash)$, we will write $\mathtt{out}\langle c, T\rangle$. We will also drop parentheses whenever it does not interfere with clarity.

Definition 3.6. An expression $\mathcal{Q}$ appears *guarded* in $\mathcal{P}$ by an input (resp. output) operator if it appears in the scope of an input operator (resp. if $\mathtt{out}\langle c, T\rangle.\mathcal{C}[\mathcal{Q}]$ is a subexpression of $\mathcal{P}$ for some one-holed context $\mathcal{C}[\ \ ]$, some channel c and some term T). An expression $\mathcal{P}$ is *blocked* if each match and non-atomic term appearing in $\mathcal{P}$ either appears guarded by an input operator or appears guarded by an output operator, and *unblocked* otherwise. This definition is easily extended to expressions appearing in contexts.

Intuitively a blocked process is blocked from performing local computation (as embodied in terms) since all unguarded terms are atoms. On the other hand, an unblocked process can continue performing local computation. We model situations where a local computation requires some data via communication over a network by guarding a term representing that computation with a suitable input operator.

3.3. The Operational Semantics for PPC.

The evaluation of a process proceeds in a series of communication steps. Each communication step is probabilistically selected by a *scheduler.* A communication step takes an *input-output pair* (which is an input and an output on the same channel, neither of which are blocked) and performs the communication by substituting the output value for the bound variable in the scope of the input. This procedure is repeated until there are no input-output pairs left.

Performing a communication step on a blocked process might yield an unblocked process rather than a blocked process. For example, the blocked process $\mathtt{in}\langle c, x\rangle.[T_1(x) = T_2(x)].\emptyset \mid \mathtt{out}\langle c, 1\rangle$ has only one possible communication step: the input $\mathtt{in}\langle c, x\rangle.[T_1(x) = T_2(x)].\emptyset$ communicates with the output $\mathtt{out}\langle c, 1\rangle$. Performing this step yields the match $[T_1(1) = T_2(x)].\emptyset$ which is not blocked. So, we introduce a *reduction step* that runs an unblocked process until it becomes blocked *i.e.*, ready to communicate.

Consequently, an *evaluation step* on a process is a three-stage procedure that first performs a reduction step to obtain a blocked process, next performs *selection step* to select a particular communication step, and finally performs the chosen communication step. The *evaluation of processes* is a series of evaluation steps that terminate only when there are no more communication steps to perform.

We observe that the only sources of probabilistic behavior in process evaluation are the reduction steps and the selection steps. Probability naturally arises in the context of process reduction since the terms are meant to capture polynomially-bounded probabilistic computation: without probability it would be difficult to capture probabilistic encryption schemes like ElGamal encryption, pseudorandom number generators, *etc.* The selection step is probabilistic since scheduling is done probabilistically. The reader might wonder why the scheduler needs to be probabilistic rather than simply nondeterministic. If nondeterministic scheduling is employed, an attacker has unreasonable computational power. Consider the following process:

$$\begin{aligned}&\mathtt{out}\langle d, \mathit{Encrypted}\rangle \mid \mathtt{out}\langle c_1, 0\rangle \mid \mathtt{out}\langle c_1, 1\rangle \mid \cdots \mid \mathtt{out}\langle c_k, 0\rangle \mid \mathtt{out}\langle c_k, 1\rangle \mid \\ &\qquad \mathtt{in}\langle d, e\rangle.\mathtt{in}\langle c_1, x_1\rangle \cdots \mathtt{in}\langle c_k, x_k\rangle.\mathtt{out}\langle c, \mathit{Decrypt}(e, x_1 \parallel \cdots \parallel x_k)\rangle\end{aligned}$$

where $\mathit{Decrypt}(e, x_1 \parallel \cdots \parallel x_k)$ is a function that decrypts the encrypted message e using the k-bit key $x_1 \parallel \cdots \parallel x_k$ (which we read as the concatenation of the bits x_i). Using nondeterministic scheduling, the attacker can simply guess the key and decrypt the message. Because we wish to constrain the computational power of the adversary by restricting it to poly-time[1] probabilistic behavior, we must adopt probabilistic scheduling. We could have used nondeterministic scheduling if we had constrained the abilities of the adversary (as in the Dolev-Yao model). We do not

[1]We show in Section 6 that the attacker is limited to time polynomial in the security parameter.

do this since such a model eliminates from consideration simple attacks that lie outside the adversary model (see *e.g.*, [62]).

The following list provides an informal discussion of the behavior of each syntactic element of PPC. We will formalize these informal semantics in the next section.

Inputs: A process $\mathtt{in}\langle c, x\rangle.P$ can receive a value a of size no greater than the bandwidth $b = \sigma(c)(i)$ (recall that we assume all processes have the same value i substituted for the security parameter). The value $a \bmod 2^b - 1$ is then substituted for the variable x everywhere in P. In order to ensure that the value received by the input has size smaller than the c's bandwidth, we do not substitute a for x in P but $a \bmod (2^b - 1)$.

Outputs: A process $\mathtt{out}\langle c, a\rangle.P$ can transmit the value a on the channel c. The value a is generated by reducing some term T. Whilst reducing T we ensure we bandwidth-limit the result.

Matches: A match $[T_1 = T_2].P$ proceeds with P only when T_1 and T_2 evaluate to the same value. Otherwise, the match becomes the empty process.

ν-Operator: The ν-operator controls whether a channel is private or public. A channel c bound by ν is private in the scope of the binding. If a channel name c is bound somewhere in a process, then it cannot appear in an input or output operator outside the scope of that binding since we α-rename channel names before evaluation.

Intuitively, a private channel models communication through some privileged secret medium that cannot be seen by eavesdroppers. In terms of the calculus, an output on a private channel can only be captured by an input on the same channel if the input is also in the scope of the binding ν-operator. We will use private channels to transmit shared secrets like private keys, seed values and so on.

Parallel Composition: Processes combined using the parallel composition operator, $|$, evaluate in parallel. If a process that can perform an input communication step and another process that can perform an output communication step on the same channel are composed in parallel, then the combined process can perform a communication step where the input gets its value from the output (subject to the bandwidth restrictions on channels). Alternatively, two processes composed by $|$ may proceed independently of each other without communicating with each other.

Replication: For a particular choice of value i for the security parameter $\underline{\mathtt{N}}$, the replication $!_{q(\underline{\mathtt{N}})}.\big(\mathcal{P}\big)$ is just the $q(i)$-fold parallel composition of $P^{\underline{\mathtt{N}}\leftarrow i}$. That is to say

$$!_{q(i)}.\big(P\big) \equiv \overbrace{P \mid \cdots \mid P}^{q(i)\text{ times}}$$

Recall that we assume that parallel composition is left-associative.

We use the bandwidths associated with each channel name (using the bandwidth map σ) to ensure that no exponentially long messages can ever be transmitted. This property is crucial in obtaining the polynomial time bound on process evaluation given in Section 6.5. Consider the expression $\mathcal{P} \equiv !_{\underline{\mathtt{N}}}.\big(\mathtt{in}\langle c, x\rangle.\mathtt{out}\langle c, x^2\rangle\big) \mid \mathtt{out}\langle c, 2\rangle$. It is easy to see that $P^{\underline{\mathtt{N}}\leftarrow i}$ simply squares the value 2 i times (thanks to the replication operator). Thus, $\mathcal{P}$ generates values of length exponential in $\underline{\mathtt{N}}$. Now if the output of $\mathcal{P}$ is used as the input to some poly-time expression $\mathcal{Q}$, we will obtain an exponential-time expression since $\mathcal{Q}$ must run on exponentially long

Figure 1 Process Reduction.

$$
\begin{array}{lcl}
\mathrm{Prob}\big[\emptyset \rightarrowtail \emptyset\big] & = & 1 \\
\mathrm{Prob}\big[\mathtt{in}\langle c,x\rangle.P \rightarrowtail \mathtt{in}\langle c,x\rangle.P\big] & = & 1 \\
\mathrm{Prob}\big[\mathtt{out}\langle c,T\rangle.P \rightarrowtail \mathtt{out}\langle c,a\rangle.P\big] & = & \\
 & & \sum_{\{m \mid m \bmod (2^{\sigma(c)(i)}-1)=a\}} \mathrm{Prob}\big[T \hookrightarrow m\big] \\
\mathrm{Prob}\big[\nu_c(P) \rightarrowtail \nu_c(Q)\big] & = & \mathrm{Prob}\big[P \rightarrowtail Q\big] \\
\mathrm{Prob}\big[[T_1 = T_2].\emptyset \rightarrowtail \emptyset\big] & = & 1 \\
\mathrm{Prob}\big[[T_1 = T_2].P \rightarrowtail Q\big] & = & \\
\multicolumn{3}{r}{\left(\sum_{a\in\mathbb{N}} \mathrm{Prob}\big[T_1 \hookrightarrow a\big]\cdot \mathrm{Prob}\big[T_2 \hookrightarrow a\big]\right)\cdot \mathrm{Prob}\big[P \rightarrowtail Q\big] \qquad (P \not\equiv \emptyset)} \\
\mathrm{Prob}\big[[T_1 = T_2].P \rightarrowtail \emptyset\big] & = & \\
\multicolumn{3}{r}{\sum_{\{\langle a,b\rangle\in\mathbb{N}^2 \mid a\neq b\}} \mathrm{Prob}\big[T_1 \hookrightarrow a\big]\cdot \mathrm{Prob}\big[T_2 \hookrightarrow b\big] \qquad (P \not\equiv \emptyset)} \\
\mathrm{Prob}\big[P_1 \mid P_2 \rightarrowtail Q_1 \mid Q_2\big] & = & \mathrm{Prob}\big[P_1 \rightarrowtail Q_1\big]\cdot \mathrm{Prob}\big[P_2 \rightarrowtail Q_2\big]
\end{array}
$$

values. However, if we truncate the messages transmitted then no message can ever get exponentially long.

We now formalize the intuitive operational semantics just given. Our presentation will consist of four steps. We start by formally defining reduction. Our goal is to define the meaning of a process as the labelled transition system for that process. In order to define this "process graph", we will need to define the labels on edges in the graph (actions), as well as the probabilities that annotate those edges (which denote the probability that the action labelling the edge occurs during evaluation). So, after defining reduction, we will define actions and the probability that a particular action can be taken by a process. With these definitions in hand, we will give the operational semantics for PPC as a set of inference rules. Only then can we define the meaning of a process as a process graph. Our final step is to formalize scheduling.

In the remainder of this section, we will confine ourselves to closed processes. The extension to open processes can be achieved by considering the open process under all valuations of free variables, and the extension to expressions can be achieved by considering the expression under all values for the security parameter.

Reduction. Evaluation of processes involves the reduction of unblocked processes to blocked processes, and this is where we start. We define in Figure 1 the probability $\mathrm{Prob}\big[P \rightarrowtail Q\big]$ that P *reduces* to Q by induction on the structure of P. In this definition we assume that all processes have the same value substituted for $\underline{\mathrm{N}}$, the security parameter. We will write $P \overset{p}{\rightarrowtail} Q$ to indicate that P *reduces* to Q with probability p. If $\mathrm{Prob}\big[P \rightarrowtail Q\big] > 0$ we say that Q *is a reduct of* P. A reduct Q of P is *trivial* just when $Q \equiv P$ and *non-trivial* otherwise.

The definition omits the case of replication, $!_{q(i)}.\big(P\big)$, since we simply treat replication as $q(i)$-fold parallel composition of P. In all other cases, we have that $\mathrm{Prob}\big[P \rightarrowtail Q\big] = 0$. In Lemma 3.7 we will prove that reduction is a probabilistic function from unblocked to blocked processes and that reduction is idempotent.

The idea behind reduction is to evaluate all terms in exposed outputs and eliminates exposed matches by evaluating them. Consider what happens when we attempt to reduce the match $[T_1 = T_2].P$. The probability that we obtain some blocked process Q is equal to the product of the probability that the match succeeds and the probability that P reduces to Q. We notice, however, that the only reduct of $\oslash$ is $\oslash$. Unfortunately, we can also obtain that $\oslash$ by failing the match in the case that $P \equiv \oslash$. Hence when dealing with matches, we consider that case that $P \equiv \oslash$ separately from the case that $P \not\equiv \oslash$.

We also draw the reader's attention to the probability that $\texttt{out}\langle c, T\rangle.Q$ reduces to $\texttt{out}\langle c, a\rangle.Q$. Intuitively this probability should just be the probability that T evaluates to the value a. However, the reader might recall that we truncate values to $\sigma(c)(i)$ bits during transmission. Thus, the probability of seeing an a placed on the channel c is more properly calculated as the sum over all values m congruent to a modulo $2^{\sigma(c)(i)} - 1$ of the probability that the term T reduces to a.

Lemma 3.7. *Reduction, denoted $\rightarrowtail$, is a probabilistic function from unblocked processes to blocked processes. Furthermore, reduction is idempotent i.e., if P is a blocked process then $Prob\big[P \rightarrowtail P\big] = 1$.*

Proof. A routine induction on the structure of processes. □

Actions. We now define actions which will label edges in the process graph of a process. A communication step, as we already noted, consists of an input-output pair. However, in order to capture the behavior of a process in any arbitrary context, it will be useful to have communication where one party in the communication is part of an arbitrary context in which the process is being evaluated. Thus, we will require communications representing an input that receives a value from an arbitrary evaluation context and communications representing an output that sends a value to an arbitrary evaluation context. Thus, we will have labels, which we will call actions, indicating communications in the conventional sense as well as these partial communications where one of the participants is in the arbitrary context in which the process is being evaluated. In addition we will make use of a silent action to signify communications on private channels as well as reduction steps.

Definition 3.8. Let $\Lambda = \{\texttt{in}\langle c, a\rangle \,|\, c \in \mathit{Channel}, a \in \mathbb{N}\}$ be the set of *input actions* and $\bar{\Lambda} = \{\texttt{out}\langle c, a\rangle \,|\, \texttt{in}\langle c, a\rangle \in \Lambda\}$ the set of *output actions.* We let τ be the *silent action.* If $\alpha, \beta \in \Lambda \cup \bar{\Lambda} \cup \{\tau\}$, we will say that β is the *co-action* of α, written $\beta = \bar{\alpha}$, exactly when either

(1) $\alpha = \texttt{in}\langle c, a\rangle$ and $\beta = \texttt{out}\langle c, a\rangle$; or,
(2) $\alpha = \texttt{out}\langle c, a\rangle$ and $\beta = \texttt{in}\langle c, a\rangle$; or,
(3) $\alpha = \beta = \tau$.

Clearly, $\bar{\bar{\alpha}} = \alpha$. We let $\Lambda^1 = \Lambda \cup \bar{\Lambda}$ be the set of *partial actions* and $\Lambda^2 = \{\alpha \cdot \bar{\alpha} \,|\, \alpha \in \Lambda^1\}$ the set of *actual actions.* The set $\mathit{Act} = \Lambda^1 \cup \Lambda^2 \cup \{\tau\}$ is the set of *actions* of PPC. We will define the set of *public actions* to be the set $\mathit{Act} \setminus \{\tau\}$ *i.e.*, the set of all non-silent actions.

The precise meaning of the various actions is given by the operational semantics induced by the inference rules of Figure 2. An actual action $\alpha \cdot \beta$ is just the *simultaneous ordered occurrence* of the actions α and β. We will show in Section 4.2 that the action product is commutative with respect to the probabilistic bisimulation relation.

In the rest of this paper, we use α to range over all actions, 2^A to denote the powerset of the set A, $\mathfrak{P}(A)$ to denote the set of all multisets whose elements are drawn from A, the function $\mathit{chan}\colon \mathit{Act} \to 2^{\mathit{Channel}}$ to determine the channel(s) on which an action occurs, and, the function $\mathit{priv}\colon \mathit{Proc} \to 2^{\mathit{Channel}}$ to obtain the set of private channels in a process P.

We now define an equivalence relation over actions. Intuitively, two actions should be equivalent if they generate the same observable behavior. That is to say, two actions should be equivalent if they put the same values on the same channels. In order to formally define this equivalence relation we will need to define the *carrier* of an action α, denoted $\operatorname{car}\alpha$, which represents the underlying channel and value being transmitted by the action. We define

$$\operatorname{car}\colon \mathit{Act} \to \emptyset \cup \{\langle\text{input}, c, a\rangle, \langle\text{output}, c, a\rangle, \langle\text{actual}, c, a\rangle \,|\, c \in \mathit{Channel}, a \in \mathbb{N}\}$$

by cases:

(1) $\operatorname{car}\tau = \emptyset$,
(2) $\operatorname{car}\mathtt{in}\langle c, a\rangle = \langle\text{input}, c, a\rangle$,
(3) $\operatorname{car}\mathtt{out}\langle c, a\rangle = \langle\text{output}, c, a\rangle$, and,
(4) $\operatorname{car}(\alpha \cdot \bar{\alpha}) = \langle\text{actual}, c, a\rangle$ just when $\alpha \neq \tau$ and α is an input (resp. output) on the channel c receiving (resp. sending) the value a.

Definition 3.9. We say that α and β are *equivalent*, written $\alpha \sim \beta$, exactly when $\operatorname{car}\alpha = \operatorname{car}\beta$. Given a multiset of actions A, we say that the equivalence class $[\alpha]_\sim \in A$ iff $\exists\beta \in [\alpha]_\sim$ such that $\beta \in A$. We extend this notion of action-equivalence to multisets of actions as follows: given two multisets of actions η and θ, we say that $\eta \sim \theta \iff \eta/_\sim = \theta/_\sim$ using the definition of multiset quotient from Section 2.2.

Since $\sim$ places each action into an equivalence class determined by its observable behavior, it is easy to see that $\alpha \cdot \bar{\alpha} \sim \bar{\alpha} \cdot \alpha$. Henceforth we shall use the variables $[\alpha], [\beta], \ldots$ to range over the equivalence classes of *Act* induced by $\sim$.

Locating Contexts, Separators, and Normalization Functions. In this section we introduce some some technical material that will allow us to properly define the probabilities that will annotate edges in a process graph. The three notions we will study are: (1) locating contexts, (2) maximal c-separators, and, (3) the normalization function N. Locating contexts will prove convenient in precisely identifying the participants in a single action. Maximal c-separators will allow us to properly specify the effects of the ν-operator. We will use the normalization function to correctly compute the probabilities associated with a particular edge in a process graph.

Definition 3.10. Given specific occurrences of the sub-processes $P_1, \ldots, P_k$ of Q, we will refer to the k-holed context $C[\ \ ]$ such that $C[P_1, \ldots, P_k] \equiv Q$ as a *locating context for* $P_1, \ldots, P_k$ *in* Q. We will also say that $C[\ \ ]$ *locates* $P_1, \ldots, P_k$ *in* Q.

We will use locating contexts to exactly identify the subprocesses of a process involved a (partial or actual) communication step. We will define a silent action of a process P as either a reduction step or the simultaneous occurrence of all actual actions of P on private channels that can go simultaneously. In order to define the behavior of a silent action representing communications on private channels, we will need to locate all the inputs and outputs corresponding to that action. A c-separator for a process P is, essentially, a locating context that identifies all the actual actions on the channel c that P can simultaneously take, and we will use separators to specify the effects of the ν-operator. Naturally, there might be many ways in which a process P can simultaneously take all actual actions on the (private) channel c that can simultaneously go, and we will require one c-separator for each way. A c-separator that identifies m distinct action will be a $2m$-holed context whose holes are numbered from 1 to $2m$. The even-numbered holes will locate inputs and the odd-numbered holes will locate outputs (condition 1 below). Finally, a c-separator must identify actions that can be taken by a process. Therefore, each subprocess located by the separator must not be guarded by an input or an output (condition 2 below).

Definition 3.11. Let P be a blocked process, c a channel, $K = \{1, \dots, 2m\}$, and $C[\ ]_K$ a $2m$-holed locating context for $2m$ processes $Q_1, \dots, Q_{2m}$ in P such that

(1) For each even $i \in [1..2m]$ we have that Q_i is an input expression on the channel c and for each odd $i \in [1..2m]$ we have that Q_i is an output expression on the channel c, and,
(2) For each $i \in [1..2m]$ there does not exist a $2m$-holed context $D[\ ]$ and a one-holed context $E[\ ]$ such that either
    (a) $C[\ ]_K \equiv D[[\ ]_1, \dots, [\ ]_{l-1}, \mathtt{in}\langle c, x\rangle.E[\ ]_l, \dots, [\ ]_{2m}]$ holds for some $l \in [1..2m]$, $c \in \mathit{Channel}$ and $x \in \mathit{Var}$; or, or
    (b) $C[\ ]_K \equiv D[[\ ]_1, \dots, [\ ]_{l-1}, \mathtt{out}\langle c, T\rangle.E[\ ]_l, \dots, [\ ]_{2m}]$ holds for some $l \in [1..2m]$, $c \in \mathit{Channel}$ and $T \in \mathit{Term}$.

Then, $\mathcal{C}[\ ]_K$ is a *c-separator* for P. A $2m$-holed c-separator $\mathcal{C}[\ ]_K$ for P is *maximal* just when there does not exist a $2(m+1)$-holed c-separator $\mathcal{D}[\ ]_{K'=K\cup\{2m+1,2m+2\}}$ for P such that

$$\mathcal{D}[[\ ]_1, \dots, [\ ]_{i-1}, \mathtt{in}\langle c, x\rangle.\mathcal{Q}_i, [\ ]_{i+1}, \dots, [\ ]_{j-1}, \mathtt{out}\langle c, T\rangle.\mathcal{Q}_j, [\ ]_{j+1}, \dots, [\ ]_{2m+2}] \\ \equiv \mathcal{C}[\ ]_K$$

for some even $i \in K'$ and odd $j \in K'$ *i.e.*, $\mathcal{C}[\ ]_K$ is a c-separator that identifies one way of simultaneously taking all possible actual actions on the channel c.

Given a c-separation of P we will also assume, without any loss of generality, that the output located by the hole indexed i communicates with the input located by the hole indexed $i-1$. Thus, an $2m$-holed c-separator for P "separates" out or *identifies* a set of m actual actions on the channel c that P can simultaneously take. Given m actual actions, $\alpha_1, \dots, \alpha_m$ of P, we will say that the α_i $(1 \le i \le m)$ are *mutually non-interfering* if there exists a $2m$-holed c-separator for P that identifies the actions $\alpha_1, \dots, \alpha_m$.

We define $\mathit{Sep}_c(P)$ to be the set of all maximal c-separators of P. Let P be a blocked process with i exposed inputs on the channel c and j exposed outputs on the channel c. Then, the number of ways that P can simultaneously take a maximal set of mutually non-interfering actual actions on the channel c is

$$\begin{array}{ll} \dfrac{i!}{(i-j)!} & \text{if } j \le i, \\ \dfrac{j!}{(j-i)!} & \text{if } j > i, \text{ and,} \\ 0 & \text{if either } i = 0 \text{ or } j = 0. \end{array}$$

If either i or j is zero, then no c-separation exists and $\mathit{Sep}_c(P) = \emptyset$. Since $\mathit{Sep}_c(P)$ is the set of all maximal c-separators of P, this computation yields the size of $\mathit{Sep}_c(P)$ (denoted $|\mathit{Sep}_c(P)|$).

Now we turn to the normalizing function N. The normalization factor of a process P with respect to the action α counts the number of distinct ways that P

can take an action of type α. When evaluating a process P, we will pick a particular action of type α uniformly at random from the set of all actions of type α that the process P can possibly take. In Figure 2, we use the normalization factor to ensure that we combine probability distributions on actions correctly when we combine processes using $|$ or apply a ν-operator.

Definition 3.12. Let *BlockedProc* be the set of blocked processes. We define the *normalizing function* $N\colon \mathit{BlockedProc} \times \mathit{Act} \to \mathbb{R}$ by induction:

(1) $N(\emptyset, \alpha) = 0$,

(2) $$N(\nu_c(P), \alpha) = \begin{cases} N(P, \alpha) & \text{if } \forall a \in \mathbb{N}\colon \mathtt{in}\langle c, a\rangle, \mathtt{out}\langle c, a\rangle \notin \mathrm{car}\, \alpha \\ & \text{and } \alpha \nsim \tau, \\ N(P, \tau) \cdot |\mathit{Sep}_c(P)| & \text{if } \alpha \sim \tau \text{ and } N(P, \tau) \neq 0, \\ |\mathit{Sep}_c(P)| & \text{if } \alpha \sim \tau \text{ and } N(P, \tau) = 0, \text{ and,} \\ 0 & \text{otherwise.} \end{cases}$$

(3) $$N(\mathtt{in}\langle c, x\rangle.P, \alpha) = \begin{cases} 1 & \text{if } \exists a \in \mathbb{N}\colon \alpha \sim \mathtt{in}\langle c, a\rangle, \\ 0 & \text{otherwise.} \end{cases}$$

(4) $$N(\mathtt{out}\langle c, a\rangle.P, \alpha) = \begin{cases} 1 & \text{if } \alpha \sim \mathtt{out}\langle c, a\rangle, \\ 0 & \text{otherwise.} \end{cases}$$

(5) $$N(P_1 \mid P_2, \alpha) = \begin{cases} N(P_1, \alpha) + N(P_2, \alpha) + & \\ \quad + \sum_{\{\beta \cdot \bar{\beta} \sim \alpha\}} N(P_1, \beta) \cdot N(P_2, \bar{\beta}) & \text{if } \alpha \nsim \tau, \\ N(P_1, \tau) \cdot N(P_2, \tau) & \text{otherwise.} \end{cases}$$

(6) $$N(!_{q(i)}.(P), \alpha) = N(\overbrace{P \mid \cdots \mid P}^{q(i) \text{ times}}, \alpha).$$

Lemma 3.13. *The normalization function N is well-defined.*

Proof. By induction on the structure of processes. □

Probabilistic Transitions and Process Graphs. In this section we give the semantics of PPC processes as a(n evaluation) graph induced by a set of inference rules. We will write

$$P \xrightarrow{\alpha[p]} Q$$

iff it can be obtained from the inference rules of Figure 2. We refer to $P \xrightarrow{\alpha[p]} Q$ as a *transition* and its intuitive meaning is that if P takes the action *labelled* α, then with probability p it will become Q. The first group of axioms deal with inputs and outputs. In Axioms (I) and (O), we stipulate that $a \in [0..2^{\sigma(c)(i)} - 1]$ since values placed on a channel are truncated by the channel's bandwidth. These two axioms allow the evaluation of input and output processes.

We draw the reader's attention to the fact that if a process is unblocked then it can only take a reduction action: in Axioms (CL), (CR), (SL), (SR), (C), (S), and (N1) we assume that $P_1 \mid P_2$ is blocked. Thus the only actions available to an unblocked process are the reduction actions specified by Axiom (R) which states that an unblocked process can take a silent transition to one of its non-trivial reducts.

Axioms (SL), (SR), and (S) characterize the behavior of silent actions under the parallel composition. Essentially, if P and Q can take silent actions, then $P \mid Q$ *simultaneously* takes a silent action of P and one of Q. This behavior is meant to

Figure 2 The Operational Semantics of PPC.

$$\mathtt{in}\langle c, x\rangle.P \xrightarrow{\mathtt{in}\langle c,a\rangle[1]} [a/x]P \tag{I}$$

$$\mathtt{out}\langle c, a\rangle.P \xrightarrow{\mathtt{out}\langle c,a\rangle[1]} P \tag{O}$$

$$\frac{P \text{ unblocked}}{P \xrightarrow{\tau[\mathrm{Prob}[P \text{ reduces to } Q]]} Q} \tag{R}$$

$$\frac{\begin{array}{c}P_1 \xrightarrow{\alpha[p]} Q_1, P_1 \mid P_2 \text{ blocked}, \alpha \nsim \tau,\\ P_2 \text{ has no silent actions}\end{array}}{P_1 \mid P_2 \xrightarrow{\alpha\left[\frac{1}{N(P_1 \mid P_2, \alpha)}\right]} Q_1 \mid P_2} \tag{CL}$$

$$\frac{\begin{array}{c}P_2 \xrightarrow{\alpha[p]} Q_2, P_1 \mid P_2 \text{ blocked}, \alpha \nsim \tau,\\ P_1 \text{ has no silent actions}\end{array}}{P_1 \mid P_2 \xrightarrow{\alpha\left[\frac{1}{N(P_1 \mid P_2, \alpha)}\right]} P_1 \mid Q_2} \tag{CR}$$

$$\frac{P_1 \xrightarrow{\alpha[p]} Q_1, P_2 \xrightarrow{\bar{\alpha}[q]} Q_2, P_1 \mid P_2 \text{ blocked}, \alpha \nsim \tau}{P_1 \mid P_2 \xrightarrow{\alpha\cdot\bar{\alpha}\left[\frac{1}{N(P_1 \mid P_2, \alpha\cdot\bar{\alpha})}\right]} Q_1 \mid Q_2} \tag{C}$$

$$\frac{P_1 \xrightarrow{\tau[p]} Q_1, P_2 \text{ has no silent actions}, P_1 \mid P_2 \text{ blocked}}{P_1 \mid P_2 \xrightarrow{\tau[p]} Q_1 \mid P_2} \tag{SL}$$

$$\frac{P_1 \text{ has no silent actions}, P_2 \xrightarrow{\tau[p]} Q_2, P_1 \mid P_2 \text{ blocked}}{P_1 \mid P_2 \xrightarrow{\tau[p]} P_1 \mid Q_2} \tag{SR}$$

$$\frac{P_1 \xrightarrow{\tau[p]} Q_1, P_2 \xrightarrow{\tau[q]} Q_2, P_1 \mid P_2 \text{ blocked}}{P_1 \mid P_2 \xrightarrow{\tau[pq]} Q_1 \mid Q_2} \tag{S}$$

$$\frac{\begin{array}{c}C[\ \ ] \in \mathit{Sep}_c(P), C[R_1, \ldots, R_k] \equiv P, P \text{ blocked},\\ \exists D[\ \ ] \in \mathit{PCon}\colon \text{either } \mathrm{Prob}\big[C[\oslash, \ldots, \oslash] \xrightarrow{\tau} D[\oslash, \ldots, \oslash]\big] > 0 \text{ or } C[\ \ ] \equiv D[\ \ ],\\ \forall i \in [1..k].i \text{ odd}\colon \mathrm{Prob}\big[C[R_1, \ldots, R_i, R_{i+1}, \ldots, R_k] \xrightarrow{\mathtt{in}\langle c,a\rangle\cdot\mathtt{out}\langle c,a\rangle}\\ \qquad C[R_1, \ldots, R_{i-1}, S_i, S_{i+1}, R_{i+2}, \ldots, R_k]\big] > 0\end{array}}{\nu_c(C[R_1, \ldots, R_k]) \xrightarrow{\tau\left[\frac{1}{N(\nu_c(P), \tau)}\right]} \nu_c(D[S_1, \ldots, S_k])} \tag{N1}$$

$$\frac{P \text{ blocked}, \mathit{Sep}_c P = \emptyset, P \xrightarrow{\tau[p]} Q}{\nu_c(P) \xrightarrow{\tau[p]} \nu_c(Q)} \tag{N2}$$

$$\frac{P \xrightarrow{\alpha[p]} Q, \alpha \nsim \tau, \nexists a \in \mathbb{N}\colon \mathtt{in}\langle c, a\rangle, \mathtt{out}\langle c, a\rangle \in \mathrm{car}\,\alpha}{\nu_c(P) \xrightarrow{\alpha[p]} \nu_c(Q)} \tag{N3}$$

embody the idea that a silent action is somehow internal to a process. Thus, when two processes are composed with $|$, the two processes can take their silent (internal) actions simultaneously. Axiom (N1) states that binding the channel c in the process P to the ν-operator allows the process $\nu_c(P)$ to take a silent action representing the simultaneous execution of a maximal set of non-interfering actual actions of P on the channel c and some silent action of P. Axiom (N2) states that if P has no actual actions on the channel c, then binding c to the ν has no effect on silent actions. Axiom (N3) states that the ν-operator has no effect on the probability distribution for an action that is neither silent nor on the channel c.

The operational semantics for PPC (as given by the rules of Figure 2) induce a mapping φ from *ClosedProc* to a domain of probabilistic labelled transition systems which we can extend to a mapping χ from *Proc* to a domain of sets of probabilistic labelled transition systems. We can further extend χ to a mapping ψ from *Expr* to a domain of sets of sets of probabilistic labelled transition systems. More concretely, the operational semantics for PPC gives the labelled transition system of a process as a(n evaluation) graph where each node is labelled by a process and each directed edge $\langle u, v\rangle$ represents a transition from the process labelling u to the process labelling v and is labelled by the pair consisting of the action labelling t and the probability associated with t. To give the semantics of an open process P we consider all valuations of the free variables of P. Similarly to give the semantics of an expression $\mathcal{P}$, we consider all processes obtained by substituting in $\mathcal{P}$ a value for the security parameter. We defer the details until after Defn. 3.16.

An examination of the rules given in Figure 2 reveals that it is possible for a node in the process graph of P to have several outgoing edges labelled with the same action-probability pair. Consider the process graph of $\mathtt{out}\langle c, a\rangle \mid \mathtt{out}\langle c, a\rangle$:

$$\begin{tikzcd}
 & \mathtt{out}\langle c, a\rangle \mid \mathtt{out}\langle c, a\rangle \arrow[dl, "{\langle \mathtt{out}\langle c,a\rangle, \frac{1}{2}\rangle}"'] \arrow[dr, "{\langle \mathtt{out}\langle c,a\rangle, \frac{1}{2}\rangle}"] & \\
\oslash \mid \mathtt{out}\langle c, a\rangle \arrow[d, "{\langle \mathtt{out}\langle c,a\rangle, 1\rangle}"'] & & \mathtt{out}\langle c, a\rangle \mid \oslash \arrow[d, "{\langle \mathtt{out}\langle c,a\rangle, 1\rangle}"] \\
\oslash \mid \oslash & & \oslash \mid \oslash
\end{tikzcd}$$

Often we will want to sum probabilities over the outgoing edges of a node in order to calculate the total probability that a process does something. In the example above, for instance, we might wish to know the cumulative probability that $\mathtt{out}\langle c, a\rangle \mid \mathtt{out}\langle c, a\rangle$ takes an action $\mathtt{out}\langle c, a\rangle$—in this case that value is 1. In order to ensure that we calculate these sums correctly, it will be important to distinguish between transitions with the same labels. In order to do so, we will identify transitions with *indices* such that no two outgoing transitions of a process have the same index.

It now remains for us to discuss the indices themselves. We need to find a set of indices that ensures that no two transitions leaving a process has the same index. We can use a locating context to identify the input and/or output expressions involved in an action *i.e.*, we can simply index a transition labelled by α with the locating context that identifies the inputs and outputs that communicate via the action α.[2] In order to accommodate reduction actions, we will use zero-holed

[2]Many locating contexts are of no use in identifying communication steps. Consider, for example, the locating context $[T_1 = T_2].\mathcal{C}[\ \ ]$. No communication located by this locating context can

contexts to "locate" the participant (the entire process) involved in a reduction action. We shall refer to the set of locating contexts as the *index set* and denote it by $Index$. Using locating contexts to index our transitions also confers the benefit that we can single out the subprocesses of a process that are communicating. Clearly $Index \subseteq PCon$. Using these indices, the process graph of $\mathtt{out}\langle c, a\rangle \mid \mathtt{out}\langle c, a\rangle$ becomes:

$$\begin{tikzcd}
 & \mathtt{out}\langle c, a\rangle \mid \mathtt{out}\langle c, a\rangle \arrow[dl, "{\langle \mathtt{out}\langle c,a\rangle, \frac{1}{2}, [\ \ ] \mid \mathtt{out}\langle c,a\rangle\rangle}"'] \arrow[dr, "{\langle \mathtt{out}\langle c,a\rangle, \frac{1}{2}, \mathtt{out}\langle c,a\rangle \mid [\ \ ]\rangle}"] & \\
\oslash \mid \mathtt{out}\langle c, a\rangle \arrow[d, "{\langle \mathtt{out}\langle c,a\rangle, 1, \oslash \mid [\ \ ]\rangle}"'] & & \mathtt{out}\langle c, a\rangle \mid \oslash \arrow[d, "{\langle \mathtt{out}\langle c,a\rangle, 1, [\ \ ] \mid \oslash\rangle}"] \\
\oslash \mid \oslash & & \oslash \mid \oslash
\end{tikzcd}$$

We will write $P \xrightarrow{\alpha[p]}_j Q$ to denote a transition $P \xrightarrow{\alpha[p]} Q$ that we can index with $j \in Index$. The following lemma states that sums over the probabilities of transitions leaving a process are well-defined *i.e.*, for each type of action they are less than 1.

Lemma 3.14.

$$\forall P \in Proc. \forall \alpha \in Act\colon \sum_{\beta \sim \alpha, j \in Index} Prob\big[P \xrightarrow{\beta}_j Q\big] \leq 1$$

Proof. There are two cases to distinguish:

(1) P is unblocked. Then the only action available to P is a reduction step. From the definition of reduction (see Defn. 3.3) it is easy to see that $\sum_{j \in J} \mathrm{Prob}\big[P \xrightarrow{\tau}_j Q\big] \leq 1$ since reduction is a probabilistic function from unblocked processes to blocked processes.

(2) P is blocked. Our inference rules guarantee that only a finite number of transitions leaving P are labelled by $\beta \sim \alpha$. In particular, only $N(P, \alpha)$ of the transitions leaving P are labelled by $\beta \sim \alpha$. Each of these transitions has probability $1/N(P, \alpha)$ whence the desired result follows.

Since the case analysis is exhaustive, the desired result follows. □

Remark 3.15. Our indices are chosen so that no two identically-labelled transitions leaving a process P have the same index. Consequently, whenever we use some set $A \subseteq Act$ to sum over probabilities of transitions, we will make use of indices and Lemma 3.14 to guarantee that the sum is well-defined. However, since we require indices purely for this technical reason, whenever we can safely do so, we will not annotate transitions with indices.

An examination of the definition of the normalization function together with the inference rules of Figure 2 shows that $\mathrm{Prob}\big[P \xrightarrow{\alpha} Q\big] \cdot N(P, \alpha) = 1$ holds whenever P is a blocked process. This is because $N(P, \alpha)$ counts the number of ways that you can take an α-transition in P and the semantics given for PPC ensure that the

possibly be taken, since the outermost match needs to be reduced first. However, for every valid communication it is easy to see that there must exist a locating context identifying the participants in that communication step. Hence, we can use (a subset of all) locating contexts to identify valid communications.

action α of a process P is chosen according to the uniform distribution on α-actions available to P.

We now define the process graph of a process P.

Definition 3.16. A *probabilistic transition system* or *process graph* is a 3-tuple $\langle R, T, I\rangle$ with

(1) R a set of states,
(2) $T \subseteq R \times \mathit{Act} \times [0,1] \times \mathit{Index} \times R$ a set of transitions, such that
  (a) $\big(\langle s,\alpha,p,j,t\rangle \in T \wedge \langle s,\beta,q,j,r\rangle \in T\big) \implies (\alpha=\beta)\wedge(p=q)\wedge(t=r)$
  (b) $\forall s \in R. \forall \alpha \in \mathit{Act}\colon \Big(\sum_{\{\langle s,\beta,p,j,t\rangle\in T \mid \beta\sim\alpha\}} p\Big) \le 1$
  and,
(3) $I \in R$ the initial state.

The first requirement on T says that every outgoing transition of a state must have different indices. The second one says that, for each state s and for each action α, the sum of the probabilities over all the outgoing transitions β with $\beta\sim\alpha$ does not exceed 1. We will write $P \xrightarrow{[\alpha]\,[p]} Q$ iff

$$\sum_{j\in \mathit{Index}, \beta\sim\alpha} \mathrm{Prob}\big[P \xrightarrow{\beta}_j Q\big] = p$$

We will say that Q *is α-reachable from* P exactly when $\mathrm{Prob}\big[P \xrightarrow{[\alpha]} Q\big] > 0$. It is clear from the definition of process graphs that a process can have several process graphs. For example, given a process graph G for P we can construct another process graph G' for P by adding a node u labelled by a process Q that is not reachable from P by any path. We remove this ambiguity by defining an isomorphism on process graphs and then identifying all isomorphic graphs.

We will consider superfluous all parts of a transition system not reachable from the root, and we will identify states labelled with the same process. Thus, letting $\mathfrak{T}$ be the set of all transitions, an isomorphism between two transition systems $\langle R, T, I\rangle$ and $\langle R', T', I'\rangle$ is a bijective mapping $f\colon \mathfrak{T}\cup \mathit{Proc} \to \mathfrak{T}\cup\mathit{Proc}$ between their states and transitions satisfying:

(1) $f(s,\alpha,p,j,t) = (f(s),\alpha,p,k,f(t))$ where j and k may be different indices[3],
(2) $f(s) = s'$ with $s'\in R'$, and,
(3) $f(I) = I'$.

We will identify isomorphic transition systems. So, writing Proc^i for the set of all processes having i as the value of the security parameter, $\varphi(P^{\underline{\mathtt{N}}\leftarrow i})$ for $P^{\underline{\mathtt{N}}\leftarrow i}\in\mathit{Proc}$ is defined to be the transition system $\langle \mathit{Proc}^i, T, P^{\underline{\mathtt{N}}\leftarrow i}\rangle$ with

$$T = \{\langle Q^{\underline{\mathtt{N}}\leftarrow i}, \alpha, p, j, R^{\underline{\mathtt{N}}\leftarrow i}\rangle \mid Q^{\underline{\mathtt{N}}\leftarrow i} \xrightarrow{\alpha[p>0]}_j R^{\underline{\mathtt{N}}\leftarrow i}\}$$

It is easy to see that the transition system for $P^{\underline{\mathtt{N}}\leftarrow i}$ is a directed acyclic graph.

Let $\mathbb{G}$ be the domain of transition systems or process graphs. We can extend $\varphi\colon \mathit{ClosedProc}\to\mathbb{G}$ to an interpretation $\chi\colon \mathit{Proc}\to 2^{\mathbb{G}}$ of all PPC processes by extending φ to open processes. To do so, let P be an open PPC process and let ξ denote a valuation of the free variables of P in $\mathbb{N}$. Then, denoting by $P(\xi)$ the result for all free variables x of substituting $\xi(x)$ for x in P, we can define $\chi(P)$ as the

[3]The arguments of remark 3.15 can be used to demonstrate that the second condition on T in the definition of probabilistic transition systems—Defn. 3.16—is never violated.

set $\{\varphi(P(\xi)) \mid \xi$ is a valuation of $FV(P)\}$. To extend $\chi\colon Proc \to 2^{\mathbb{G}}$ to an interpretation $\psi\colon Expr \to 2^{2^{\mathbb{G}}}$ of expressions we define $\psi(\mathcal{P})$ as the set $\{\chi(P^{\underline{\mathbb{N}} \leftarrow i}) \mid i \in \mathbb{N}\}$.

Finally, we will define the *transition set* of P, denoted $Trans(P)$, as the set

$$\{[\alpha]_{\sim} \mid \exists Q \in Proc\colon \operatorname{Prob}\big[P \xrightarrow{\alpha} Q\big] > 0\}$$

of equivalence classes of actions labelling arrows leaving the node labelled by P in $\varphi(P)$. We note that the transition set of P is finite since an examination of the rules of Figure 2 reveals that any node in the process graph of P has finite outdegree.

Schedulers. Our goal i is to provide a process-calculus-based treatment of security protocols and security properties. To do so, we will model attackers as contexts within which the protocol, modelled as an expression, executes. However, in the real world, the network over which a protocol is running might be under the control of the adversary. We model this situation by placing the network (*i.e.*, the order in which messages are sent) under the control of a scheduler which is then made part of the definition of an adversary *i.e.*, formally, an adversary is a pair consisting of a context and a scheduler. Since we do not want to fix a particular scheduler in our operational semantics, we defined the process graph of a process without specifying an particular schedule. As a consequence, the process graph of a process is not probabilistic in that the sum over the probabilities of transitions leaving a process can be greater than one. We apply a scheduler to a process graph to get probabilistically well-defined behavior: the scheduler selects a type of action, and then a particular transition labelled by an action of that type is chosen uniformly at random.

Typically, the analysis of a security protocol assumes that the adversary has total control over the scheduling of messages. In particular, the adversary schedules particular messages. However, in our setting, the scheduler only controls the scheduling of *types* of messages rather than the messages themselves. If the scheduler had direct control over the scheduling of messages, then we would be able to distinguish processes in an unreasonable way. For example, a scheduler that always schedules the leftmost message would be able to distinguish $P \mid Q$ from $Q \mid P$. That is to say, a scheduler that has total control over the scheduling of messages themselves can make scheduling-decisions on the basis of information derived from quirks of the syntax of PPC (such as a well-defined notion of leftmost) rather than relying just on information having to do with the structure of the protocol's communications. By having the scheduler schedule solely on the basis of the types of actions, we blind the scheduler to all information derived from the syntax of PPC (as opposed to the network behavior of the protocol).

While restricting scheduling to types of messages is indeed a restriction (since in principle the adversary cannot assign an arbitrary distribution to the actions available to a process), it is not a significant problem. If each type of action in the set of types of actions available to a process contains exactly one action, then the scheduler essentially picks individual actions from the set of actions according to some arbitrary distribution. We believe that any security protocol can be systematically rewritten so that at each step of the evaluation, the set of types provided to the scheduler consists of singletons. We can do so by adding time-stamps to messages, unique message identifiers and so on to ensure that at no point can a process take two identically-labelled transitions. In addition, scheduling over types of actions rather than particular actions conforms to the intuition that two (indexed) transitions generating the same observable should be considered equivalent since we only care about the observables seen and not how the observables were generated.

Let us now define schedulers formally. Let $2^{Act/\sim}$ be the set of all subsets of equivalence classes of *Act* induced by $\sim$.

Definition 3.17. A *scheduler* $S\colon 2^{Act/\sim} \to Act/_\sim$ is a stochastic probabilistic function from subsets of $Act/_\sim$ to elements of $Act/_\sim$ that satisfies the condition

$$\forall A \subseteq Act/_\sim \colon [\tau]_\sim \in A \implies \big(\text{Prob}\big[S(A) = [\tau]_\sim\big] = 1\big)$$

We define *Sched* as the set of all schedulers.

Given a set of equivalence classes of actions under $\sim$, a scheduler picks one of those equivalence classes. The stochastic property guarantees that the scheduler always picks some equivalence class out of its input. Thus the stochastic property functions as a "progress" condition. We note that the time bound on the running time of a scheduler is polynomial in the size of the set of equivalence classes of actions given to it as input.

We can classify schedulers based on their behavior on $A/_\sim$. Thus, for example, the *uniform scheduler* U is the scheduler satisfying

$$\forall A \subseteq Act/_\sim . \forall [\alpha]_\sim, [\beta]_\sim \in A \colon \text{Prob}\big[U(A) = [\alpha]_\sim\big] = \text{Prob}\big[U(A) = [\beta]_\sim\big]$$

We can also define special classes of process expressions based on how schedulers act on them. An important such class is the class of *scheduler-insensitive* process expressions. Let $\mathcal{P}$ be a process expression such that for all choices i of security parameter, $P^{\underline{\mathrm{N}}\leftarrow i}$ is a process for which at each stage of evaluation all possible actions are of the same type. Since schedulers are stochastic, the choice of scheduler becomes irrelevant.

3.4. **Probabilistic Bisimulation.** In this section we adapt weak bisimulation [50] to a in a probabilistic model involving a probabilistic (rather than nondeterministic) scheduler. We will call this *weak probabilistic bisimulation* or just *probabilistic bisimulation* from here on. We will present probabilistic bisimulation as an equivalence relation over *Proc*. The development follows the presentation given in van Glabbeek, Smolka, and Steffen [65] which studies various approaches to probabilistic bisimulation and presents an elegant and economical treatment of probabilistic bisimulation.

We will refer to a sequence of actions leading from the process P to the process Q as a *path from P to Q*. We will refer to a sequence of silent actions terminated by an α-step as an *α-path*. An α-path must be of length at least 1 whenever α is public; on the other hand, if α is silent the path can have zero length. We will call zero length paths *empty paths* and τ-paths *silent paths*. Let $\alpha_1, \cdots, \alpha_{k-1}$ with

$$R_1 \xrightarrow{p_1[\alpha_1]}_{j_1} R_2 \xrightarrow{p_2[\alpha_2]}_{j_2} \ldots \xrightarrow{p_{k-2}[\alpha_{k-2}]}_{j_{k-2}} R_{k-1} \xrightarrow{p_{k-1}[\alpha_{k-1}]}_{j_{k-1}} R_k$$

be a path from some process R_1 to some process R_k. Given a scheduler S, we will say that this path is *achieved under S with probability* p give by

$$p = \prod_{j=1}^{k-1} \big(\text{Prob}\big[S(Trans(R_j)) = [\alpha_j]_\sim\big] \cdot p_j\big)$$

We say that the sequence $\pi = \langle j_1, \ldots, j_{k-1} \rangle$ of indices *supports* $\alpha_1, \ldots, \alpha_{k-1}$. In fact we will denote paths by their supports since a sequence of indices $\langle j_1, \ldots, j_m \rangle$ uniquely defines a path provided the source node of the transition indexed by j_{i+1} is the destination node of transition indexed by j_i for $1 \leq i < m$. We will write $\text{Prob}\big[\langle j_1, \ldots, j_k \rangle_S\big] = p$ just when that the path supported by $\langle j_1, \ldots, j_k \rangle$ is achieved under scheduler S with probability p. In order to denote a set of processes, we will decorate letters as in $\mathfrak{R}$. We will say that an α-path from P to a process $Q \in \mathfrak{R}$ supported by $\langle j_1, \ldots, j_k \rangle$ is *minimal with respect to* $\mathfrak{R}$ just when there does not exist a shorter α-path from P to another process in $\mathfrak{R}$ supported by $\langle j_1, \ldots, j_{k'} \rangle$ for some $k' < k$. That is, an α-path π is minimal with respect to a set of process $\mathfrak{R}$ just when no initial path π' of π is an α-path into $\mathfrak{R}$.

Definition 3.18. We define $\mathit{Paths}(P, \alpha, \mathfrak{R})$ as the set of α-paths from P to some process in $\mathfrak{R}$ that are minimal with respect to $\mathfrak{R}$.

We note that the minimality condition in the definition of $\mathit{Paths}(P, \alpha, Q)$ only affects silent paths since every α-path in trivially minimal (an α-path cannot have more than one non-silent transition).

Definition 3.19. We define a *cumulative probability distribution function* (cPDF) $\mu\colon \mathit{Proc} \times \mathit{Act} \times 2^{\mathit{Proc}} \times \mathit{Sched} \to [0, 1]$ by

$$\mu(P, \alpha, \mathfrak{R}, S) = \begin{cases} 1 & \text{if } \alpha \sim \tau \text{ and } \mathit{Paths}(P, \tau, \mathit{Proc}) = \emptyset, \\ \displaystyle\sum_{\pi \in \mathit{Paths}(P, \alpha, \mathfrak{R})} \text{Prob}\big[\pi_S\big] & \text{otherwise.} \end{cases}$$

Given a scheduler S, the sum $\mu(P, \alpha, \mathfrak{R}, S)$ measures the total probability that a process can take an α-path scheduled by S to reach a process in the set $\mathfrak{R}$. The first case in the definition of a cPDF essentially add reflexive silent loops to all processes with no silent actions. These silent actions are not actually available to the process and hence can't be scheduled but they are technically important for the proof of the congruence theorem of Section 3.5.

Lemma 3.20.

$$\forall P \in \mathit{Proc}. \forall \alpha \in \mathit{Act}. \forall \mathfrak{R} \subseteq \mathit{Proc}. \forall S \in \mathit{Sched}\colon \mu(P, \alpha, \mathfrak{R}, S) \leq 1$$

Proof Sketch. The proof is a straightforward induction on the length of α-paths whose details are left to Appendix D. A key fact to remember in the proof is that we only consider τ-paths of minimal length. This avoids the problem of a path contributing multiple terms to the cumulative probability. For example, consider a process P with the process graph

$$P \xrightarrow{\tau[1]} Q \xrightarrow{\tau[1]} R \xrightarrow{\alpha \nsim \tau[1]} \oslash$$

By not restricting to minimal silent paths, $\mu(P, \tau, [P]_{\simeq}, S) = 2$ since $P \simeq Q \simeq R$. However, restricting to minimal paths sidesteps this problem. $\square$

For an equivalence relation R over Proc we write $\mathit{Proc}/_R$ to denote the set of equivalence classes of Proc induced by R, and $[P]_R$ to denote the equivalence class (with respect to R) of which P is a member.

We are now in a position to define bisimulation equivalence. Our presentation will follow [50,65]. We start by characterizing the conditions under which a relation is a bisimulation. We will then define a particular bisimulation, $\simeq$, as the union over all bisimulations. Finally, we will show that $\simeq$ defined as a union of bisimulations is the largest bisimulation, thereby defining bisimulation equivalence. The advantage of this presentation is that in order to prove that two processes are bisimilar, we need only provide a bisimulation that establishes the desired bisimulation equivalence.

The following definition provides a test of whether a given equivalence relation is a bisimulation. Intuitively, an equivalence relation R passes the test if, two processes P and Q are related by R iff they have the same probability of passing into any equivalence class induced by R by taking the same type of action.

Definition 3.21. An equivalence relation $R \subseteq \mathit{ClosedProc} \times \mathit{ClosedProc}$ is a *weak probabilistic bisimulation*, or *bisimulation*, just when $(P, Q) \in R$ implies that

$$\forall \mathfrak{U} \in \mathit{ClosedProc}/_R . \forall S \in \mathit{Sched}. \forall \alpha \in \mathit{Act}\colon \mu(P, \alpha, \mathfrak{U}, S) = \mu(Q, \alpha, \mathfrak{U}, S)$$

Two processes P and Q are *bisimulation equivalent* (denoted $P \simeq Q$) if there exists a bisimulation R such that $(P, Q) \in R$. It immediately follows that $\simeq = \bigcup\{R |\, R \text{ is a bisimulation}\}$. We extend bisimulation to all processes by stipulating that $P, Q \in \mathit{Proc}$ are bisimulation equivalent iff for all substitutions ξ of values for their free variables, $P(\xi) \simeq Q(\xi)$. We extend bisimulation equivalence to expressions by stipulating that $\mathcal{P}, \mathcal{Q} \in \mathit{Expr}$ are bisimulation equivalent iff $\forall i \in \mathbb{N}\colon P^{\underline{\mathrm{N}} \leftarrow i} \simeq Q^{\underline{\mathrm{N}} \leftarrow i}$.

We note that Defn. 3.21 is a weakening of the intuitive definition given previously since we replace "iff" with "implies". Clearly, any relation satisfying the intuitive definition satisfies Defn. 3.21. It remains for us to show that Defn. 3.21 satisfies the intuitive definition. It is traditional to use a fixed-point iteration technique [50] to establish this kind of result.

Definition 3.22. Let $\mathcal{F}$ be a function over subsets of $\mathit{ClosedProc} \times \mathit{ClosedProc}$ such that if $R \subseteq \mathit{ClosedProc} \times \mathit{ClosedProc}$ then $(P, Q) \in \mathcal{F}(R)$ iff

$$\forall \mathfrak{R} \in \mathit{Proc}/_R . \forall S \in \mathit{Sched}. \forall \alpha \in \mathit{Act}\colon \mu(P, \alpha, \mathfrak{R}, S) = \mu(Q, \alpha, \mathfrak{R}, S)$$

We can extend $\mathcal{F}$ to all processes and then to expressions by using the same method we used to extend bisimulations to all processes and expressions.

Lemma 3.23. *For $\mathcal{F}$ and binary relations R, R_1, and, R_2 over Proc, we have*

1. *$\mathcal{F}$ is monotonic i.e., $R_1 \subseteq R_2 \implies \mathcal{F}(R_1) \subseteq \mathcal{F}(R_2)$, and,*
2. *R is a bisimulation iff $R \subseteq F(R)$.*

Proof. (1) follows immediately from Defn. 3.22. (2) is just a reformulation of the definition of bisimulation (Defn. 3.21) with '$\subseteq$' taking the place of 'implies'. □

We call R a *pre-fixed-point* of $\mathcal{F}$ if $R \subseteq \mathcal{F}(R)$. If $R = \mathcal{F}(R)$ then we call R a *fixed-point* of $\mathcal{F}$. Bisimulation equivalences are exactly the pre-fixed-points of $\mathcal{F}$; it is easy to show $\simeq$ is the largest pre-fixed-point of $\mathcal{F}$.

Lemma 3.24. *Bisimulation equivalence is the largest fixed point of $\mathcal{F}$.*

Proof. Since $\simeq$ is a bisimulation, $\simeq \subseteq \mathcal{F}(\simeq)$. Since $\mathcal{F}$ is monotonic, we have that $\mathcal{F}(\simeq) \subseteq \mathcal{F}(\mathcal{F}(\simeq))$ *i.e.*, $\mathcal{F}(\simeq)$ is a pre-fixed-point of $\mathcal{F}$. Since $\simeq$ is the largest pre-fixed-point of $\mathcal{F}$, $\mathcal{F}(\simeq) \subseteq\ \simeq$ whence $\simeq = \mathcal{F}(\simeq)$. Since $\simeq$ is the largest pre-fixed-point of $\mathcal{F}$, it must be the largest fixed-point of $\mathcal{F}$. □

It follows that $\simeq$ is the largest bisimulation equivalence over *Proc*. It will also be useful to establish that bisimulation equivalence is an equivalence relation. We will adapt the method used by van Glabbeek, Smolka, and Steffen [65] in establishing that nondeterministic bisimulation was an equivalence relation to show that probabilistic bisimulation is also an equivalence relation.

Lemma 3.25. *If B_k $(k \in K)$ is a collection of bisimulations, then their reflexive, transitive closure $(\bigcup_k B_k)^*$ is a bisimulation.*

Proof. Each of the relations B_k are symmetric, so $(\bigcup_k B_k)^*$ is also symmetric. Whence $(\bigcup_k B_k)^*$ is also an equivalence relation. Suppose $(P, Q) \in (\bigcup_k B_k)^*$. Then there are n processes $P_0, \dots, P_n$ for a certain $n \in \mathbb{N}$ such that $P = P_0$, $Q = P_n$ and for $j \in \{1, \dots, n\}, (P_{j-1}, P_j) \in B_i$ for certain $i \in K$. Suppose $\mathfrak{U} \in Proc/_{(\bigcup_k B_k)^*}$ and $\alpha \in PubAct$. Let $1 \leq j \leq n$ and $(P_{j-1}, P_j) \in B_i$. Since $\mathfrak{U}$ is the union of several equivalence classes $\mathfrak{T} \in Proc/_{R_k}$ and for each $\mathfrak{T}$ we have $\forall S \in Sched\colon \mu(P_{j-1}, \alpha, \mathfrak{T}, S) = \mu(P_j, \alpha, \mathfrak{T}, S)$, it follows that $\forall S \in Sched\colon \mu(P_{j-1}, \alpha, \mathfrak{U}, S) = \mu(P_j, \alpha, \mathfrak{U}, S)$. Since this holds for all $j \in \{1, \dots, n\}$ we conclude that $\forall S \in Sched\colon \mu(P, \alpha, \mathfrak{U}, S) = \mu(Q, \alpha, \mathfrak{U}, S)$. Whence $(\bigcup_k B_k)^*$ is a bisimulation. $\square$

As an immediate corollary of the Lemma 3.25 we conclude that $\simeq$ is an equivalence relation.

Corollary 3.26. *Bisimulation equivalence is an equivalence relation over Proc.*

We remind the reader that a process determines its own behavior given that a particular action type is chosen. In particular, given that P takes a particular action of type α it will reach some α-reachable Q with probability uniformly chosen over the number of α-actions P can take. However the process does not define a distribution on the types of actions to take *i.e.*, the process evaluates in the presence of someone who chooses which type of action to next perform by pushing, as it were, buttons. This is precisely the notion of a *reactive bisimulation* [40, 49, 59] *i.e.*, a process reacts to the environment as embodied in the button-pusher. In our setting the person pressing buttons to choose the next action to perform is an explicitly probabilistic scheduler as opposed to the nondeterministic scheduler seen commonly in the literature.

3.5. **Bisimulation Equivalence is a Congruence.** In this section, we prove a congruence theorem inspired by Milner [50]. The proof adapts the approach used of van Glabbeek, Smolka, and Steffen [65] as this simplifies the proof considerably. Essentially, we reason in terms of μ (the cPDF) rather than in terms of the underlying transitions.

In order to prove the main theorem of the this section, we will make use of the the following two lemmas which are proved in Appendices A and B respectively.

Lemma 3.27. *Let P_1, P_2 be processes with $P_1 \simeq P_2$ and Q be another process. Then, $\forall \mathfrak{R} \in Proc/_{\simeq}. \forall S \in Sched. \forall \alpha \in Act$:*

(1) $\mu(P_1 \mid Q, \alpha, \mathfrak{R}, S) = \mu(P_2 \mid Q, \alpha, \mathfrak{R}, S)$, *and,*
(2) $\mu(Q \mid P_1, \alpha, \mathfrak{R}, S) = \mu(Q \mid P_2, \alpha, \mathfrak{R}, S)$.

Lemma 3.28. *Let P, Q be processes such that $P \simeq Q$. Then,*

$$\forall \mathfrak{R} \in Proc/_{\simeq}. \forall S \in Sched. \forall \alpha \in Act\colon \mu(\nu_c(P), \alpha, \mathfrak{R}, S) = \mu(\nu_c(Q), \alpha, \mathfrak{R}, S)$$

Main Theorem 3.29.

$$\forall P, Q \in Proc.\forall C[\ \ ] \in Con_1 \colon P \simeq Q \implies C[P] \simeq C[Q]$$

Proof. We start by noting that any variable in $C[P]$ either occurs bound in P, free in P but bound in $C[\ \ ]$, or free even in $C[P]$. Since we defined bisimulation equivalence on open processes $C[P]$, $C[Q]$ by considering $C[P]$, $C[Q]$ under all possible valuations, we can eliminate from consideration variables that appear free in $C[P]$, $C[P]$ as well as variables that appear free in just $C[\ \ ]$.

We will write $C[\ \ ] \in \underline{Con_1}$ to denote that $C[\ \ ] \in Con_1$ and $C[P]$ and $C[Q]$ are variable-closed processes. So, in order to establish the desired congruence property, it is enough to show that $B = \{\langle C[P], C[Q]\rangle \mid P \simeq Q, C[\ \ ] \in \underline{Con_1}\}$ is a bisimulation *i.e.*, we need to show that $\forall P, Q \in Proc$ with $P \simeq Q$,

$$\forall C[\ \ ] \in \underline{Con_1}.\forall \mathfrak{R} \in Proc/_B.\forall S \in Sched.\forall \alpha \in Act\colon$$
$$\mu(C[P], \alpha, \mathfrak{R}, S) = \mu(C[Q], \alpha, \mathfrak{R}, S) \quad (1)$$

Thus far, what we have done is to simplify our problem by exploiting the definition of bisimulation over variable-open processes as bisimulation over all valuations of those variable-open processes. We can now proceed with the proof proper.

We proceed by an induction on the maximum of the number of free variables in P and Q. The basis occurs when neither P nor Q have any free variables. The proof for this case closely follows the proof of the inductive hypothesis. The only difference is when we consider a context whose top-level operator is an input. In that case, the action of the input is trivial since P and Q have no free variables for the context to bind. Consequently, we leave the details of the proof of the basis to the reader and we just assume as our *first inductive hypothesis* that (1) holds for pairs $\langle N, O\rangle \in Proc$ with at most m free variables.

To establish the inductive hypothesis (when P, Q have at most $m+1$ free variables), we need only show one direction of (1) (by substituting $\leq$ for $=$) since the other direction ($\geq$) follows by a similar argument. We will write $Paths^n(P, \alpha, \mathfrak{R})$ just when the support of a path in $Paths(P, \alpha, \mathfrak{R})$ can be derived by a proof tree of height at most n from the inference rules of Figure 2. We then define $\mu^n \colon Proc \times Act \times 2^{Proc} \times Sched \to [0, 1]$ as

$$\mu^n(P, \alpha, \mathfrak{R}, S) = \sum_{\pi \in Paths^n(P,\alpha,\mathfrak{R})} \text{Prob}\big[\pi_S\big]$$

We adopt the convention that $\mu^0(P, \alpha, \mathfrak{R}, S) = 0$. The well-definededness of μ^n follows from the well-definededness of μ (see Lemma 3.20). Intuitively μ^n measures the cumulative probability that P can take a sequence, scheduled by S, of at most $n-1$ silent actions followed by an α-step in order to reach some set of processes $\mathfrak{R}$. Let us write $Proc_k$ for the set of all processes with at most k free variables. Since $\mu(P, \alpha, \mathfrak{R}, S) = \lim_{n\to\infty} \mu^n(P, \alpha, \mathfrak{R}, S)$ we need just show, via an induction on n, that for all $n \geq 0$ we have

$$\forall P, Q \in Proc_{m+1}.\forall C[\ \ ] \in \underline{Con_1}.\forall \mathfrak{R} \in Proc/_B.\forall S \in Sched.\forall \alpha \in Act\colon$$
$$\mu^n(C[P], \alpha, \mathfrak{R}, S) \leq \mu(C[Q], \alpha, \mathfrak{R}, S) \quad (2)$$

For our *second inductive hypothesis*, we may assume (2) for some $n \geq 0$ since the basis (when $n = 0$) is trivial. In proving (2) for $n+1$ we pick an arbitrary scheduler S and undertake a case analysis depending on the topmost operator of $C[\ \ ]$.

(1) $C[\ ] \equiv [\ ]$. We show that for all $\mathfrak{R} \in \mathit{Proc}/_B$, $S \in \mathit{Sched}$, and $\alpha \in \mathit{Act}$

$$\mu^{n+1}(P, \alpha, \mathfrak{R}, S) \leq \mu(Q, \alpha, \mathfrak{R}, S) \tag{3}$$

Now $\simeq \subseteq B$ (from the definition of B'). Thus, $\mathfrak{R}$ is the disjoint union of one or more $\mathfrak{T} \in \mathit{Proc}/_{\simeq}$, and we need only prove (3) for these $\mathfrak{T}$ rather than the whole of $\mathfrak{R}$. Then, we can immediately conclude (from $P \simeq Q$) that $\mu^{n+1}(P, \alpha, \mathfrak{R}, S) \leq \mu(P, \alpha, \mathfrak{R}, S) = \mu(Q, \alpha, \mathfrak{R}, S)$.

(2) $C[\ ] \equiv \mathtt{in}\langle c, x\rangle.D[\ ]$. We show that for all $\mathfrak{R} \in \mathit{Proc}/_B$, $S \in \mathit{Sched}$, and $\alpha \in \mathit{Act}$

$$\mu^{n+1}(\mathtt{in}\langle c, x\rangle.D[P], \alpha, \mathfrak{R}, S) \leq \mu(\mathtt{in}\langle c, x\rangle.D[Q], \alpha, \mathfrak{R}, S) \tag{4}$$

Clearly, $\mathit{Trans}(\mathtt{in}\langle c, x\rangle.D[P]) \sim \mathit{Trans}(\mathtt{in}\langle c, x\rangle.D[Q])$ (the only actions available to either process are input actions on the channel c). Thus if, for all $a \in [0..2^{\sigma(c)(i)} - 1]$, we have that $\alpha \nsim \mathtt{in}\langle c, a\rangle$ then, trivially,

$$\mu(\mathtt{in}\langle c, x\rangle.D[P], \alpha, \mathfrak{R}, S) = (\mathtt{in}\langle c, x\rangle.D[Q], \alpha, \mathfrak{R}, S) = 0$$

and the desired result is obtained.

Now, we assume that $\alpha \sim \mathtt{in}\langle c, a\rangle$ for some $a \in [0..2^{\sigma(c)(i)} - 1]$. Since $\forall O \in \mathit{Proc}\colon \alpha, \beta \in \mathit{Trans}(\mathtt{in}\langle c, x\rangle.D[O]) \implies \alpha \nsim \beta$, it follows that

$$\mu^{n+1}(\mathtt{in}\langle c, x\rangle.D[P], \alpha, \mathfrak{R}, S) = \mathrm{Prob}\big[S(\mathit{Trans}(\mathtt{in}\langle c, x\rangle.D[P])) = [\alpha]_{\sim}\big]$$

The action $\mathtt{in}\langle c, a\rangle$ ($a \in [0..2^{\sigma(c)(i)} - 1]$) causes $\mathtt{in}\langle c, x\rangle.D[P]$ to evaluate to $[a/x]D[P]$. By the second inductive hypothesis we have that $D[P] \simeq D[Q]$ and by the first inductive hypothesis we have $[a/x]D[P] \simeq [a/x]D[P]$. Then from $\mathit{Trans}(\mathtt{in}\langle c, x\rangle.D[P]) \sim \mathit{Trans}(\mathtt{in}\langle c, x\rangle.D[Q])$ we can show

$$\mu^{n+1}(\mathtt{in}\langle c, x\rangle.D[P], \alpha, \mathfrak{R}, S) = \mu^{n+1}(\mathtt{in}\langle c, x\rangle.D[Q], \alpha, \mathfrak{R}, S)$$

We finish this case by noting that $\mu(\mathtt{in}\langle c, x\rangle.D[Q], \alpha, \mathfrak{R}, S)$ is the limit of $\mu^{n+1}(\mathtt{in}\langle c, x\rangle.D[Q], \alpha, \mathfrak{R}, S)$. In establishing the basis we use the same argument except we note that the action $\mathtt{in}\langle c, a\rangle$ ($a \in [0..2^{\sigma(c)(i)} - 1]$) causes $\mathtt{in}\langle c, x\rangle.D[P]$ to evaluate to $D[P]$ since, in the basis, P has no free variables.

(3) $C[\ ] \equiv \mathtt{out}\langle c, T\rangle.D[\ ]$. We show that for all $\mathfrak{R} \in \mathit{Proc}/_B$ and $\alpha \in \mathit{Act}$

$$\mu^{n+1}(\mathtt{out}\langle c, T\rangle.D[P], \alpha, \mathfrak{R}, S) \leq \mu(\mathtt{out}\langle c, T\rangle.D[Q], \alpha, \mathfrak{R}, S) \tag{5}$$

We will write $\mathrm{Prob}\big[T \hookrightarrow [a]_m\big]$ for $\sum_{\{n \mid n \bmod m \equiv a\}} \mathrm{Prob}\big[T \hookrightarrow n\big]$. Noting that silent actions are hidden, it is easy to see that for each $\alpha \sim \mathtt{out}\langle c, a\rangle$

$$\mu^{n+1}(\mathtt{out}\langle c, T\rangle.D[P], \alpha, \mathfrak{R}, S) = \mathrm{Prob}\big[T \hookrightarrow [a]_m\big]$$

For any $\alpha \nsim \mathtt{out}\langle c, a\rangle$ we also have that $\mu^{n+1}(\mathtt{out}\langle c, T\rangle.D[P], \alpha, \mathfrak{R}, S) = 0$. We do not have a term denoting the probability of scheduling the action since if T is not a value, then the only action available is a reduction step, and if T is a value a, then the only action available is the output action $\mathtt{out}\langle c, a\rangle$. In either case, the set of equivalence classes of $\mathit{Trans}(\mathtt{out}\langle c, T\rangle.D[P])$ induced by $\sim$ is a singleton whence every scheduler picks the single element with probability one.

We finally note that our assumption that $C[P]$, $C[Q]$ be closed processes implies that $D[P]$, $D[Q]$ are closed processes. We then finish in the manner of case 2 by using the inductive hypotheses.

(4) $C[\ ] \equiv [T_1 = T_2].D[\ ]$. We show that for all $\mathfrak{R} \in \mathit{Proc}/_B$ and $\alpha \in \mathit{Act}$

$$\mu^{n+1}([T_1 = T_2].D[P], \alpha, \mathfrak{R}, S) \leq \mu([T_1 = T_2].D[Q], \alpha, \mathfrak{R}, S) \tag{6}$$

It is easy to check that every action that $[T_1 = T_2].D[P]$ (resp. $[T_1 = T_2].D[Q]$) can take is guarded by the reduction step needed to eliminate the outermost exposed match. Thus we see that

$$\mu^{n+1}([T_1 = T_2].D[P], \alpha, \mathfrak{R}, S) = \\ \big(\mathrm{Prob}\big[T_1 \hookrightarrow a\big] \cdot \mathrm{Prob}\big[T_2 \hookrightarrow a\big]\big) \cdot \mu^n(D[P], \alpha, \mathfrak{R}, S)$$

Again, we note that $D[P]$ and $D[Q]$ are closed processes since we assume that $C[P]$ and $C[Q]$ are closed processes. Thus, we can apply our two inductive hypotheses to show that

$$\mu^{n+1}([T_1 = T_2].D[P], \alpha, \mathfrak{R}, S) \leq \mu^{n+1}([T_1 = T_2].D[Q], \alpha, \mathfrak{R}, S)$$

and then finish as in case 2.

(5) $C[\ ] \equiv \nu_c(D[\ ])$. We show that for all $\mathfrak{R} \in \mathit{Proc}/_B$ and $\alpha \in \mathit{Act}$

$$\mu^{n+1}(\nu_c(D[P]), \alpha, \mathfrak{R}, S) \leq \mu(\nu_c(D[Q]), \alpha, \mathfrak{R}, S) \tag{7}$$

A ν-operator restricting channel c makes all actions $\mathtt{in}\langle c, a\rangle \cdot \mathtt{out}\langle c, a\rangle$ or $\mathtt{in}\langle c, a\rangle \cdot \mathtt{out}\langle c, a\rangle$ in its scope invisible and prevents any other transitions on the channel c from occurring. We can therefore distinguish two cases.

(a) $c \in \mathit{chan}(\alpha)$. Then

$$\mu^{n+1}(\nu_c(D[P]), \alpha, \mathfrak{R}, S) = \mu^{n+1}(\nu_c(D[Q]), \alpha, \mathfrak{R}, S) = 0$$

We then finish as in case 2.

(b) $c \notin \mathit{chan}(\alpha)$. Then we use Lemma 3.28 and the inductive hypotheses.

(6) $C[\ ] \equiv O \mid D[\ ]$ or $C[\ ] \equiv D[\ ] \mid O$. We just show $C[\ ] \equiv O \mid D[\ ]$ since the other case follows by a symmetric argument. We show that for all $\mathfrak{R} \in \mathit{Proc}/_B$ and $\alpha \in \mathit{Act}$

$$\mu^{n+1}(O \mid D[P], \alpha, \mathfrak{R}, S) \leq \mu(O \mid D[Q], \alpha, \mathfrak{R}, S) \tag{8}$$

We just use apply Lemma 3.27 using the inductive hypotheses.

Since the case analysis is exhaustive, the desired result is established. □

4. Observational Equivalence

We wish to be able to say that two closed expressions are equivalent if and only if they behave in the same way in the presence of any adversary. Formally speaking, we will identify the space of adversaries with the space of pairs of contexts and schedulers and then stipulate that two closed expressions are observationally equivalent under some scheduler if the two closed expressions produce "approximately" the same observable behavior when messages are scheduled according to the chosen scheduler. By "approximate" we will mean "asymptotically close in the security parameter". So, even though an adversary may distinguish two closed expressions for a small value of the security parameter (via, say, a brute force search over keys), once we increase the security parameter sufficiently, that adversary will get defeated.

4.1. **The Observational Equivalence Relation.** We will first define the notion of an observable and then the probability that an expression generates a particular observable. Then we will define observable equivalence and show that it is a congruence. Next we relate probabilistic bisimulation equivalence and observational equivalence.

Definition 4.1. An *observable* o is a pair $\langle c, a\rangle \in \mathit{Channel} \times \mathbb{N}$. Let Obs be the set of all observables.

Let us denote the set of actual actions with $\mathit{Act}^{\times}$. Let $P \in \mathit{Proc}$ be a blocked process and let $o = \langle c, a\rangle$ be an observable. We will say that P *generates the observable* o *under scheduler* S, written $P \rightsquigarrow_S o$, just when an action equivalent to $\mathtt{in}\langle c, a\rangle \cdot \mathtt{out}\langle c, a\rangle$ is selected by S during the course of the evaluation of P. Since evaluation is a sequence of actual actions, we need not consider partial actions when we compute the probability that P generates the observable o under the scheduler S. We remind the reader that partial actions were used solely to prove that $\simeq$ was a congruence.

Remark 4.2. Since $\simeq$ is an equivalence relation over Proc, we know that $\simeq$ breaks up Proc into several distinct equivalence classes that cover the whole of Proc. Thus,

$$\sum_{\mathfrak{R} \in \mathit{Proc}/_{\simeq}} \mu(P, \alpha, \mathfrak{R}, S) = \sum_{\mathfrak{R} \in \mathit{Proc}/_{\simeq}} \sum_{Q \in \mathfrak{R}} \mu(P, \alpha, \{Q\}, S) = \sum_{R \in \mathit{Proc}} \mu(P, \alpha, \{R\}, S)$$

Consequently, we will move between the two formulations as convenience dictates.

The probability that P *generates the observable* o *under the scheduler* S is the probability that P can take a $\mathtt{in}\langle c, a\rangle \cdot \mathtt{out}\langle c, a\rangle$-path, under S, plus the probability that P can, under S, take an α-path (with α not equivalent to $\mathtt{in}\langle c, a\rangle \cdot \mathtt{out}\langle c, a\rangle$) to some process R times the probability that R generates o under S. Formally, $\mathrm{Prob}\big[P \rightsquigarrow_S o\big]$ is given by

$$\sum_{R \in \mathit{Proc}} \mu(P, \mathtt{in}\langle c, a\rangle \cdot \mathtt{out}\langle c, a\rangle, \{R\}, S) + \sum_{\substack{\{\beta \in \mathit{Act}^{\times} \mid \beta \not\sim \mathtt{in}\langle c,a\rangle \cdot \mathtt{out}\langle c,a\rangle\} \\ R \in \mathit{Proc}}} \mu(P, \beta, \{R\}, S) \cdot \mathrm{Prob}\big[R \rightsquigarrow_S o\big]$$

Lemma 4.3. *Prob*$\big[P \rightsquigarrow_S o\big] \leq 1$.

Proof. We leave the proof to Appendix D. □

Definition 4.4. We define *PSched*, the set of *perceptible schedulers*, as the set of poly-time schedulers that only schedule private actions and actual actions.

We view process evaluation as a series of actual actions whose types are picked by a scheduler. Since we are interested only in actual actions, we need only consider perceptible schedulers when defining observational equivalence.

Definition 4.5 (observational equivalence)**.** Let $\mathcal{P}$ and $\mathcal{Q}$ be two expressions. We will say that $\mathcal{P}$ and $\mathcal{Q}$ are *observationally equivalent*, written $\mathcal{P} \cong \mathcal{Q}$, if:

$$\forall q(y) \in \mathit{Poly}. \forall \sigma \in \Sigma. \forall \mathcal{C}[\;\;] \in \mathit{Con}_1. \forall o \in \mathit{Obs}. \forall S \in \mathit{PSched}. \exists i_o \in \mathbb{N}. \forall i > i_o :$$

$$\Big|\mathrm{Prob}\big[\sigma(C^{\underline{\mathrm{N}} \leftarrow i}[P^{\underline{\mathrm{N}} \leftarrow i}]) \rightsquigarrow_S o\big] - \mathrm{Prob}\big[\sigma(C^{\underline{\mathrm{N}} \leftarrow i}[Q^{\underline{\mathrm{N}} \leftarrow i}]) \rightsquigarrow_S o\big]\Big| \leq \frac{1}{q(i)}$$

Theorem 4.6. $\cong$ *is a congruence.*

Proof. Reflexivity and symmetry follow from the definition of $\cong$. For transitivity, we let $\mathcal{P} \cong \mathcal{Q}$ and $\mathcal{Q} \cong \mathcal{R}$. Then:

$$\forall q(x) \in Poly . \forall \sigma \in \Sigma . \forall \mathcal{C}[\ ] \in Con_1 . \forall o \in Obs . \forall S \in PSched . \exists i_1 \in \mathbb{N} . \forall i > i_1 :$$
$$\left|\text{Prob}\left[\sigma(C^{\underline{\mathbb{N}}\leftarrow i}[P^{\underline{\mathbb{N}}\leftarrow i}]) \rightsquigarrow_S o\right] - \text{Prob}\left[\sigma(C^{\underline{\mathbb{N}}\leftarrow i}[Q^{\underline{\mathbb{N}}\leftarrow i}]) \rightsquigarrow_S o\right]\right| \leq \frac{1}{2 \cdot q(i)}$$

and:

$$\forall q(x) \in Poly . \forall \sigma \in \Sigma . \forall \mathcal{C}[\ ] \in Con_1 . \forall o \in Obs . \forall S \in PSched . \exists i_2 \in \mathbb{N} . \forall i > i_2 :$$
$$\left|\text{Prob}\left[\sigma(C^{\underline{\mathbb{N}}\leftarrow i}[Q^{\underline{\mathbb{N}}\leftarrow i}]) \rightsquigarrow_S o\right] - \text{Prob}\left[\sigma(C^{\underline{\mathbb{N}}\leftarrow i}[R^{\underline{\mathbb{N}}\leftarrow i}]) \rightsquigarrow_S o\right]\right| \leq \frac{1}{2 \cdot q(i)}$$

We let $i_3 = \max\{i_1, i_2\}$. Then transitivity follows directly:

$$\forall q(x) \in Poly . \forall \sigma \in \Sigma . \forall \mathcal{C}[\ ] \in Con_1 . \forall o \in Obs . \forall S \in PSched . \forall i > i_3 :$$
$$\left|\text{Prob}\left[\sigma(C^{\underline{\mathbb{N}}\leftarrow i}[P^{\underline{\mathbb{N}}\leftarrow i}]) \rightsquigarrow_S o\right] - \text{Prob}\left[\sigma(C^{\underline{\mathbb{N}}\leftarrow i}[R^{\underline{\mathbb{N}}\leftarrow i}]) \rightsquigarrow_S o\right]\right| \leq \frac{1}{q(i)}$$

To establish the congruence property, we need to show that $\mathcal{P} \cong \mathcal{Q}$ implies that $\forall \mathcal{C}[\ ] \in Con_1 : \mathcal{C}[\mathcal{P}] \cong \mathcal{C}[\mathcal{Q}]$. Let $\mathcal{C}[\ ] \in Con_1$ be a context. So we must show that $\forall \mathcal{D}[\ ] \in Con_1 : \mathcal{D}[\mathcal{C}[\mathcal{P}]] \cong \mathcal{D}[\mathcal{C}[\mathcal{Q}]]$. But $\mathcal{D}[\mathcal{C}[\ ]] \in Con_1$. Since $\mathcal{P} \cong \mathcal{Q}$ it follows that $\mathcal{D}[\mathcal{C}[\mathcal{P}]] \cong \mathcal{D}[\mathcal{C}[\mathcal{Q}]]$ and the congruence property is established. $\square$

Finally

Theorem 4.7. $\mathcal{P} \simeq \mathcal{Q} \implies \mathcal{P} \cong \mathcal{Q}$.

Proof. Direct from Theorem 3.29 and the definition of $\simeq$. In particular, Theorem 3.29 says that if $\mathcal{P} \simeq \mathcal{Q}$ then $\forall \mathcal{C}[\ ] \in Con_1 : \mathcal{C}[\mathcal{P}] \simeq \mathcal{C}[\mathcal{Q}]$. But the fact that two processes are bisimular implies that the respective induced distributions on observables are identical (since in the definition of $\simeq$ we insist that the respective cumulative probability distribution functions match exactly). As a result, the two processes must be observationally equivalent. $\square$

Let Σ be the set of valuations of free variables. If $\sigma \in \Sigma$ is a valuation, then we denote the result of performing the valuation σ on $\mathcal{P}$ by $\sigma(\mathcal{P})$. We note if $\mathcal{P}$ is variable-closed then $\sigma(\mathcal{P}) \equiv \mathcal{P}$. Additionally, we note that each valuation σ can be expressed as a context $\mathcal{C}_\sigma[\ ]$. Let σ be a substitution that substitutes the value a_i for the variable x_i where $1 \leq i \leq k$. Then, we can capture this valuation in the context $\nu_{c_1}(\cdots(\nu_{c_k}(\mathtt{out}\langle c_1, a_1\rangle \mid \cdots \mid \mathtt{out}\langle c_k, a_k\rangle \mid \mathtt{in}\langle c_1, x_1\rangle . \cdots . \mathtt{in}\langle c_k, x_k\rangle . [\ ])\cdots))$. Via a series of private communications, each of the variables x_i $(1 \leq i \leq k)$ is replaced with the values a_i just as σ demands. Thus, $\cong$ can naturally be defined over all expressions, open and closed.

4.2. A Reasoning System for PPC.

The congruence and equivalence properties of $\cong$ will form the basis of our reasoning system for protocols. We present an incomplete but sound reasoning system in Figure 3. We now proceed to sketch justifications for the proof rules. Rules `CON`, `TRN`, and, `SYM` are formalizations of $\cong$'s congruence properties. The four rules `P1`, `P2`, `P3`, and `P4` formalize various properties of $|$, the parallel composition operator. We note that rule `P4`, which states that if $\mathcal{P}_1 \cong \mathcal{P}_2$ and $\mathcal{Q}_1 \cong \mathcal{Q}_2$, then $\mathcal{P}_1 \mid \mathcal{Q}_1 \cong \mathcal{P}_2 \mid \mathcal{Q}_2$, follows directly from

Figure 3 A Reasoning System for PPC

$$\mathcal{P} \mid \mathcal{Q} \cong \mathcal{Q} \mid \mathcal{P} \tag{\texttt{P1}}$$

$$\oslash \mid \mathcal{P} \cong \mathcal{P} \tag{\texttt{P2}}$$

$$(\mathcal{P} \mid \mathcal{Q}) \mid \mathcal{R} \cong \mathcal{P} \mid (\mathcal{Q} \mid \mathcal{R}) \tag{\texttt{P3}}$$

$$\frac{\mathcal{P}_1 \cong \mathcal{P}_2, \mathcal{Q}_1 \cong \mathcal{Q}_2}{\mathcal{P}_1 \mid \mathcal{Q}_1 \cong \mathcal{P}_2 \mid \mathcal{Q}_2} \tag{\texttt{P4}}$$

$$\frac{c \notin \mathit{Channel}(\mathcal{P}), x \notin \mathit{FreeVars}(\mathcal{P})}{\mathcal{P} \cong \nu_c(\texttt{out}\langle c, T\rangle \mid \texttt{in}\langle c, x\rangle.\mathcal{P})} \tag{\texttt{NU1}}$$

$$\frac{\begin{array}{c}\mathcal{C}[\texttt{out}\langle c, T\rangle] \text{ is scheduler-insensitive,}\\ c \notin \mathit{Channel}(\mathcal{C}[\oslash]), \mathit{Public}(\mathcal{C}[\texttt{out}\langle c, T\rangle]) = \{c\}\end{array}}{\exists T_{\mathcal{C}} \colon \texttt{out}\langle c, T_{\mathcal{C}}\rangle \cong \mathcal{C}[\texttt{out}\langle c, T\rangle]} \tag{\texttt{NU2}}$$

$$\frac{c \notin \mathit{Channels}(\mathcal{C}[\oslash])}{\nu_c(\mathcal{C}[\mathcal{P}]) \cong \mathcal{C}[\nu_c(\mathcal{P})]} \tag{\texttt{EXT}}$$

$$\frac{\mathcal{P} \text{ has no public channels}}{\mathcal{P} \cong \oslash} \tag{\texttt{ZER}}$$

$$\frac{\mathcal{P} \cong \mathcal{Q}, \mathcal{C}[\ \ ] \in \mathit{Con}_1}{\mathcal{C}[\mathcal{P}] \cong \mathcal{C}[\mathcal{Q}]} \tag{\texttt{CON}}$$

$$\frac{\mathcal{P} \cong \mathcal{Q}, \mathcal{Q} \cong \mathcal{R}}{\mathcal{P} \cong \mathcal{R}} \tag{\texttt{TRN}}$$

$$\frac{\mathcal{P} \cong \mathcal{Q}}{\mathcal{Q} \cong \mathcal{P}} \tag{\texttt{SYM}}$$

$$\frac{\sigma(c)(x) = \sigma(d)(x)}{\nu_c(\mathcal{P}) \cong \nu_d(\mathcal{P}^{[d/c]})} \tag{\texttt{R1}}$$

$$\frac{\begin{array}{c}\sigma(c)(x) = \sigma(d)(x),\\ d \notin \mathit{Channel}(\mathcal{P}), \mathcal{P} \cong \mathcal{Q}\end{array}}{\mathcal{P}^{[d/c]} \cong \mathcal{Q}^{[d/c]}} \tag{\texttt{R2}}$$

$$\frac{f_T \text{ and } f_U \text{ are computationally indistinguishable}}{\texttt{out}\langle c, T\rangle \cong \texttt{out}\langle c, U\rangle} \tag{\texttt{EQ1}}$$

$$\frac{\forall i \in [1..k] \colon \texttt{out}\langle c, T_i\rangle \cong \texttt{out}\langle c, U_i\rangle}{\texttt{out}\langle d, V(T_1, \ldots, T_k)\rangle \cong \texttt{out}\langle d, V(U_1, \ldots, U_k)\rangle} \tag{\texttt{EQ2}}$$

$$\frac{\begin{array}{r}\forall a_1, \ldots, a_k \colon \texttt{out}\langle c_i, U_i(a_1, \ldots, a_k)\rangle \cong \texttt{out}\langle c_i, V_i(a_1, \ldots, a_k)\rangle, i \in \{1, m\}\\ FV(\mathcal{C}[\texttt{out}\langle c_1, U_1(x_1, \ldots, x_k)\rangle), \ldots, \texttt{out}\langle c_m, U_m(x_1, \ldots, x_k)\rangle)]]) =\\ FV(\mathcal{C}[\texttt{out}\langle c_1, V_1(x_1, \ldots, x_k)\rangle), \ldots, \texttt{out}\langle c_m, U_m(x_1, \ldots, x_k)\rangle)]]) =\\ \{x_i\}\end{array}}{\begin{array}{r}\texttt{in}\langle d, x_i\rangle.\mathcal{C}[\texttt{out}\langle c_1, U_1(x_1, \ldots, x_k)\rangle)] \cdots [\texttt{out}\langle c_m, U_m(x_1, \ldots, x_k)\rangle)] \cong\\ \texttt{in}\langle d, x_i\rangle.\mathcal{C}[\texttt{out}\langle c_1, V_1(x_1, \ldots, x_k)\rangle)] \cdots [\texttt{out}\langle c_m, V_m(x_1, \ldots, x_k)\rangle)]\end{array}} \tag{\texttt{PUL}}$$

CON. In particular, $\mathcal{Q}_1 \cong \mathcal{Q}_2$ implies $\mathcal{P}_1 \mid \mathcal{Q}_1 \cong \mathcal{P}_1 \mid \mathcal{Q}_2$ (using CON and the context $\mathcal{P}_1 \mid [\ \ ]$). But, $\mathcal{P}_1 \cong \mathcal{P}_2$ implies $\mathcal{P}_1 \mid \mathcal{Q}_2 \cong \mathcal{P}_2 \mid \mathcal{Q}_2$ (again, using CON with the context $[\ \ ] \mid \mathcal{Q}_2$). We use the transitivity of $\cong$ to complete the proof.

The other three rules P1, P2, P3 all follow by giving suitable bisimulations and then exploiting Theorem 4.7. Rule P1 asserts the commutativity of parallel composition with respect to $\cong$. Rule P2 states that composing an expression $\mathcal{P}$ and the empty process using $\mid$ yields an expression that is equivalent to just $\mathcal{P}$. Lastly P3 states that the parentheses around parallel compositions do not matter *i.e.*, that we get equivalent expressions whether we stipulate that $\mid$ is left-associative or right-associative. As a consequence of P3 we will write $\mathcal{P} \mid \mathcal{Q} \mid \mathcal{R}$ for either $(\mathcal{P} \mid \mathcal{Q}) \mid \mathcal{R}$ or $\mathcal{P} \mid (\mathcal{Q} \mid \mathcal{R})$.

Rule NU1 states that one can place silent communication in front of $\mathcal{P}$ and obtain an observationally equivalent expression if the input of the silent communication is ignored. This rule is also proven by giving a suitable bisimulation. In the proof, we note that the first two silent actions (one to reduce T and one to actually perform the silent communication) take $\nu_c(\texttt{out}\langle c, T\rangle \mid \texttt{in}\langle c, x\rangle.P)$ to P with probability 1 just when the conditions on c and x are satisfied.

Rule NU2 states that if you have a scheduler-insensitive process expression with only one output on a public channel, then the entire process expression can be written as a single term placed in an output on the same channel. That is to say, we can fold the entire process into the single output. Essentially, this rule states the silent transitions are completely invisible—we can replace a process consisting of silent activity and a single public output with a single public output. The proof is fairly straightforward. Consider the term $T_{\mathcal{C},S}$ that simulates the evaluation of $\mathcal{C}[\texttt{out}\langle c, T\rangle]$ under the scheduler S and outputs a iff the observable $\langle c, a\rangle$ is generated.[4] Since $\mathcal{C}[\texttt{out}\langle c, T\rangle]$ is scheduler-insensitive it means that the behavior of the process does not depend on the scheduler. Thus, each $T_{\mathcal{C},S}$ (parameterized by choice of scheduler) is in fact the same term (since the scheduler does not matter). So in fact we have a single term $T_{\mathcal{C}}$ that simulates the evaluation of $\mathcal{C}[\texttt{out}\langle c, T\rangle]$ (under any scheduler) and returns a iff the observable $\langle c, a\rangle$ is generated. Whence we have demonstrated the existence of the term $T_{\mathcal{C}}$. To finish, we verify that $\texttt{out}\langle c, T_{\mathcal{C}}\rangle \cong \mathcal{C}[\texttt{out}\langle c, T\rangle]$ via a bisimulation.

Rule EXT allows us to "extrude" the scope of a private channel under certain constraints. The proof is an easy bisimulation that relies on alpha-renaming channel names apart. The rule ZER follows from the fact that an expression that produces no observables is trivially observationally equivalent to the zero-expression $\oslash$. The formal proof is via a bisimulation.

The first of the two rules dealing with renaming channels, R1 states that one can arbitrarily rename private channels (as long as bandwidths are respected). In this rule, $\mathcal{P}^{[d/c]}$ is taken to mean the closed expression obtained by replacing the channel name c with the channel name d (we define a similar notation for processes). The soundness of this rule follows from the fact that all private channels that can go at a given point in execution go *simultaneously*. It is easy to give a bisimulation between $\nu_c(P)$ and $\nu_d(P^{[d/c]})$ (where P is a process in the process expression $\mathcal{P}$) since if a private communication on c could go simultaneously with the private communication on d, then after renaming c to d the private communication on d (formerly on c) would still be able to go simultaneously with the private

[4]The existence of the term $T_{\mathcal{C},S}$ follows from Theorem 6.10.

communication on d (formerly also on d). Furthermore, since private channels do not produce observables, renaming private channels cannot cause problems.

The second rule regarding renaming, R2, allows us to rename public channels to a name that is not currently in use by the expression. There is an additional technical restriction that ensures that the bandwidth associated with the new name is as big as the bandwidth associated with the old name. Since, by assumption $\mathcal{P}$ and $\mathcal{Q}$ are observationally equivalent, renaming the same channel in the same way in both expressions, cannot violate their equivalent (all it does is change the distribution on observables on the channel c and observables on the channel d). Thus, a bisimulation can be easily given to verify this (the bisimulation between $P^{[d/c]}$ and $Q^{[d/c]}$ is just the bisimulation between P and Q with P and Q having the same security parameter i)—renaming the channels amounts to a systematic renaming of the edges in the associated process graphs).

The rule PUL asserts that if two functions $f_V \colon \mathbb{N}^k \times \mathbb{N} \to [0,1]$ and $g_U \colon \mathbb{N}^k \times \mathbb{N} \to [0,1]$ induce almost the same distribution on outputs, then we can "pull out" one of the arguments to the corresponding terms into an output. A proof sketch follows:

Proof. We proceed by contradiction. For each $1 \leq j \leq m$, we consider two processes. The first, $\mathcal{H}_j$, is defined as:

$$\begin{aligned}\mathtt{in}\langle d, x_i\rangle.\mathcal{C}[&\mathtt{out}\langle c_1, V_1(x_1,\ldots,x_k)\rangle,\ldots,\mathtt{out}\langle c_{j-1}, V_{j-1}(x_1,\ldots,x_k)\rangle,\\ &\mathtt{out}\langle c_j, U_j(x_1,\ldots,x_k)\rangle,\ldots,\mathtt{out}\langle c_m, U_m(x_1,\ldots,x_k)\rangle]\end{aligned}$$

and the second, $\mathcal{H}_{j+1}$, is defined as:

$$\begin{aligned}\mathtt{in}\langle d, x_i\rangle.\mathcal{C}[&\mathtt{out}\langle c_1, V_1(x_1,\ldots,x_k)\rangle,\ldots,\mathtt{out}\langle c_j, V_j(x_1,\ldots,x_k)\rangle,\\ &\mathtt{out}\langle c_{j+1}, U_{j+1}(x_1,\ldots,x_k)\rangle,\ldots,\mathtt{out}\langle c_m, U_m(x_1,\ldots,x_k)\rangle]\end{aligned}$$

Note that $\mathcal{H}_j$ and $\mathcal{H}_{j+1}$ only differ in the expression plugged into the jth hole—$\mathcal{H}_j$ has the expression $\mathtt{out}\langle c_j, U_j(x_1,\ldots,x_k)\rangle$ plugged into the jth hole while $\mathcal{H}_{j+1}$ has the expressions $\mathtt{out}\langle c_j, V_j(x_1,\ldots,x_k)\rangle$ plugged into the jth hole. Let us assume that $\mathcal{H}_j \not\cong \mathcal{H}_{j+1}$. Then there must be set of values $a_1,\ldots,a_k$ such that U_j and V_j at those values can be distinguished with non-negligible advantage. But this means that $\mathtt{out}\langle c_1, U_j(a_1,\ldots,a_k)\rangle \not\cong \mathtt{out}\langle c_1, V_j(a_1,\ldots,a_k)\rangle$ which is a contradiction. Therefore $\mathcal{H}_j \cong \mathcal{H}_{j+1}$.

In this manner we can build up the chain of equivalences $\mathcal{H}_1 \cong \mathcal{H}_2 \cong \cdots \cong \mathcal{H}_m$ and employ the transitivity of $\cong$ to obtain the desired result. □

The rule EQ1 states that if the functions f_T and f_U are computationally indistinguishable then $\mathtt{out}\langle c, T\rangle$ and $\mathtt{out}\langle c, U\rangle$ are observationally indistinguishable. This rule is validated by the proof of Theorem 5.8.

Finally, EQ2 states that if there are k pairs of terms $\langle T_i, U_i\rangle$ that induce almost the same distribution on the natural numbers, then given a term V with k free variables, the closed term $V(T_1,\ldots,T_k)$ induces almost the same distribution on natural numbers as $V(U_1,\ldots,U_k)$ whence the two processes $\mathtt{out}\langle d, V(T_1,\ldots,T_k)\rangle$ and $\mathtt{out}\langle d, V(U_1,\ldots,U_k)\rangle$ are observationally equivalent.

Proof. We proceed by contradiction. For each $1 \leq i \leq k$, we consider two processes. The first, $\mathcal{H}_j$ is defined as $\mathtt{out}\langle d, V(T_1,\ldots,T_{j-1},U_j,\ldots,U_k)\rangle$ and the second, $\mathcal{H}_{j+1}$, is defined as $\mathtt{out}\langle d, V(T_1,\ldots,T_j,U_{j+1},\ldots,U_k)\rangle$. We note that $\mathcal{H}_j$ and $\mathcal{H}_{j+1}$ only differ in the jth argument to V. Assume that $\mathcal{H}_j \not\cong \mathcal{H}_{j+1}$. If the context $\mathcal{C}[\ ]$ distinguishes between $\mathcal{H}_j$ and $\mathcal{H}_{j+1}$, then the context

$$\mathtt{in}\langle c, x\rangle.\mathcal{C}[\mathtt{out}\langle d, V(T_1, \dots, T_{j-1}, x, U_{j+1}, \dots, U_k)\rangle]$$

distinguishes between $\mathtt{out}\langle c, T_j\rangle$ and $\mathtt{out}\langle c, U_j\rangle$. This contradicts the hypothesis that each $\mathtt{out}\langle c, T_i\rangle$ is observationally equivalent to $\mathtt{out}\langle c, U_i\rangle$. In this manner we can build up a chain of equivalences $\mathcal{H}_1 \cong \cdots \cong \mathcal{H}_k$ and employ the transitivity of $\cong$ to obtain the desired result. □

5. Cryptographic Examples

This section requires that processes evaluate in polynomial time. We will assume this, deferring an actual proof of this fact until Section 6. In what follows we will denote an element x chosen uniformly at random from the set X by $x \in_R X$. In Section 5.1 we will show that our notion of asymptotic observational equivalence and the standard notion of indistinguishability by poly-time statistical tests coincide. In Section 5.2 we apply the previous observation to define pseudorandom number generators in PPC. In Sections 5.3 and 5.4 we respectively define semantic security and the Decision Diffie-Hellman Assumption (DDHA) in terms of process equivalences. Finally, in Section 5.5 we derive the equivalence between the DDHA and the semantic security of ElGamal encryption by making use of the formal proof system for PPC given in Section 4.2.

5.1. Computational Indistinguishability.

Here we show that our asymptotic observational equivalence relation coincides with the standard cryptographic notion of *indistinguishability by polynomial-time statistical tests.*

We start by recalling the notions of a function ensemble used in cryptography literature [29–31, 44, 47, 67].

Definition 5.1 (function ensemble)**.** A *function ensemble* f is an indexed family of functions $\{f_i\colon A_i \to B_i\}_{i\in\mathbb{N}}$. A function ensemble $f\colon A_i \to B_i$ is *uniform* if there exists a single Turing machine M that computes f for all values of i *i.e.*, $M(i, x) = f_i(x)$. A uniform function ensemble $f\colon A_i \to B_i$ is *poly-time* if there exists a polynomial q and a single Turing machine M such that $M(i, x)$ computes $f_i(x)$ in time at most $q(|i|, |x|)$. A uniform function ensemble $f\colon A_i \to B_i$ is *probabilistic poly-time* if f_i is a probabilistic poly-time function. A *poly-time statistical test* $\mathcal{A}\colon \{0,1\}^{m(x)} \to \{0,1\}$ is a $\{0,1\}$-valued probabilistic poly-time function ensemble.

The notion of computational indistinguishability is central to cryptography. Goldreich [31], in particular, has an excellent discussion.

Definition 5.2 (computational indistinguishability)**.** Let $q(x)$ be a positive polynomial. A uniform probabilistic poly-time function ensemble $f\colon \emptyset \to \{0,1\}^{l(x)}$ is *computationally indistinguishable* from a uniform probabilistic poly-time function ensemble $g\colon \emptyset \to \{0,1\}^{l(x)}$ just when for all poly-time statistical tests $\mathcal{A}$ we have:

$$\forall q(x).\exists i_o.\forall i > i_o\colon \left|\mathrm{Prob}\left[\mathcal{A}_i(f_i()) = \text{“1”}\right] - \mathrm{Prob}\left[\mathcal{A}_i(g_i()) = \text{“1”}\right]\right| \le \frac{1}{q(i)}$$

Definition 5.3. Let $\mathcal{P}$ be a closed expression with no public inputs and with exactly one public output on the channel c where $\sigma(c)(x) = q(x)$. We will say that the probabilistic poly-time function ensemble $f^{\mathcal{P}}\colon \emptyset \to \{0,1\}^{q(x)}$ is the *characteristic function for $\mathcal{P}$ with respect to the scheduler S* when we have that $\forall a \in \mathbb{N}$: $\mathrm{Prob}\big[f_i^{\mathcal{P}}() = a\big] = \mathrm{Prob}\big[P^{\underline{\mathrm{N}}\leftarrow i} \rightsquigarrow_S \langle c, a\rangle\big]$.

Definition 5.4. Let $f\colon \emptyset \to \{0,1\}^{q(x)}$ be a probabilistic poly-time function ensemble. Let T_f be a term such that M_{T_f} computes f. Then, we say that $\mathtt{out}\langle c, T_f\rangle$ is the *characteristic expression for f*.

Let $f\colon \emptyset \to \{0,1\}^{q(x)}$ be a probabilistic poly-time function ensemble and let $\mathcal{P}_f \equiv \mathtt{out}\langle c, T_f\rangle$ be its characteristic expression. Then, it is easy to see that $\forall S \in PSched.\forall a \in \mathbb{N}\colon \mathrm{Prob}\big[f_i() = a\big] = \mathrm{Prob}\big[P_f^{\underline{\mathrm{N}}\leftarrow i} \mid \mathtt{in}\langle c, x\rangle \rightsquigarrow_S \langle c, a\rangle\big]$.

We want to show, in this section, that the standard notion of computational indistinguishability can be captured elegantly in our system. Roughly speaking, we want to show that f is computationally indistinguishable from g iff the characteristic expression for f is observationally equivalent to the characteristic expression for g. To do so, we will show two facts:

(1) If there exists a poly-time statistical test $\mathcal{A}$ that distinguishes between f and g, then there exists a context $\mathcal{C}[\ \ ]$ that distinguishes between the characteristic expressions for f and g under any scheduler. This is shown in Lemma 5.5.
(2) If there exists a context $\mathcal{C}[\ \ ]$ that distinguishes between the characteristic expressions for f and g under a scheduler S, there exists a poly-time statistical test $\mathcal{A}$ that distinguishes between f and g. This is shown in Lemma 5.7.

We will use these two lemmas to show that the notion of PRNG can be captured in PPC.

Lemma 5.5. *Let $\mathcal{A}\colon \{0,1\}^{m(x)} \to \{0,1\}$ be a poly-time statistical test. Let $\mathcal{P}$ be any closed expression with no public inputs and exactly one public output such that $f^{\mathcal{P}}\colon \emptyset \to \{0,1\}^{m(x)}$ is its characteristic function. Then, we can construct a context $\mathcal{C}_{\mathcal{A}}[\ \ ]$ such that $f^{\mathcal{P}} \circ \mathcal{A}$ is the characteristic function for $\mathcal{C}_{\mathcal{A}}[\mathcal{P}]$ under any scheduler.*

Proof. By construction. If $\mathcal{A}$ is a poly-time statistical test then using the properties established by the conditions on terms given in Section 3.1, we can construct the context $\mathcal{C}_{\mathcal{A}}[\ \ ] \equiv \mathtt{in}\langle c, x\rangle.\mathtt{out}\langle d, T_{\mathcal{A}}(x)()\rangle \mid [\ \ ]$ with $\sigma(c)(x) = m(x)$ and $\sigma(d)(x) = 1$.

It is easy to see that this context applies the test to the $m(\underline{\mathrm{N}})$-bit output of some process "plugged" into the hole. By assumption, we have that $f^{\mathcal{P}}$ is the characteristic function for $\mathcal{P}$. Now, $T_{\mathcal{A}}$ is produced from $\mathcal{A}$ using the properties established by the conditions on terms given in Section 3.1. Hence, $\mathcal{C}_{\mathcal{A}}[\mathcal{P}]$ must produce the observables $\langle d, 0\rangle$ and $\langle d, 1\rangle$. The probability that $C_{\mathcal{A}}^{\underline{\mathrm{N}}\leftarrow i}[P^{\underline{\mathrm{N}}\leftarrow i}]$ produces the observable $\langle d, 0\rangle$ must be the same as the probability that the function $f_i^{\mathcal{P}} \circ \mathcal{A}_i$ produces zeroes under any scheduler since a scheduler must always make progress (*i.e.*, schedule something if it can) and $C_{\mathcal{A}}^{\underline{\mathrm{N}}\leftarrow i}[P^{\underline{\mathrm{N}}\leftarrow i}]$ has only one possible communication (the one between $P^{\underline{\mathrm{N}}\leftarrow i}$ and $C_{\mathcal{A}}^{\underline{\mathrm{N}}\leftarrow i}[\ \ ]$).

Similarly, the probability that $C_{\mathcal{A}}^{\underline{\mathrm{N}}\leftarrow i}[P^{\underline{\mathrm{N}}\leftarrow i}]$ produces the observable $\langle d, 1\rangle$ must be the same as the probability that the function $f_i^{\mathcal{P}} \circ \mathcal{A}_i$ produces ones under any scheduler. Hence, $f_{\mathcal{P}} \circ \mathcal{A}$ must be the characteristic function for $\mathcal{C}_{\mathcal{A}}[\mathcal{P}]$. □

We will say that an context so constructed is a *poly-time distinguishing context.*

Definition 5.6. Let $\mathcal{P}$ be a closed expression and o an observable. We will say that $f\colon \emptyset \to \{0,1\}$ is an *indicator for $\mathcal{P}$ with respect to o under the scheduler S* when $\mathrm{Prob}\big[P^{\underline{\mathrm{N}}\leftarrow i} \rightsquigarrow_S o\big] = \mathrm{Prob}\big[f_i() = 1\big]$ and $\mathrm{Prob}\big[P^{\underline{\mathrm{N}}\leftarrow i} \not\rightsquigarrow_S o\big] = \mathrm{Prob}\big[f_i() = 0\big]$.

Lemma 5.7. *Let $\mathcal{C}[\ ]$ be a context and let o be an observable. Let $f\colon \emptyset \to \{0,1\}^{m(x)}$ be any function and $\mathcal{P}_f$ be its characteristic closed expression. Then, we can specify a poly-time statistical test t from the triple $\langle \mathcal{C}[\ ], o, S\rangle$ such that $f \circ t$ is an indicator for $\mathcal{C}[\mathcal{P}_f]$ with respect to the observable o under the scheduler S.*

Proof. Our construction of t follows. We compute $f \circ t$ by evaluating the expression $\mathcal{C}[\mathtt{out}\langle d, T_f\rangle]$ (where M_{T_f} computes f) under the scheduler S and returning 1 if the observable o was generated and 0 otherwise. It is easy to check that

$$\mathrm{Prob}\big[f \circ t = 1\big] = \mathrm{Prob}\big[C^{\underline{\mathrm{N}}\leftarrow i}[\mathtt{out}\langle d, T_f\rangle] \rightsquigarrow_S o\big]$$

and that

$$\mathrm{Prob}\big[f \circ t = 0\big] = \mathrm{Prob}\big[C^{\underline{\mathrm{N}}\leftarrow i}[\mathtt{out}\langle d, T_f\rangle] \not\rightsquigarrow_S o\big]$$

□

Clearly, given an context and a closed expression, each potential observable defines a poly-time statistical test. We can now prove that an algorithm taking short strings to long strings is pseudorandom if and only if the process given by the algorithm, when evaluated on a short random input, is observationally equivalent to the process that returns a long random seed.

Theorem 5.8. *Let $f\colon \emptyset \to \{0,1\}^{l(x)}$ be a uniform probabilistic poly-time function ensemble. Let $g\colon \emptyset \to \{0,1\}^{l(x)}$ be another uniform probabilistic poly-time function ensemble. Let $\mathcal{F}$ (resp. $\mathcal{G}$) be the characteristic expression for f (resp. g).*

Then, f is computationally indistinguishable from g if and only if $\mathcal{F} \cong \mathcal{G}$.

Proof. Assume that $\mathcal{F} \cong \mathcal{G}$ and that, by way of producing a contradiction, f is not computationally indistinguishable from g. Then, we have that there exists a poly-time statistical test $\mathcal{A}$ that distinguishes between the output of f and the output of g. Hence, by Lemma 5.5, we can construct a poly-time distinguishing context $\mathcal{C}[\ ]$ that, under any scheduler, distinguishes between $\mathcal{F}$ and $\mathcal{G}$ with the same probability that $\mathcal{A}$ distinguishes between f and g. So $\mathcal{C}[\ ]$ will distinguish between $\mathcal{F}$ and $\mathcal{G}$ with probability greater than $1/q(i)$ for some polynomial $q(x)$ (as $\mathcal{A}$ distinguishes between f and g with probability greater than $1/q(i)$), thus producing a contradiction.

Now, assume that f is computationally indistinguishable from g and that, by way of producing a contradiction, $\mathcal{F} \not\cong \mathcal{G}$. Then we have that for some polynomial $p(x)$ and scheduler S there exists a context $\mathcal{D}[\ ]$ that distinguishes between the two processes with probability greater than $1/p(i)$. Let the distinguishing observation be $\langle d, i\rangle$.

We can then use Lemma 5.7 to construct a poly-time statistical test that distinguishes between the output of f and g with precisely the same probability that $\mathcal{D}[\ ]$ distinguishes between $\mathcal{F}$ and $\mathcal{G}$ under the scheduler S, thereby creating a contradiction. □

5.2. Pseudorandom Number Generators.

Definition 5.9. A function ensemble $f\colon \emptyset \to \{0,1\}^{l(x)}$ is *random poly-time* if f with respect to n is a poly-time function that returns random elements in $\{0,1\}^{l(x)}$.

We recall the notion of a pseudorandom number generator from cryptographic literature [29–31, 44, 47, 67].

Definition 5.10 (pseudorandom number generator)**.** Let $q(x)$ be a positive polynomial. A *pseudorandom number generator* (PRNG) is a uniform polynomial time function ensemble $f\colon \{0,1\}^{k(x)} \to \{0,1\}^{l(x)}$ such that for all poly-time statistical tests $\mathcal{A}$:

$$\forall q(x).\exists i_o.\forall i > i_o\colon \big|\mathrm{Prob}\big[\mathcal{A}_i(f_i(s)) = \text{“1”}\big]_{s \in_R \{0,1\}^{k(i)}} \\ - \mathrm{Prob}\big[\mathcal{A}_i(r) = \text{“1”}\big]_{r \in_R \{0,1\}^{l(i)}}\big| \leq \frac{1}{q(i)}$$

In general, $f\colon \{0,1\}^{k(i)} \to \{0,1\}^{l(i)}$ is an interesting PRNG only when $\forall x \in \mathbb{N}\colon l(x) \gg k(x)$.

Theorem 5.11. *Let $f'\colon \{0,1\}^{k(x)} \to \{0,1\}^{l(x)}$ ($\forall x \in \mathbb{N}\colon l(x) > k(x)$) be a uniform probabilistic poly-time function ensemble. Let $r\colon \emptyset \to \{0,1\}^{l(x)}$ and $s\colon \emptyset \to \{0,1\}^{k(x)}$ be uniform poly-time random function ensembles. Define f as $s \circ f'$. Let $\mathcal{F}$ (resp. $\mathcal{R}$) be the characteristic expression for f (resp. r).*

Then, f' is a PRNG if and only if $\mathcal{F} \cong \mathcal{R}$.

The closed expression $\mathcal{F}$, essentially, transmits a random seed generated by s to f' (a candidate PRNG) via function composition, and then transmits the value computed by f' on a public channel. In contrast, $\mathcal{R}$ is a closed expression that simply transmits the value computed by r (a function that returns truly random values of the same length as those generated by f) on a public channel.

Proof. This theorem is a special case of the Theorem 5.8. We note that $s \circ f'\colon \emptyset \to \{0,1\}^{l(x)}$ is a uniform probabilistic poly-time function ensemble. and that the definition of PRNG simply states that $s \circ f'$ is computationally indistinguishable from r. □

5.3. **Semantic Security.** Semantic security is an important cryptographic property due to Goldwasser and Micali [34]. Our definition of semantic security, though, is adapted from presentations by Goldreich [32] and by Goldwasser and Bellare [10, 33]. The definition of semantic security we work with assumes uniform-complexity.

Before we provide a definition of semantic security, we need to define an encryption scheme. The ideas behind public-key cryptosystems were first proposed by Diffie and Hellman [24]. Our presentation of public-key cryptosystems is drawn from Goldreich [32] as well as Goldwasser and Bellare [33].

Definition 5.12. [24, 32, 33] A *public-key encryption scheme* or, more simply, an *encryption scheme* is a triple $\langle G, E, D\rangle$ with the following properties:

(1) The key-generator is a probabilistic poly-time algorithm G that, on input 1^k (the security parameter) produces a pair $\langle e, d\rangle$ where e is the public or *encryption* key and d is the corresponding private or *decryption* key.
(2) The encryption algorithm is a probabilistic poly-time algorithm E with takes as input the security parameter 1^k, an encryption key e and a string m called the *message* or *plaintext* and produces an output string c called the *ciphertext*.

(3) The decryption algorithm is a probabilistic poly-time algorithm D with takes as input the security parameter 1^k, a decryption key d and a ciphertext c and produces a message m' such that for every m, for every $c \in E(1^k, e, m)$, the probability that $D(1^k, d, c) \neq m$ is negligible.

Now onto semantic security. Intuitively, an encryption scheme is semantically secure if, given a ciphertext, no polynomially-bounded adversary can reliably compute something about the associated plaintext *i.e.*, the encryption scheme does not reveal anything about the plaintext. We note that this version semantic security is in a chosen-plaintext model of security since the adversary, begin in possession of the public key, can encrypt a polynomial number of plaintexts it chooses before it attempts to compute something about the associated plaintext.

A useful formulation of semantic security is in terms of indistinguishability. Intuitively, if it is infeasible for any adversary to distinguish between the encryptions of any two messages (even when it chooses the messages) then the encryption scheme cannot be revealing anything about the plaintext. Goldwasser and Micali [34] showed that if an encryption scheme is secure in the indistinguishable sense, then it is semantically secure. The reverse direction, that semantic security implies security in the indistinguishable sense, was shown by Micali, Rackoff, and Sloan [48]. Goldreich [32] has a fairly detailed proof in both directions. We will work with security in the indistinguishable sense since it is more convenient for our purposes. Our presentation is drawn from Tsiounis and Yung [64] as well as Goldwasser and Bellare [33].

Definition 5.13. An encryption scheme $\langle G, E, D\rangle$ is *indistinguishably secure* if for every probabilistic poly-time Turing machine F, A, for every polynomial q, and for sufficiently large i:

$$\Big|\mathrm{Prob}\big[A(1^k, e, F(1^k, e), c) = m \,|\, c \in E(e, m_0)\big] - \\ \mathrm{Prob}\big[A(1^k, e, F(1^k, e), c) = m \,|\, c \in E(e, m_1)\big]\Big| \leq \frac{1}{q(i)}$$

where $\langle m_0, m_1\rangle \in F(1^k, e)$.

In other words, it is impossible to efficiently generate two messages (using F) such that an attack A can reliably distinguish between their encryptions. It is clear that we are considering adaptive chosen plaintext semantic security since the adversary, being in possession of the encryption key, can generate a polynomial number of messages to encrypt before it responds to the challenge.

Encoding the statement of indistinguishable encryptions as an observational equivalence in PPC is straightforward. In order to encode the statement of indistinguishable encryptions as an observational equivalence, we will assume an efficient tupling function *i.e.*, a function $\langle x_1, x_k\rangle$ that is polynomial in the lengths of $x_1, \ldots, x_k$. Since we truncate messages that are too long, a tupling function that generates outputs of super-exponential length will not work correctly.[5] In what follows, we will use the notation $\mathtt{in}\langle c, \langle x_1, \ldots, x_k\rangle\rangle$ to mean that the input obtained on channel c should be treated as a k-tuple whose ith element is named x_i. We start by defining the notion of observationally indistinguishable encryptions.

[5]Normal "diagonal" pairing or a scheme based on bit-interleaving will do nicely.

Definition 5.14. Let $\langle G, E, D\rangle$ be an encryption scheme. Then $\langle G, E, D\rangle$ is an *observationally indistinguishable encryption scheme* iff

$$\nu_c(\mathtt{out}\langle c, pkey(G(1^{\underline{\mathrm{N}}}))\rangle \mid \mathtt{in}\langle c, \mathrm{key}\rangle.\mathtt{out}\langle \mathtt{pub}, \langle \mathrm{key}, 1^{\underline{\mathrm{N}}}\rangle\rangle.\mathtt{in}\langle \mathtt{msg}, \langle m_0, m_1\rangle\rangle. \\ \mathtt{out}\langle \mathtt{challenge}, \langle \mathrm{key}, \langle m_0, m_1\rangle, E(\mathrm{key}, m_0)\rangle\rangle) \quad (\mathcal{L}\text{-}\mathcal{SS})$$

is observationally indistinguishable from

$$\nu_c(\mathtt{out}\langle c, pkey(G(1^{\underline{\mathrm{N}}}))\rangle \mid \mathtt{in}\langle c, \mathrm{key}\rangle.\mathtt{out}\langle \mathtt{pub}, \langle \mathrm{key}, 1^{\underline{\mathrm{N}}}\rangle\rangle.\mathtt{in}\langle \mathtt{msg}, \langle m_0, m_1\rangle\rangle. \\ \mathtt{out}\langle \mathtt{challenge}, \langle \mathrm{key}, \langle m_0, m_1\rangle, E(\mathrm{key}, m_1)\rangle\rangle) \quad (\mathcal{R}\text{-}\mathcal{SS})$$

where *pkey* is a function that, given a private-public key-pair, returns only the public key.

An examination of the expression $\mathcal{L}$-$\mathcal{SS}$ shows that it

(1) Generates a encryption-decryption key-pair,
(2) Publishes the security parameter and the public key,
(3) Obtains a message pair (that could be a function of the security parameter[6] and the public key[7]),
(4) Publishes the encryption of the first message, along with the message pair and the encryption key.[8]

An examination of the expression $\mathcal{R}$-$\mathcal{SS}$ shows that it is identical to expression $\mathcal{L}$-$\mathcal{SS}$ except that $\mathcal{R}$-$\mathcal{SS}$ chooses to encrypt the second message. It should be intuitively apparent that the statement that the two processes are observationally equivalent is a reformulation in PPC of the statement that $\langle G, E, D\rangle$ is indistinguishably secure (and hence semantically secure). We now proceed to formalize this intuition.

Theorem 5.15. *Let $\langle G, E, D\rangle$ be an encryption scheme. Then, $\langle G, E, D\rangle$ is semantically secure iff*

$$\nu_c(\mathtt{out}\langle c, pkey(G(1^{\underline{\mathrm{N}}}))\rangle \mid \mathtt{in}\langle c, key\rangle.\mathtt{out}\langle \mathtt{pub}, \langle key, 1^{\underline{\mathrm{N}}}\rangle\rangle.\mathtt{in}\langle \mathtt{msg}, \langle m_0, m_1\rangle\rangle. \\ \mathtt{out}\langle \mathtt{challenge}, \langle key, \langle m_0, m_1\rangle, E(key, m_0)\rangle\rangle) \quad (\mathcal{L}\text{-}\mathcal{SS})$$

is observationally indistinguishable from

$$\nu_c(\mathtt{out}\langle c, pkey(G(1^{\underline{\mathrm{N}}}))\rangle \mid \mathtt{in}\langle c, key\rangle.\mathtt{out}\langle \mathtt{pub}, \langle key, 1^{\underline{\mathrm{N}}}\rangle\rangle.\mathtt{in}\langle \mathtt{msg}, \langle m_0, m_1\rangle\rangle. \\ \mathtt{out}\langle \mathtt{challenge}, \langle key, \langle m_0, m_1\rangle, E(key, m_1)\rangle\rangle) \quad (\mathcal{R}\text{-}\mathcal{SS})$$

Proof. Let us assume that $\langle G, E, D\rangle$ is not indistinguishably secure and then show that $\mathcal{L} \cong \mathcal{R}$ does not hold. Since $\langle G, E, D\rangle$ is not semantically secure, there must exist a pair of probabilistic poly-time algorithms A, F such that, with non-negligible

[6]There is no real need to publish the security parameter, as we have done in the previous step. Since the adversary (context) and the protocol (expression) are run with the same value for the security parameter, the message-generation function F 'knows' the security parameter without the need for an explicit publish.

[7]The message-generation algorithm F can adaptively choose the message-pair based on the public key and security parameter.

[8]After this point the adversary knows the public (encryption) key as well as the message-pair. So the adversary could mount an adaptive chosen plaintext attack by selecting several plaintexts and computing their encryptions.

probability, A can distinguish between encryptions of messages chosen by F *i.e.*, for some positive polynomial q we have

$$\Big|\mathrm{Prob}\big[A(1^k, e, F(1^k, e), c) = m |\, c \in E(e, m_0)\big] - \\ \mathrm{Prob}\big[A(1^k, e, F(1^k, e), c') = m |\, c' \in E(e, m_1)\big]\Big| > \frac{1}{q(i)}$$

Let us now construct a context $\mathcal{A}[\ \ ]$ out of A and F.

$$\begin{aligned}&[\ \ ] \mid \mathtt{in}\langle \mathtt{pub}, \langle \mathrm{key}, \mathrm{sec}\rangle\rangle. \\ &\qquad \mathtt{out}\langle \mathtt{msg}, F(\mathrm{sec},\mathrm{key})\rangle.\mathtt{in}\langle \mathtt{challenge}, \langle \mathrm{pubkey}, \text{m-pair}, \mathrm{cipher}\rangle\rangle. \\ &\qquad\qquad \mathtt{out}\langle \mathtt{response}, A(\mathrm{sec}, \mathrm{pubkey}, \text{m-pair}, \mathrm{cipher})\rangle \\ &\qquad\qquad\qquad\qquad\qquad\qquad \mid \mathtt{in}\langle \mathtt{response}, x\rangle.\emptyset\end{aligned}$$

Let us consider $\mathcal{A}[\mathcal{L}]$ (resp. $\mathcal{A}[\mathcal{R}]$). The first thing to note is that in any evaluation path of $\mathcal{A}[\mathcal{L}]$ (resp. $\mathcal{A}[\mathcal{R}]$), at each evaluation step there is only one communication step that can happen. Hence, the choice of scheduler is irrelevant since schedulers are stochastic.

The attacking context $\mathcal{A}[\ \ ]$ supplies a pair of messages chosen by F (based on the security parameter and public key) to $\mathcal{L}$ (resp. $\mathcal{R}$) which then returns the encryption of the first (resp. second) message along with the encryption key and the pair of messages. Then, the attacking context $\mathcal{A}[\ \ ]$ applies A to the tuple consisting of the security parameter, message-pair, encryption key, and ciphertext. This yields a guess as to the message which is then check for accuracy by $\mathcal{L}$ (resp. $\mathcal{R}$). Since $\mathcal{A}[\ \ ]$ simply applies A to the encryption of a message chosen from the pair given by F, it must distinguish between encryptions of messages chosen by F with the same probability that A distinguishes between encryptions of messages chosen by F. Whence if A can reliably distinguish between encryptions of messages (*i.e.*, A can distinguish with probability better than half), it follows that $\mathcal{A}[\ \ ]$ can distinguish between $\mathcal{L}$ and $\mathcal{R}$ with non-negligible probability (*i.e.*, with probability greater than $1/q(\underline{\mathrm{N}})$ which is the probability with which A distinguishes between encryptions of messages given by F). Thus, $\mathcal{L} \not\cong \mathcal{R}$ since we have constructed a distinguishing context $\mathcal{A}[\ \ ]$ out of A and F.

Let us now tackle the reverse direction. We assume that $\mathcal{L} \not\cong \mathcal{R}$ and show that $\langle G, E, D\rangle$ is not indistinguishably secure. Since $\mathcal{L} \not\cong \mathcal{R}$, we have a context $\mathcal{A}[\ \ ]$ that, under some perceptible scheduler S, distinguishes between $\mathcal{L}$ and $\mathcal{R}$ on the basis of some observable o. Furthermore, we assume $\mathcal{A}[\ \ ]$ must provide messages to $\mathcal{L}$ (resp. $\mathcal{R}$) in order for the protocol to run (since $\mathcal{L}$ and $\mathcal{R}$ differ only in the challenge step, any distinguishing observable can only be generated after the challenge step has occurred[9]). Thus there must exists a probabilistic poly-time algorithm F such that the probability that $\mathcal{A}[\ \ ]$, using security parameter k and public key e, provides the message pair π to $\mathcal{L}$ (resp. $\mathcal{R}$) is precisely the probability that $F(1^k, e)$ generates the message pair π. We can then create an attack A that distinguishes between encryptions of messages picked by F as follows. We compute $A(1^k, e, \pi = F(1^k, e), c = E(e, m_b))$ where $m_b \in \pi$ by constructing the expression

[9]If the context $\mathcal{A}[\ \ ]$ does not communicate with the expression plugged into its hole and the expression plugged into the hole does not run via any internal public communications it can perform, the context cannot distinguish between any two expressions. In particular $\mathcal{A}[\emptyset]$ will be equivalent to $\mathcal{A}[\mathcal{P}]$ for any $\mathcal{P}$.

$\mathcal{P}_{\langle e,\pi,c\rangle}$ and then evaluating $\mathcal{A}[\mathcal{P}_{\langle e,\pi,c\rangle}]$ under the scheduler S and with security parameter $\underline{\mathrm{N}}$ set to k. If the observable o is generated we return a "1" and if the observable o is not generated we return a "0". Let us denote the encryption of the ith message in the message-pair ($i \in \{0,1\}$) by c_i. The expression $\mathcal{P}_{\langle e,\pi,c_i\rangle}$ is defined as

$$\nu_c(\mathtt{out}\langle c, pkey(G(1^{\underline{\mathrm{N}}}))\rangle \mid \mathtt{in}\langle c, \mathrm{key}\rangle.\mathtt{out}\langle \mathtt{pub}, \langle \mathrm{key}, 1^{\underline{\mathrm{N}}}\rangle\rangle. \\ \mathtt{in}\langle \mathtt{msg}, \langle m_0, m_1\rangle\rangle.\mathtt{out}\langle \mathtt{challenge}, \langle e, \pi, c_i\rangle\rangle)$$

Clearly, the function A will successfully distinguish between encryptions of messages in π with the same probability that, using the scheduler S, $\mathcal{A}[\ \ ]$ distinguishes between $\mathcal{P}_{\langle e,\pi,c_0\rangle}$ and $\mathcal{P}_{\langle e,\pi,c_1\rangle}$. Hence $\langle G, E, D\rangle$ is not indistinguishably secure. Thus $\langle G, E, D\rangle$ is indistinguishably secure iff $\mathcal{L} \cong \mathcal{R}$. We finish by noting that $\langle G, E, D\rangle$ is indistinguishably secure iff it is semantically secure. □

5.4. **The Decision Diffie-Hellman Assumption.** We start by defining the Decision Diffie-Hellman assumption [24]. Our version is drawn from Boneh [12] and Tsiounis and Yung [64]. Goldreich [32], as well as Cramer and Shoup [20] also offer helpful discussions.

A group family $\mathbb{G}$ is a set of finite cyclic groups $\{G_p\}$ where the index p ranges over an infinite set. An instance generator $IG(n)$ takes security parameter n, runs in time polynomial in n and returns a random index p as well as a generator g of the group G_p.

Definition 5.16. A *Decision Diffie-Hellman algorithm A* for $\mathbb{G}$ is a probabilistic polynomial time algorithm such that:

(1) Given $\langle p, g, g^a, g^b, g^c\rangle$ the algorithm A reliably decides if $c = ab$; and,
(2) There exists a non-constant positive polynomials $q(\cdot)$ such that $IG(n) = \langle p, g\rangle$ implies that $|\langle p, g\rangle| = \Omega(q(n))$.

The probability is taken over the probability that the instance generator $IG(1^n)$ returns $\langle p, g\rangle$ given n, random choice of a, b, c in $[1..\operatorname{ord} G_p]$ and random bits used by A. The *Decision Diffie-Hellman assumption* for $\mathbb{G}$ is that no Decision Diffie-Hellman algorithm exists.

The condition on the instance generator that the size of the outputs of the instance generator grow faster than some non-constant positive polynomial ensures that the groups returned by the instance generator become larger as the security parameter increases. Since the instance generator runs in polynomial time, the outputs of the generator cannot become too large *i.e.*, there exists another non-constant positive polynomial $r(\cdot)$ such that $|\langle p, g\rangle| = O(r(n))$.

Remark 5.17. We give some examples of groups in which the DDHA is believed to be intractable. These examples are drawn from Boneh [12].

(1) Let $p = 2q + 1$ where both p and q are prime. Let Q_p be the subgroup of quadratic residues in $\mathbb{Z}_p^*$. It is a cyclic group of prime order. This group is parameterized by the choice of prime p.
(2) Let $N = pq$ where $p, q, \frac{p-1}{2}, \frac{q-1}{2}$ are prime. Let T be the cyclic subgroup of order $(p-1)(q-1)$. The DDHA is believed to be intractable for T. The group T is parameterized by choice of N.
(3) Let p be a prime and $E_{a,b}/\mathbb{F}_p$ be an elliptic curve where $|E_{a,b}|$ is prime. This group is parametrized by choice of p, a, b.

(4) Let p be a prime and J be a Jacobian of a hyper elliptic curve over $\mathbb{F}_p$ with a prime number of reduced divisors. The group is parameterized by p and the coefficients of the defining equation.

The index p encodes the group parameters. The instance generator selects a random member of the group family $\mathbb{G}$ by picking the group parameters according to some suitable distribution. In general we might not wish to use a uniform distribution; for instance, one might wish to avoid primes of the form 2^k+1 in the case where we are working with a subgroup of quadratic residues in $\mathbb{Z}_p^*$. Thus, the DDHA challenge $\langle p, g, g^a, g^b, g^c\rangle$ is completely general over the type of group we are working in: p selects a member of the group family, g is a generator of that group, a, b, c are integers chosen from $[1.. \operatorname{ord} G_p]$, g^a is the a-fold application of the group operation, and $=$ is group identity.

We can express the DDHA as an observational equivalence. Let us define two expressions using the convention that $\langle p, g, g^a, g^b, g^c\rangle$ is shorthand for the term $T(a, b, c)$ defined by the program

1. $\langle p, g\rangle = IG(1^{\underline{\mathrm{N}}})$
2. return $\langle p, g, g^a, g^b, g^c\rangle$

where n is the security parameter. With this notation in place, we define what it means for the group family $\mathbb{G}$ to be observationally DDHA-secure.

Definition 5.18. The group family $\mathbb{G}$ is *observationally DDHA-secure* if

$$\mathtt{out}\langle \mathtt{ch}, \langle p, g, g^a, g^b, g^{ab}\rangle \,|\, a, b \in_R [1.. \operatorname{ord} G_p]\rangle \cong \mathtt{out}\langle \mathtt{ch}, \langle p, g, g^a, g^b, g^c\rangle \,|\, a, b, c \in_R [1.. \operatorname{ord} G_p]\rangle$$

where the term $\langle p, g, g^a, g^b, g^{ab}\rangle \,|\, a, b \in_R [1.. \operatorname{ord} G_p]$ denotes the term $T(a, b, ab)$ with a, b chosen uniformly at random from $[1.. \operatorname{ord} G_p]$.

We note the DDHA is known to be easy for certain groups (such as G_2). Thus, in order for the above equivalence to hold, it is necessary that the instance generator does not select groups for which the DDHA is easy. However, as we have previously noted, the instance generator can select groups within the group family according to arbitrary distributions and, thereby, avoid easy cases.

The following theorem validates our attempt to express the DDHA in PPC by asserting that the assumption of being observationally DDHA-secure is precisely the DDHA.

Theorem 5.19. *The DDHA holds for the group family $\mathbb{G}$ iff $\mathbb{G}$ is observationally DDHA-secure.*

Proof. A special case of Theorem 5.8. □

5.5. **The Semantic Security of ElGamal Encryption.** We now proceed to give an example of the use of PPC to establish properties of protocols. In particular, we will show that ElGamal encryption is semantically secure given the Decision Diffie-Hellman assumption.

We start by describing the ElGamal encryption scheme [28].

Definition 5.20. Let $\cdot$ denote the group operation and $=$ denote group equality. An *ElGamal encryption scheme* is a triple $\langle G, E, D\rangle$ of probabilistic poly-time algorithms such that:

(1) The key generating algorithm G, on input 1^k outputs a public key $e = \langle p, g, g^a\rangle$ and a private key $d = a$ where $\langle g, p\rangle \in IG(1^k)$, $a \in_R [1..\operatorname{ord} G_p]$.
(2) An encryption algorithm E that, on input, $e = \langle p, g, g^a\rangle$ and m outputs $\langle g^b, m \cdot g^{ab} \bmod p\rangle$ as the ciphertext (where $b \in_R [1..\operatorname{ord} G_p]$).
(3) A decryption algorithm D that, given ciphertext $c = \langle k, c'\rangle$ and decryption key d computes c'/k^d. To see why this works, we note that $k = g^a$, $c' = m \cdot g^{ab} \bmod p$, and $d = b$ for some a, b, m. Then

$$\frac{c'}{k^d} = \frac{m \cdot g^{ab}}{g^{b^a}} = \frac{m \cdot g^{ab}}{g^{ab}} = m$$

In order to show that ElGamal is semantically secure given the Decision Diffie-Hellman assumption, we will derive the assertion that ElGamal is an observationally indistinguishable encryption scheme from an assertion expressing the DDHA. In particular, we will derive the observational equivalence of

$$\begin{aligned}\nu_c(\texttt{out}\langle c, pkey(G(1^{\underline{\mathrm{N}}}))\rangle \mid \texttt{in}\langle c, \langle p, g, g^a\rangle\rangle.\\ \texttt{out}\langle \texttt{pub}, \langle\langle p, g, g^a\rangle, 1^{\underline{\mathrm{N}}}\rangle\rangle.\texttt{in}\langle \texttt{msg}, \langle m_0, m_1\rangle\rangle.\\ \texttt{out}\langle \texttt{challenge}, \langle\langle p, g, g^a\rangle, \langle m_0, m_1\rangle, \langle g^b, m_0 \cdot g^{ab}\rangle\rangle \mid b \in_R [1..\operatorname{ord} G_p]\rangle) \quad (\mathcal{L}\text{-}\mathcal{EG})\end{aligned}$$

and

$$\begin{aligned}\nu_c(\texttt{out}\langle c, pkey(G(1^{\underline{\mathrm{N}}}))\rangle \mid \texttt{in}\langle c, \langle p, g, g^a\rangle\rangle.\\ \texttt{out}\langle \texttt{pub}, \langle\langle p, g, g^a\rangle, 1^{\underline{\mathrm{N}}}\rangle\rangle.\texttt{in}\langle \texttt{msg}, \langle m_0, m_1\rangle\rangle.\\ \texttt{out}\langle \texttt{challenge}, \langle\langle p, g, g^a\rangle, \langle m_0, m_1\rangle, \langle g^b, m_1 \cdot g^{ab}\rangle\rangle \mid b \in_R [1..\operatorname{ord} G_p]\rangle) \quad (\mathcal{R}\text{-}\mathcal{EG})\end{aligned}$$

from the assertion that

$$\begin{aligned}\texttt{out}\langle \texttt{ch}, \langle p, g, g^a, g^b, g^{ab}\rangle \mid a, b \in_R [1..\operatorname{ord} G_p]\rangle \cong\\ \texttt{out}\langle \texttt{ch}, \langle p, g, g^a, g^b, g^c\rangle \mid a, b, c \in_R [1..\operatorname{ord} G_p]\rangle \quad (\mathcal{DDHA})\end{aligned}$$

(From Section 5.4 we know that this second assertion is an expression of the DDHA).

Main Theorem 5.21. *If the Decision Diffie-Hellman assumption holds for a group family $\mathbb{G}$, then ElGamal encryption using $\mathbb{G}$ is semantically secure against adaptive chosen plaintext attacks. Furthermore, we can derive, using just the proof rules for PPC given in Figure 3, the assertion in PPC that ElGamal encryption using $\mathbb{G}$ is semantically secure (i.e., the equivalence $\mathcal{L}\text{-}\mathcal{EG} \cong \mathcal{R}\text{-}\mathcal{EG}$) from the assertion in PPC that the DDHA holds for $\mathbb{G}$ (i.e., the equivalence $\mathcal{DDHA}$).*

The proof is fairly straightforward. We will start with the equivalence $\mathcal{DDHA}$ and build up the equivalence between $\mathcal{L}\text{-}\mathcal{EG}$ and $\mathcal{R}\text{-}\mathcal{EG}$ by systematically transforming the term that outputs a challenge instance of the DDHA. In particular, we note that, with the exception of the message-pair, the challenge instance $\langle p, g, g^a, g^b, g^c\rangle$ looks almost like a challenge instance of ElGamal's semantic security (which is a tuple $\langle\langle p, g, g^a\rangle, \pi, \langle g^b, g^c\rangle\rangle$) where the message being encrypted is g^c divided by g^{ab}. Thus, it seems reasonable to assume that we can systematically transform the DDHA challenge instance into a challenge instance for ElGamal's semantic security (where the messages are provided by the adversary).

Before we give the formal proof, we will give an informal mathematical proof showing that DDHA implies the semantic security of ElGamal encryption. This mathematical argument will be formalized in PPC and constitute the first half of the formal proof. We start by assuming that the $\langle p, g, g^a, g^b, g^c\rangle$ is computationally indistiguishable from the $\langle p, g, g^a, g^b, g^{ab}\rangle$ (where p, g are given by the instance generator and a, b, c are chosen uniformly at random from $[1..\operatorname{ord} G_p]$). We will use $\doteq$ to denote computational indistinguishability. We recall to the reader's attention that computational indistinguishability is transitive. So we have that

$$\langle p, g, g^a, g^b, g^{ab}\rangle \doteq \langle p, g, g^a, g^b, g^c\rangle$$

Since the two tuples are computationally indistinguishable, it follows that

$$\forall m_0, m_1 \in G_p \colon \langle p, g, g^a, m_0, m_1, g^b, g^{ab}\rangle \doteq \langle p, g, g^a, m_0, m_1, g^b, g^c\rangle$$

i.e., if no algorithm can distinguish between $\langle p, g, g^a, g^b, g^{ab}\rangle$ and $\langle p, g, g^a, g^b, g^c\rangle$, then no algorithm can do so given two arbitrary elements of the group. Furthermore, since g is a generator of the group it follows that $m_0 \cdot g^{ab} = g^{r+ab}$. Since a, b are chosen uniformly at random from $[1..\operatorname{ord} G_p]$, it follows that $g^{r+ab} = g^{c'}$ for randomly chosen $c' \in [1..\operatorname{ord} G_p]$. Thus we get that for all $m_0, m_1 \in [1..\operatorname{ord} G_p]$

$$\langle p, g, g^a, m_0, m_1, g^b, g^{ab}\rangle \doteq \langle p, g, g^a, m_0, m_1, g^b, m_0 \cdot g^{ab}\rangle$$

In other words, since g is a generator we can view g^c as g^{ab} multiplied by some arbitrary message. Similarly, we can show that for all message m_0, m_1

$$\langle p, g, g^a, m_0, m_1, g^b, g^{ab}\rangle \doteq \langle p, g, g^a, m_0, m_1, g^b, m_1 \cdot g^{ab}\rangle$$

whence the transitivity of $\doteq$ yields

$$\langle p, g, g^a, m_0, m_1, g^b, m_0 \cdot g^{ab}\rangle \doteq \langle p, g, g^a, m_0, m_1, g^b, m_1 \cdot g^{ab}\rangle$$

for all message m_0, m_1. This claim is (almost) the assertion that ElGamal encryption is semantically secure since the tuple contains the public key $\langle p, g, g^a\rangle$, the message pair $\langle m_0, m_1\rangle$ and the encryption of one of the messages $\langle g^b, m_i \cdot g^{ab}\rangle$. In the formal proof, we will continue from this point using various structural rules in PPC to convert this assertion to an assertion of the right form.

Proof. We start by assuming that $\mathcal{DDHA}$ is true *i.e.*, that

$$\texttt{out}\langle \texttt{ch}, \langle p, g, g^a, g^b, g^{ab}\rangle | \, a, b \in_R [1..\operatorname{ord} G_p]\rangle \cong \\ \texttt{out}\langle \texttt{ch}, \langle p, g, g^a, g^b, g^c\rangle | \, a, b, c \in_R [1..\operatorname{ord} G_p]\rangle$$

holds. For each $m_0 \in G_p$ we can use EQ2 to obtain that

$$\texttt{out}\langle \texttt{ch}, \langle p, g, g^a, m_0, g^b, m_0 \cdot g^{ab}\rangle | \, a, b \in_R [1..\operatorname{ord} G_p]\rangle \cong \\ \texttt{out}\langle \texttt{ch}, \langle p, g, g^a, m_0, g^b, m_0 \cdot g^c\rangle | \, a, b, c \in_R [1..\operatorname{ord} G_p]\rangle$$

Furthermore, since $m = g^d$ for some choice of d, it follows that the term $m_0 \cdot g^c$ and the term g^c (with c chosen uniformly at random in $[1..\operatorname{ord} G_p]$) both induce the same distribution on elements of the group. Thus

$$\forall m_0 \in G_p \colon \texttt{out}\langle \texttt{ch}, \langle p, g, g^a, m_0, g^b, m_0 \cdot g^c\rangle | \, a, b, c \in_R [1..\operatorname{ord} G_p]\rangle \cong \\ \texttt{out}\langle \texttt{ch}, \langle p, g, g^a, m_0, g^b, g^c\rangle | \, a, b, c \in_R [1..\operatorname{ord} G_p]\rangle$$

Then, TRN (transitivity) gives us

$$\forall m_0 \in G_p \colon \texttt{out}\langle \texttt{ch}, \langle p, g, g^a, m_0, g^b, m_0 \cdot g^{ab}\rangle | \, a, b \in_R [1..\operatorname{ord} G_p]\rangle \cong \\ \texttt{out}\langle \texttt{ch}, \langle p, g, g^a, m_0, g^b, g^c\rangle | \, a, b, c \in_R [1..\operatorname{ord} G_p]\rangle$$

Another application of EQ2 gives us

$$\forall m_0, m_1 \in G_p \colon \texttt{out}\langle \texttt{ch}, \langle p, g, g^a, m_0, m_1, g^b, m_0 \cdot g^{ab}\rangle \,|\, a, b \in_R [1..\operatorname{ord} G_p]\rangle \cong \texttt{out}\langle \texttt{ch}, \langle p, g, g^a, m_0, m_1, g^b, g^c\rangle \,|\, a, b, c \in_R [1..\operatorname{ord} G_p]\rangle \qquad (\dagger)$$

Using a similar argument to the one used to establish (†) we can show

$$\forall m_0, m_1 \in G_p \colon \texttt{out}\langle \texttt{ch}, \langle p, g, g^a, m_0, m_1, g^b, m_1 \cdot g^{ab}\rangle \,|\, a, b \in_R [1..\operatorname{ord} G_p]\rangle \cong \texttt{out}\langle \texttt{ch}, \langle p, g, g^a, m_0, m_1, g^b, g^c\rangle \,|\, a, b, c \in_R [1..\operatorname{ord} G_p]\rangle \qquad (\ddagger)$$

Then, using SYM (symmetry) and TRN, we can combine (†) and (‡) to obtain

$$\forall m_0, m_1 \in G_p \colon \texttt{out}\langle \texttt{ch}, \langle p, g, g^a, m_0, m_1, g^b, m_0 \cdot g^{ab}\rangle \,|\, a, b \in_R [1..\operatorname{ord} G_p]\rangle \cong \texttt{out}\langle \texttt{ch}, \langle p, g, g^a, m_0, m_1, g^b, m_1 \cdot g^{ab}\rangle \,|\, a, b \in_R [1..\operatorname{ord} G_p]\rangle \qquad (✠)$$

Until this point, we have taken advantage of various mathematical facts that follow from working with G_p. In particular, being in G_p allows us to show that g^c is just g^{ab} times some random string R. We can then multiply the g^c and g^{ab} terms by some message since the distribution induced by g^c is the same as that induced by g^{ab}. Multiplying this message into g^c and g^{ab} is effectively the same as multiplying g^{ab} by one of two random messages. Thus far, we have formalized in PPC the informal mathematical arguments given just prior to the formal proof.

We have obtained an observational equivalence that looks almost like the observational equivalence stating the semantic security of ElGamal encryption. In particular, we have an observational equivalence that states the tuple consisting of the elements of an ElGamal public key, the elements of a message-pair, and the elements of the encryption of the first message, is computationally indistinguishable from a tuple consisting of the elements of the same ElGamal public key, the elements of the same message-pair, but the elements of an encryption of the second message. This is almost precisely the definition of the semantic security of ElGamal (in terms of indistinguishability of encryptions) except that the challenge needs to consist of three tuples encoding respectively the public key, the message-pair, and the ciphertext. We also need those elements of the expression that allow an adversary to provide the message-pair after seeing the public key. We can finish the proof by repeatedly using PUL to 'pull out' arguments that we want to provide via channels. In particular, we will pull out the message pair and provide it on an input that waits for a suitable output by an adversary. We will also make use of R2 to ensure that the channel-names in the derived expression match the channel-names used in our statement of semantic security (see Theorem 5.15).

We now complete the derivation. First, we get the challenge into the right form using EQ2 on (✠):

$$\forall m_0, m_1 \in G_p \colon \texttt{out}\langle \texttt{ch}, \langle\langle p, g, g^a\rangle, \langle m_0, m_1\rangle, \langle g^b, m_0 \cdot g^{ab}\rangle\rangle \,|\, a, b \in_R [1..\operatorname{ord} G_p]\rangle \cong \texttt{out}\langle \texttt{ch}, \langle\langle p, g, g^a\rangle, \langle m_0, m_1\rangle, \langle g^b, m_1 \cdot g^{ab}\rangle\rangle \,|\, a, b \in_R [1..\operatorname{ord} G_p]\rangle$$

Using proof rule PUL and R2 we can obtain the message-pair from a context:

$$\begin{aligned}
&\mathtt{in}\langle \mathtt{msg}, \langle m_0, m_1\rangle\rangle.\\
&\quad \mathtt{out}\langle \mathtt{challenge}, \langle\langle p,g,g^a\rangle, \langle m_0,m_1\rangle, \langle g^b, m_0\cdot g^{ab}\rangle\rangle \,|\, a,b \in_R [1..\operatorname{ord} G_p]\rangle\\
&\qquad\qquad\cong\\
&\mathtt{in}\langle \mathtt{msg}, \langle m_0, m_1\rangle\rangle.\\
&\quad \mathtt{out}\langle \mathtt{challenge}, \langle\langle p,g,g^a\rangle, \langle m_0,m_1\rangle, \langle g^b, m_1\cdot g^{ab}\rangle\rangle \,|\, a,b \in_R [1..\operatorname{ord} G_p]\rangle
\end{aligned}$$

The context $\mathtt{out}\langle \mathtt{pub}, \langle\langle p,g,g^a\rangle, 1^{\underline{\mathrm{N}}}\rangle\rangle.[\ \ ]$ where, the term $\langle p,g,g^a\rangle$ is shorthand for the term $U(a)$ defined as

1. $\langle p, g\rangle = IG(1^{\underline{\mathrm{N}}})$
2. return $\langle p,g,g^a\rangle$

and an application of CON allow us to publish the security parameter and public key:

$$\begin{aligned}
&\mathtt{out}\langle \mathtt{pub}, \langle\langle p,g,g^a\rangle, 1^{\underline{\mathrm{N}}}\rangle\rangle.\mathtt{in}\langle \mathtt{msg}, \langle m_0, m_1\rangle\rangle.\\
&\quad \mathtt{out}\langle \mathtt{challenge}, \langle\langle p,g,g^a\rangle, \langle m_0,m_1\rangle, \langle g^b, m_0\cdot g^{ab}\rangle\rangle \,|\, a,b \in_R [1..\operatorname{ord} G_p]\rangle\\
&\qquad\qquad\cong\\
&\mathtt{out}\langle \mathtt{pub}, \langle\langle p,g,g^a\rangle, 1^{\underline{\mathrm{N}}}\rangle\rangle.\mathtt{in}\langle \mathtt{msg}, \langle m_0, m_1\rangle\rangle.\\
&\quad \mathtt{out}\langle \mathtt{challenge}, \langle\langle p,g,g^a\rangle, \langle m_0,m_1\rangle, \langle g^b, m_1\cdot g^{ab}\rangle\rangle \,|\, a,b \in_R [1..\operatorname{ord} G_p]\rangle
\end{aligned}$$

Finally, another application of PUL, followed by an application of CON allows us to 'pull' out the p, g, a from the term, and then (via CON) provide them by making use of the key generator (which will generate the pair consisting of $\langle p,g,g^a\rangle$ (the public key) an a (the private key). Since the challenge transmitted requires only the value g^a and does not require a, we do not have to transmit the private key. To ensure security, we also use CON to wrap up this communication in a private channel.

$$\begin{aligned}
&\nu_c(\mathtt{out}\langle c, pkey(G(1^{\underline{\mathrm{N}}}))\rangle \mid \mathtt{in}\langle c, \langle p,g,g^a\rangle\rangle.\\
&\qquad \mathtt{out}\langle \mathtt{pub}, \langle\langle p,g,g^a\rangle, 1^{\underline{\mathrm{N}}}\rangle\rangle.\mathtt{in}\langle \mathtt{msg}, \langle m_0, m_1\rangle\rangle.\\
&\quad \mathtt{out}\langle \mathtt{challenge}, \langle\langle p,g,g^a\rangle, \langle m_0,m_1\rangle, \langle g^b, m_0\cdot g^{ab}\rangle\rangle \,|\, b \in_R [1..\operatorname{ord} G_p]\rangle)\\
&\qquad\qquad\cong\\
&\nu_c(\mathtt{out}\langle c, pkey(G(1^{\underline{\mathrm{N}}}))\rangle \mid \mathtt{in}\langle c, \langle p,g,g^a\rangle\rangle.\\
&\qquad \mathtt{out}\langle \mathtt{pub}, \langle\langle p,g,g^a\rangle, 1^{\underline{\mathrm{N}}}\rangle\rangle.\mathtt{in}\langle \mathtt{msg}, \langle m_0, m_1\rangle\rangle.\\
&\quad \mathtt{out}\langle \mathtt{challenge}, \langle\langle p,g,g^a\rangle, \langle m_0,m_1\rangle, \langle g^b, m_1\cdot g^{ab}\rangle\rangle \,|\, b \in_R [1..\operatorname{ord} G_p]\rangle)
\end{aligned}$$

But this equivalence is precisely the statement that ElGamal encryption is semantically secure (see Theorem 5.15). Hence, given the DDHA we can show that ElGamal encryption is semantically secure. □

Remark 5.22. We draw the reader's attention to the fact that the proof of the semantic security of ElGamal encryption from the Decision Diffie-Hellman assumption in the language of PPC can be split into two distinct parts. In the first part, we used mathematical facts about the group operation $\cdot$ in the group G_p to transform the DDHA challenge $\langle p,g,g^a,g^b,g^c\rangle$ into a tuple $\langle p,g,g^a,m_0,m_1,g^b,m_i\cdot g^{ab}\rangle$ that *almost* looked like a semantic security of ElGamal encryption challenge. Furthermore, the key fact about G_p used was that g was a generator of G_p whence for some c it followed that $m \cdot g^{ab} = g^c$. The other facts used were trivial facts about the group product (*i.e.*, its associativity) and pairing (*i.e.*, $\pi_i(\langle m_1, m_2\rangle = m_i)$). The remainder of the proof consisted of purely structural transformations on the expressions. We suggest that proofs in PPC, in general, can be separated into a large sequence of structural transformations required to achieve the right shape of the protocol, couple with a few transformations whose soundness are grounded in mathematical facts about the special nature of the problem. These special facts can be represented with special hypotheses (like $\mathcal{DDHA}$) and special rules (such as the one equation $m \cdot g^{ab}$ with g^c). This proof could be rendered entirely mechanical by axiomatizing in PPC, as it were, important mathematical properties about groups that are relevant for this proof. Taken with the structural rules of Figure 3 this would allow us to derive ElGamal's semantic security from the DDHA in an entirely mechanical manner. This observation also draws up the intriguing possibility that we can work 'backward' from a statement of a desired property to some conditions that must hold. In particular, we could have started with the statement of ElGamal encryption's semantic security and reversed the proof's structural transformations to obtain the DDHA—in fact that is just what we will do in Theorem 5.23. In general, we hope to be able to precisely state the security conditions that need to hold of the primitives for a given protocol to satisfy a desired security property.

Main Theorem 5.23. *If ElGamal encryption using the group family $\mathbb{G}$ is semantically secure against adaptive chosen plaintext attacks, then the Decision Diffie-Hellman assumption holds for $\mathbb{G}$. Furthermore, we can derive, using just the proof rules for PPC given in Figure 3, the assertion in PPC that the DDHA holds for $\mathbb{G}$ (i.e., the equivalence $\mathcal{DDHA}$) from the assertion in PPC that ElGamal encryption using $\mathbb{G}$ is semantically secure (i.e., the equivalence $\mathcal{L}\text{-}\mathcal{EG} \cong \mathcal{R}\text{-}\mathcal{EG}$).*

Proof. Since we assume that ElGamal encryption using the group family $\mathbb{G}$ is semantically secure, it follows that

$$\begin{aligned}&\nu_c(\mathtt{out}\langle c, pkey(G(1^{\underline{N}}))\rangle \mid \mathtt{in}\langle c, \langle p, g, g^a\rangle\rangle.\\ &\qquad\qquad \mathtt{out}\langle \mathtt{pub}, \langle\langle p, g, g^a\rangle, 1^{\underline{N}}\rangle\rangle.\mathtt{in}\langle \mathtt{msg}, \langle m_0, m_1\rangle\rangle.\\ &\quad \mathtt{out}\langle \mathtt{challenge}, \langle\langle p, g, g^a\rangle, \langle m_0, m_1\rangle, \langle g^b, m_0\cdot g^{ab}\rangle\rangle \mid b \in_R [1..\operatorname{ord} G_p]\rangle) \qquad (\mathcal{L}\text{-}\mathcal{EG})\end{aligned}$$

is observationally equivalent to

$$\begin{aligned}&\nu_c(\mathtt{out}\langle c, pkey(G(1^{\underline{N}}))\rangle \mid \mathtt{in}\langle c, \langle p, g, g^a\rangle\rangle.\\ &\qquad\qquad \mathtt{out}\langle \mathtt{pub}, \langle\langle p, g, g^a\rangle, 1^{\underline{N}}\rangle\rangle.\mathtt{in}\langle \mathtt{msg}, \langle m_0, m_1\rangle\rangle.\\ &\quad \mathtt{out}\langle \mathtt{challenge}, \langle\langle p, g, g^a\rangle, \langle m_0, m_1\rangle, \langle g^b, m_1\cdot g^{ab}\rangle\rangle \mid b \in_R [1..\operatorname{ord} G_p]\rangle) \qquad (\mathcal{R}\text{-}\mathcal{EG})\end{aligned}$$

But then, using the rule CON we obtain that

$$\begin{gathered}\nu_{\mathtt{pub}}(\nu_{\mathtt{msg}}(\nu_{\mathtt{challenge}}(\mathcal{L}\text{-}\mathcal{EG} \mid \mathtt{in}\langle \mathtt{pub}, \langle p, g, g^a\rangle\rangle.\mathtt{out}\langle \mathtt{msg}, \langle 1, g^r\rangle \mid r \in_R [1..\operatorname{ord} G_p]\rangle.\\ \mathtt{in}\langle \mathtt{challenge}, \langle\langle p, g, g^a\rangle, \langle m_0, m_1\rangle, \langle g^b, g^c\rangle\rangle\rangle.\mathtt{out}\langle \mathtt{ddh}, \langle p, g, g^a, g^b, g^c\rangle\rangle)))\\ \cong\\ \nu_{\mathtt{pub}}(\nu_{\mathtt{msg}}(\nu_{\mathtt{challenge}}(\mathcal{R}\text{-}\mathcal{EG} \mid \mathtt{in}\langle \mathtt{pub}, \langle p, g, g^a\rangle\rangle.\mathtt{out}\langle \mathtt{msg}, \langle 1, g^r\rangle \mid r \in_R [1..\operatorname{ord} G_p]\rangle.\\ \mathtt{in}\langle \mathtt{challenge}, \langle\langle p, g, g^a\rangle, \langle m_0, m_1\rangle, \langle g^b, g^c\rangle\rangle\rangle.\mathtt{out}\langle \mathtt{ddh}, \langle p, g, g^a, g^b, g^c\rangle\rangle)))\end{gathered}$$

We draw the reader's attention to the fact that the only observables that the left-hand side and right-hand side expressions can generate are on the output on the channel named `ddh`. Furthermore the output on `ddh` is a DDHA-challenge tuple. The final output of the left-hand side expression is the tuple $\langle p, g, g^a, g^b, 1{\cdot}g^{ab}\rangle$ where p, g are chosen by the ElGamal instance generator and a, b are chosen at random from $[1..\operatorname{ord} G_p]$. For the right-hand side we get the tuple $\langle p, g, g^a, g^b, g^{r+ab}\rangle$ where r is chosen at random from $[1..\operatorname{ord} G_p]$. Since g is a generator of G_p, g^{r+ab} is a random element of G_p. Finally, both the right-hand side and the left-hand side are scheduler-insensitive processes since, at each stage of evaluation, only one possible action can occur. We can then use proof rule `NU2` to obtain the equivalence

$$\mathtt{out}\langle d, \langle p, g, g^a, g^b, g^{ab}\rangle \,|\, \langle p, g\rangle \in IG(1^{\underline{\mathrm{N}}}), a, b \in [1..\operatorname{ord} G_p]\rangle \cong$$
$$\mathtt{out}\langle d, \langle p, g, g^a, g^b, g^{c}\rangle \,|\, \langle p, g\rangle \in IG(1^{\underline{\mathrm{N}}}), a, b, c \in [1..\operatorname{ord} G_p]\rangle$$

which is precisely the statement of the DDHA. □

This proof is interesting in two respects. The first is that we use a context to create a semantic security challenge of a particular form. We then use the fact that private channels are invisible to 'collapse' the entire process into a term. In general, this technique will be useful in going from long expressions to shorter expressions.

6. PPC Evaluates in Polynomial-Time

Our proof of the polynomial time-bound will proceed in two stages. First we will show that the length of any *perceptible* path (*i.e.*, a path whose edges are labelled only by actual or silent actions) is a polynomial in the security parameter. This follows from the proof of Lemma D.2. In particular, since there are only a finite number of inputs and outputs in a closed expression, there must only be a finite number of actual steps and silent actions that can occur during any evaluation of a process expression. We will show that this finite number is actually a polynomial in the length of the expression, and polynomial in the security parameter.

Then we will show that each action can be performed in time polynomial in the length of the expression, and polynomial in the security parameter. To do so, we will construct a Turing machine $M_{\mathcal{P}}$ (see Section 6.1) for each closed expression $\mathcal{P}$ such that $M_{\mathcal{P}}(i, S)$ evaluates $P^{\underline{\mathrm{N}}\leftarrow i}$ and produces the observable o with probability $\mathrm{Prob}\big[P^{\underline{\mathrm{N}}\leftarrow i} \rightsquigarrow_S o\big]$.

Our discussion of a mechanical evaluation procedure for processes will benefit from the following useful definition:

Definition 6.1. We say that $\mathcal{Q}$ is a *component* of $\mathcal{P}$ just when $\mathcal{P} \equiv \mathcal{P}_1 \mid \mathcal{P}_2$ and $\mathcal{Q} \equiv \mathcal{P}_k$ or $\mathcal{Q}$ is a component of $\mathcal{P}_k$ (with $k \in \{1, 2\}$).

6.1. **A Turing Machine for Process Evaluation.**

In this section we will define an *evaluator for the closed expression* $\mathcal{P}$ as a probabilistic Turing machine $M_{\mathcal{P}}$ that takes as input a value i for the security parameter and a perceptible scheduler S and then continues as follows:

(1) It first performs a reduction step. Even if the process expression is blocked, performing a reduction step will not harm anything since reductions are idempotent (Lemma 3.7).
(2) It then performs a non-reduction action (it can do this since at this point it is working on a blocked process). It does this by first computing the

set of all actual actions and silent actions that the process can take—we shall call this set the set of *eligible actions*. Then it applies a perceptible scheduler to select the particular action being taken. Since this action is always a non-reduction silent action (since the process is blocked at this stage) or an actual action (*i.e.*, in the equivalence class of some $\mathtt{in}\langle c, a\rangle \cdot \mathtt{out}\langle c, a\rangle$) we shall call this step a *communication step*. We note that the probability of taking any action in this step is equal to the probability that the perceptible scheduler selects that action out of all the (actual, non-actual, and silent[10] actions available—since perceptible schedulers never pick non-actual actions, we can safely drop them from the set of possible actions.

(3) It then repeats this procedure until the set of eligible actions becomes empty at which point it halts.

We note that we have provided an evaluation procedure only for closed expressions. The evaluation of expressions in general can be defined as evaluation under a possible valuation of the free variables. Also, we will refer to steps 1 and 2 together as a single *evaluation step* in the evaluation of the expression.

Before we define our Turing machine, we will make an assumption that will simplify both our construction as well as our analysis of the time-bound on process evaluation (see Section 6). Since $\mathcal{P}$ is a closed expression, every term with variables must appear in the scope of sufficiently many inputs that bind those variables. For example, the term $T(x_1, \ldots, x_i)$ must appear in the scope of at least i inputs, each of those inputs binding exactly one of those variables. Furthermore, we note that a term in only reduced when it becomes exposed *i.e.*, is not longer in the scope of any input. At this point, the term has no variables—values have been substituted for all variables in the term. Thus, there is no harm in *delaying* the substitution of values into variables of terms until the terms need to be evaluated. In particular, our Turing machine, instead of performing the substitution indicated by a communication step, will create a λ-substitution instance of that term. For example, given a term $T(x_1, \ldots, x_i)$ and a communication step that substitutes a for x_k, our Turing machine will create the term $\lambda x_k.T(x_1, \ldots, x_i)\ a$ rather than $T(x_1, \ldots, x_{k-1}, a, x_{k+1}, \ldots, x_i)$ (assuming, of course, that $1 \le k \le i$). We note that a single λ-substitution instance of a term requires a constant amount of space for the λ and enough space for the variable name and the value to substitute for the variable name. We will assume that variable names takes an amount of space linear in the length of $\mathcal{P}$—since the length of $\mathcal{P}$ is finite and each variable name requires at least one character to write down, the number of distinct variables appearing in $\mathcal{P}$ must be linear in the length of $\mathcal{P}$.

Now, we note that each variable in $\mathcal{P}$ gets a value via some channel. Thus, the size of the largest value that can be substituted for that variable is bounded by the bandwidth polynomial associated with the channel on which the value arrives. Let $B_{\mathcal{P}}$ be the set of bandwidth polynomials appearing in $\mathcal{P}$ and define the polynomial χ at value i as $\chi(i) = \sum_{i=j}^{k} a_i x^{p_i}$ where a_i is defined as the maximum over all the bandwidth polynomials of the absolute value of the coefficient of the x^{p_i} term (we

[10]In keeping with our intuitive semantics, we will ensure that a silent action is picked out of each component of a parallel composition *i.e.*, given $P_1 \mid \cdots \mid P_k$ a silent transition will consist of a silent action taken by each P_j, $(1 \le j \le k)$, keeping in mind that we allow a process to take a trivial action to itself if it has no silent actions.

assume that if the bandwidth polynomial $p(\cdot)$ does not have a x^{p_i} term, then the associated coefficient is 0); and k is the maximum order of any of the bandwidth polynomials in $B_{\mathcal{P}}$. Clearly $\chi(\cdot)$ so defined is a polynomial and, equally clearly thanks to the construction of $\chi(\cdot)$, $\forall a \in \mathbb{N}.\forall q(\cdot) \in B_{\mathcal{P}} \colon \chi(a) \geq q(a)$.

So our first assumption is that every value a to be substituted for a variable in $\mathcal{P}$ has length given by $\chi(\underline{\mathrm{N}})$. Since all values to be substituted for variables are truncated by some bandwidth polynomial (since these values are transmitted over channels), this ensures that we leave enough space to write down any value that could possibly be substituted for that variable during evaluation. In particular, during as we perform a communication step, if we leave enough room for the λ-substitution instance that we will create (constant space for the λ plus linear in the length of $\mathcal{P}$ space for the variable name plus $\chi(\underline{\mathrm{N}})$ space for the value), we will not need to "push" symbols around in order to make room for the value we want to write down. But how many communication steps can possibly happen? From Lemma 6.4 we know that the number of communication steps can be bounded by the minimum of the number of inputs ($in(\mathcal{P})$) and the number of outputs ($out(\mathcal{P})$) in the process (see Defn. 6.3). Thus, we will assume that each term T has size

$$c_T + \min\{in(\mathcal{P}), out(\mathcal{P})\} \cdot (\chi(x) + \upsilon(\mathcal{P}) + c_\lambda)$$

where c_T is a constant giving the length of the term T and c_λ is the constant amount of space needed for the λ in a λ-substitution instance, and $\upsilon(\mathcal{P})$ bounds the amount of space needed for variables in $\mathcal{P}$—since each distinct variable in $\mathcal{P}$ needs a unique name and the number of distinct variables in $\mathcal{P}$ is limited by the length of $\mathcal{P}$, it follows that $\upsilon(\mathcal{P})$ is linear in the length of $\mathcal{P}$. We note that if either $in(\mathcal{P})$ or $out(\mathcal{P})$ is 0 then, technically, the inequality $\chi(x) \leq c_T + \min\{in(\mathcal{P}), out(\mathcal{P})\} \cdot (\chi(x) + \upsilon(\mathcal{P}) + c_\lambda)$ is false. However, this is not a significant issue since evaluation cannot proceed if there are either no inputs or no outputs.

Before we continue let us consider the space needed to write down the value to which a term reduces. We note that a term either appears in a match or in an output. For match terms, we note that we never really require the value to which the term evaluates. All we need know is whether the match holds or fails. Thus the space allocated to a term in the first assumption is sufficient since all we will do is either erase the match (assuming the match was passed) or erase the entire expression (assuming the match was failed). For an output term on the channel c, we note that at most $\sigma(c)(i)$ bits of the term's value is transmitted. Thus we need only write down the $\sigma(c)(i)$ least significant bits. However we note that $\sigma(c)(x) \leq \chi(x) \leq c_T + \min\{in(\mathcal{P}), out(\mathcal{P})\} \cdot (\chi(x) + \upsilon(\mathcal{P}) + c_\lambda)$ whence the amount of space already budgeted by the first assumption is more than enough. From Defn. 6.3 we note that $c_T + \min\{in(\mathcal{P}), out(\mathcal{P})\} \cdot (\chi(x) + \upsilon(\mathcal{P}) + c_\lambda)$ is just a polynomial in the security parameter that we shall denote by $p_T(x)$. The result of this assumption is that each reduction simulated on our Turing machine will either not change the length of the process or decrease the length of the process (by replacing inputs and outputs with $\oslash$s and evaluating matches).

The ideas given thus far should be enough to allow us to proceed with the proof of the time-bound on evaluation (Section 6). We will leave the detailed construction of the evaluator for the expression $\mathcal{P}$ to Appendix C.

6.2. **The Correctness of Evaluators.**

Lemma 6.2. *Let $\mathcal{P}$ be a closed expression and $M_{\mathcal{P}}$ be the associated evaluator. Then $M_{\mathcal{P}}(i, S)$ generates the observable o with the same probability that $P^{\underline{\mathrm{N}} \leftarrow i}$ generates the observable o under the scheduler S.*

The proof will refer to details about the construction of the evaluator and so we invite the reader to peruse Appendix C before reading the sketch of the proof.

Proof sketch. We will show that each evaluation step is correct *i.e.*, the if $Q^{\underline{\mathrm{N}} \leftarrow i}$ is obtained from $\mathcal{P}i$, then the next evaluation step that $M_{\mathcal{P}}(i, S)$ performs results in the process $R^{\underline{\mathrm{N}} \leftarrow i}$ with the same probability that $Q^{\underline{\mathrm{N}} \leftarrow i}$ evaluates to $R^{\underline{\mathrm{N}} \leftarrow i}$ in one evaluation step using the scheduler S.

We proceed by induction on the length of the evaluation path. The basis is proved exactly as the inductive step and so we assume the lemma for evaluation paths of length at most k and establish the result for the $k+1$th evaluation step.

An evaluation step consists of a scheduling step (which includes performing the communication) and a reduction step. Let us consider the scheduling step first. Constructing the set of equivalence classes of actions is a deterministic step: each process induces exactly one set of equivalence classes of actions. We then invoke the scheduler by provide the set of equivalence classes to the scheduler is input. Since the scheduler is assumed to be probabilistic poly-time, it follows that the scheduling step selects the communication type to perform with the right probability. We then need to pick an action of that type to perform. This is done by selecting an action of that type uniformly at random. As we have already noted, this is precisely what happens during the evaluation of a process: a scheduler selects an action type and then picks an action in that type uniformly at random. Thus, the action to perform is picked with the right probability. Actually performing that communication is another deterministic step.

Now for the reduction step. Whenever the evaluator encounters an exposed term of the form $(\lambda x_1 \cdots \lambda x_k.T)\ a_k \cdots a_1$, the evaluator replaces that term with the value obtained by running $M_T(a_1, \ldots, a_k)$. Since terms are assumed to be probabilistic poly-time computable, $M_T(a_1, \ldots, a_k)$ will produce a with probability p iff the term T reduces, with probability p, to a on inputs $a_1, \ldots, a_k$.

Thus the $k+1$th evaluation step is performed with the right probability and the inductive step is established. We now know that each evaluation step is correct. Then it is easy to see that the probability that $M_{\mathcal{P}}(i)(S)$ produces the observable o is precisely the probability that $P^{\underline{\mathrm{N}} \leftarrow i}$ generates o under scheduler S. In particular, the probability that $P^{\underline{\mathrm{N}} \leftarrow i}$ generates the observable o under scheduler S is the probability of some evaluation path producing that observable. Since all evaluation steps are performed correctly, all evaluation paths are performed correctly whence all observables are generated with the right probability. $\square$

6.3. **A Bound on the Length of any Evaluation Sequence.**

Definition 6.3. Let $\mathcal{P}$ be a closed expression. We define $\mathit{in}(\mathcal{P})$, the *number of inputs in* $\mathcal{P}$, and $\mathit{out}(\mathcal{P})$, the *number of outputs in* $\mathcal{P}$, inductively as follows:

$$\begin{aligned}
\mathit{in}(\oslash) &= 0 & \mathit{out}(\oslash) &= 0\\
\mathit{in}(\nu_c(\mathcal{Q})) &= \mathit{in}(\mathcal{Q}) & \mathit{out}(\nu_c(\mathcal{Q})) &= \mathit{out}(\mathcal{Q})\\
\mathit{in}(\mathtt{in}\langle c,x\rangle.\mathcal{Q}) &= 1+\mathit{in}(\mathcal{Q}) & \mathit{out}(\mathtt{in}\langle c,x\rangle.\mathcal{Q}) &= \mathit{out}(\mathcal{Q})\\
\mathit{in}(\mathtt{out}\langle c,T\rangle.\mathcal{Q}) &= \mathit{in}(\mathcal{Q}) & \mathit{out}(\mathtt{out}\langle c,T\rangle.\mathcal{Q}) &= 1+\mathit{out}(\mathcal{Q})\\
\mathit{in}([T_1=T_2].\mathcal{Q}) &= \mathit{in}(\mathcal{Q}) & \mathit{out}([T_1=T_2].\mathcal{Q}) &= \mathit{out}(\mathcal{Q})\\
\mathit{in}(\mathcal{Q}_1 \mid \mathcal{Q}_2) &= \mathit{in}(\mathcal{Q}_1)+\mathit{in}(\mathcal{Q}_2) & \mathit{out}(\mathcal{Q}_1 \mid \mathcal{Q}_2) &= \mathit{out}(\mathcal{Q}_1)+\mathit{out}(\mathcal{Q}_2)\\
\mathit{in}(!_{q(\underline{\mathtt{N}})}.(\mathcal{Q})) &= q(\underline{\mathtt{N}})\cdot \mathit{in}(\mathcal{Q}) & \mathit{out}(!_{q(\underline{\mathtt{N}})}.(\mathcal{Q})) &= q(\underline{\mathtt{N}})\cdot \mathit{out}(\mathcal{Q})
\end{aligned}$$

It is clear from the definition that $\mathit{in}(\mathcal{P})$ and $\mathit{out}(\mathcal{P})$ are polynomials in $\underline{\mathtt{N}}$ that are positive for all input values.

Lemma 6.4. *Let $\mathcal{P}$ be a closed process. Then, for all values i for the security parameter and perceptible schedulers S, during any evaluation of $P^{\underline{\mathtt{N}}\leftarrow i}$, at most* $\min\{\mathit{in}(\mathcal{P})(i), \mathit{out}(\mathcal{P})(i)\}$ *evaluation steps occur.*

Proof. Consider any evaluation step (see Section 6.1) during the evaluation of $\mathcal{P}$. An evaluation step is made up of a reduction step followed by a communication step. From the definition of reduction we now that a reduction step cannot increase the number of inputs our outputs in a process (at worst, by causing a match to fail, it will only reduce the number of inputs and outputs). Clearly, a communication step syntactically eliminates an input/output pair from the expression being evaluation. Thus each evaluation step must reduce the number of inputs and outputs by at least one. Furthermore, each evaluation step requires both an input and an output (so that the communication step can go through). Thus during evaluation we can only have at most $\min\{\mathit{in}(\mathcal{P})(i), \mathit{out}(\mathcal{P})(i)\}$ communication steps whence we can only have at most that many evaluation steps. $\square$

Corollary 6.5. *Let $\mathcal{P}$ be a closed expression. Then the number of evaluation steps that can possibly occur during an evaluation of $\mathcal{P}$ is a polynomial in the security parameter.*

Proof. Using Lemma 6.4, we define the number of evaluating steps h that occur during the evaluation of $\mathcal{P}$ as

$$h(i) = \min\{\mathit{in}(\mathcal{P})(i), \mathit{out}(\mathcal{P})(i)\} \leq \mathit{in}(\mathcal{P})(i) + \mathit{out}(\mathcal{P})(i)$$

Clearly, h is a polynomial in the security parameter. $\square$

6.4. A Bound on the Time for an Evaluation Step.

Definition 6.6. Let $\mathcal{P}$ be a closed expression. We inductively define a polynomial $\mathit{length}(\mathcal{P})$, the length of $\mathcal{P}$, as follows:

$$\begin{aligned}
\mathit{length}(\emptyset) &= 1\\
\mathit{length}(\nu_c(\mathcal{Q})) &= 1 + \mathit{length}(\mathcal{Q})\\
\mathit{length}(\mathtt{in}\langle c, x\rangle.\mathcal{Q}) &= 1 + \mathit{length}(\mathcal{Q})\\
\mathit{length}(\mathtt{out}\langle c, T\rangle.\mathcal{Q}) &= 1 + p_T(\underline{\mathrm{N}}) + \mathit{length}(\mathcal{Q})\\
\mathit{length}([T_1 = T_2].\mathcal{Q}) &= 1 + p_{T_1}(\underline{\mathrm{N}}) + p_{T_2}(\underline{\mathrm{N}}) + \mathit{length}(\mathcal{Q})\\
\mathit{length}(\mathcal{Q}_1 \mid \mathcal{Q}_2) &= 1 + \mathit{length}(\mathcal{Q}_1) + \mathit{length}(\mathcal{Q}_2)\\
\mathit{length}(!_{q(\underline{\mathrm{N}})}.(\mathcal{Q})) &= q(\underline{\mathrm{N}}) \cdot \mathit{length}(\mathcal{Q})
\end{aligned}$$

It is clear from the definition that $\mathit{length}(\mathcal{P})$ is an always positive polynomial of $\underline{\mathrm{N}}$. Furthermore, the correctness of this definition follows from the construction of the evaluating Turing machine (see Appendix C). In particular, this definition accounts for the padding that we add to the expression to ensure that we never need to push symbols around when we perform communication steps.

Lemma 6.7. *Let $P^{\underline{\mathrm{N}}\leftarrow i}$ be a process and $R^{\underline{\mathrm{N}}\leftarrow i}$ a process such that $R^{\underline{\mathrm{N}}\leftarrow i}$ is the result of performing some number of evaluation steps on $P^{\underline{\mathrm{N}}\leftarrow i}$. Then $\mathit{length}(\mathcal{R})(i) \leq \mathit{length}(\mathcal{P})(i)$.*

Proof. Each reduction step eliminates matches and rewrites terms with values. Thus a reduction step cannot increase the length of a process expression.

Also, each communication step removes at least one input and output from $P^{\underline{\mathrm{N}}\leftarrow i}$. Furthermore, the contribution to the length of $P^{\underline{\mathrm{N}}\leftarrow i}$ by a particular term T accounts for the maximum size that T can grow to by the creation of successive λ-substitution instances of T by communication steps. Hence $\mathit{length}(\mathcal{R})(i)$ cannot exceed $\mathit{length}(\mathcal{P})(i)$. □

Lemma 6.8. *Let $\mathcal{P}$ be a closed, blocked expression and S a perceptible scheduler. Then the evaluator for $\mathcal{P}$, $M_{\mathcal{P}}$, selects the next action to perform in time polynomial in the length of $\mathcal{P}$ and polynomial in the security parameter.*

Proof. Since $\mathcal{P}$ is blocked, it has not exposed matches or outputs that need reduction. Before the scheduler is invoked, we need to construct the set of equivalence classes of actions. In one pass, we can decide if there are any silent actions since silent actions are handled differently from public actions. In the case that silent actions exist, there is only one equivalence class—$[\tau]_\sim$ (the scheduler must pick silent actions if it can).

In the case that no silent actions exist, we need to build the set of equivalence classes of actions induced by $\mathcal{P}$. This can be done in one pass to collect all the inputs and outputs (this pass is poly-time in the length of $\mathcal{P}$). Since a public action consists of an input and an output, the set of actions we need to consider is of size at most $\mathit{in}(\mathcal{P}) \cdot \mathit{out}(\mathcal{P})$. We then need one pass over this set to place each action in its equivalence class. Determining the equivalence class of an action can be done in constant time since we need just check the channel name and value. Thus, the set of equivalence classes of actions A can be computed in time polynomial in the $\underline{\mathrm{N}}$. Finally, the scheduler will pick an element from A in time polynomial in the size of A. Whence the time needed for the entire procedure of running the scheduler and deciding which action-class to evaluate is at most a polynomial $s_{\mathcal{P}} = c \cdot (in(\mathcal{P}) \cdot out(\mathcal{P}) + length(P))$ where c is a constant. $\square$

Lemma 6.9. *Let $\mathcal{P}$ be a closed expression. Then the Turing machine for $\mathcal{P}$, $M_{\mathcal{P}}$, performs a communication step in time polynomial in the length of $\mathcal{P}$ and $\underline{\mathrm{N}}$.*

Proof. Whenever $M_{\mathcal{P}}$ hits an exposed term $(\lambda x_1 \ldots \lambda x_i.T)\ a_1 \ldots a_k$, it evaluates T by evaluating the algorithm M_T at $a_1, \ldots, a_k$.[11] The runtime is bounded by the polynomial $q_T(|a_1|, \ldots, |a_k|)$ (since, terms are probabilistic poly-time functions of their arguments). However, each argument to a term comes via a communication step or directly from the choice of value for the security parameter. Thus, each input has size at most $\chi(\underline{\mathrm{N}})$ whence the cost of evaluating a term is given by the polynomial $q_T(\chi(x), \ldots, \chi(x))$ which is just a polynomial t_T in one variable—the security parameter. Since the number of terms in a process is a function of its length, the reduction step needs to do at most

$$r_{\mathcal{P}}(\underline{\mathrm{N}}) = length(\mathcal{P}) \cdot \left(\sum_{k=1}^{m} t_{T_k}(\underline{\mathrm{N}}) \right)$$

where $Terms(\mathcal{P}) = \{T_1, \ldots, T_m\}$.

Next, we need to select an equivalence class from the set of actions that can be taken. From Lemma 6.8, this can be done in time $s_{\mathcal{P}}(\underline{\mathrm{N}})$ which is polynomial in $\underline{\mathrm{N}}$ and the length of $\mathcal{P}$. We then need to select the particular action in the selected equivalence class of actions. This is done using the following procedure:

(1) First, we make a list of all the actions of $\mathcal{P}$ in the selected equivalence class $[\alpha]_{\sim}$. This is done in time polynomial in the length of $\mathcal{P}$.
(2) Then, we select one of those actions uniformly at random. We recall to the reader's attention that PPC is a species of reactive bisimulation (see Section 3.4) *i.e.*, the scheduler selects an equivalence class of actions to perform and the syntax of the process determines the particular action of that equivalence class to perform. Thus, having chosen a particular type of action using the scheduler, we must select the particular action to perform. An examination of the inference rules of Figure 2 shows that each action in the same equivalence class is assigned the probability $1/n$ where n is a normalization factor. Hence, the actions are selected according to the uniform distribution.

    This procedure takes time polynomial in the number of actions equivalent to α which in turn is at most $in(\mathcal{P}) \cdot out(\mathcal{P})$ which is a polynomial in $\underline{\mathrm{N}}$.

Thus, selecting the particular action to evaluate can be done in time polynomial in $\underline{\mathrm{N}}$ and the length of $\mathcal{P}$. We shall call this polynomial $d_{\mathcal{P}}$.

Next we need to perform a series of substitutions. No matter how many there are, they can be done in two passes. The first makes a list of all the values that need to be moved around. The second goes from left to right erasing outputs and replacing inputs of the form $\mathtt{in}\langle c, x\rangle.Q$ with $[a/x]Q$ (using blank symbols to erase unnecessary symbols). Thus we can perform a communication step in time $m_{\mathcal{P}} = c \cdot length(\mathcal{P})$.

[11] Since we only evaluate exposed terms and since we assume that $\mathcal{P}$ is closed, we can be sure that all variables have values substituted for them.

Thus the time for an evaluation step is given by the polynomial $e_{\mathcal{P}} = r_{\mathcal{P}} + s_{\mathcal{P}} + d_{\mathcal{P}} + m_{\mathcal{P}}$ which is a polynomial in the length of $\mathcal{P}$ and $\underline{\mathrm{N}}$. □

6.5. **A Bound on Evaluation Time.**

Main Theorem 6.10. *Let $\mathcal{P}$ be a closed expression and $M_{\mathcal{P}}$ its evaluator. Then $M_{\mathcal{P}}$ evaluates $\mathcal{P}$ in time polynomial in the security parameter.*

Proof. We know that there are at most $h(\underline{\mathrm{N}})$ evaluation steps during the evaluation of $\mathcal{P}$ (Lemma 6.5). Furthermore, each evaluation step takes time at most $e_{\mathcal{P}}$ (Lemma 6.9). But $e_{\mathcal{P}}$ is a polynomial in the length of $\mathcal{P}$ and the security parameter *i.e.*, $e_{\mathcal{P}} = \hat{e}(length(\mathcal{P}), \underline{\mathrm{N}})$ for some polynomial $\hat{e}(x, y)$. Since $length(\mathcal{P})$ is a polynomial in $\underline{\mathrm{N}}$ it follows that $e_{\mathcal{P}}$ is a polynomial in $\underline{\mathrm{N}}$. Thus, the overall cost of evaluating $\mathcal{P}_i$ is $\Phi_{\mathcal{P}}(\underline{\mathrm{N}}) = h(\underline{\mathrm{N}}) \cdot e_{\mathcal{P}}(\underline{\mathrm{N}})$ which is just a polynomial in the security parameter. □

7. Conclusion and Future Directions

The language presented in this paper allows us to define probabilistic processes which communicate over a network that gives an adversary access to those communications.

Our process language uses a separate term language to actually perform computations (the expression $\mathtt{in}\langle c, x\rangle.\mathtt{out}\langle d, x+2\rangle$, roughly speaking, computes the function $x + 2$ but all the computation is done by the term $x + 2$). One might argue that such a distinction is not particularly necessary since the presence of polynomially-bounded iteration in our process language should make it expressive enough for probabilistic poly-time functions even without a dedicated term language. This is certainly possible and we leave it to the interested reader to formalize the necessary constructions and establish properties equivalent to the ones shown here. One major advantage of our approach is that we simplify the proof of the time bound on process evaluation since we do not have to establish that computation be bounded by poly-time; rather we can stipulate that the term language we use have the desired poly-time property. Additionally, we more accurately model the situation from the point of view of an adversary since the sources of computation (terms) are invisible to an adversary; only the information flows can be accessed by an attacker.

One significant result is to show that expressions written in our language are all bounded by polynomial time. We also proposed a definition of observational equivalence for probabilistic programs that is based on the view that large differences in probability are easier to observe than small differences. When we distinguish between "large" and "small" using asymptotic behavior, we arrive at a definition of observational equivalence that coincides with a standard concept from cryptography, namely, indistinguishability by polynomial-time statistical tests. While we have not fully explored the consequences of this definition, we believe it may shed new light on other basic concepts in cryptography, such as the distinction between semantically secure and non-malleable cryptosystems.

The steps taken in this paper form the beginning of a larger program that we hope to carry out over the next few years. In part following the program established in the study of spi-calculus [3], we hope to develop methods for reasoning about observational equivalence and use these methods to establish security properties of various protocols. In this paper we also presented a reasoning mechanism inspired by the standard notion of a bisimulation between processes due to Milner [50]. While this mechanism has proved sufficient to establish some preliminary equivalences, much work needs to be done in developing a range of techniques sufficiently expressive to deal with the non-trivial problems this approach was intended to handle. In particular, an important next step is to make our bisimilarity relation asymptotic. That is to say, rather than requiring that the cumulative probability distribution functions (μ-function) of two processes be identical, our next step will be to allow the cumulative distributions to be (asymptotically) "close".

Finally, we applied our calculus to some simple and well-known cryptographic facts. First, we showed that a fundamental cryptographic notion, that of a pseudorandom number generator, is expressible in our language in a fairly natural way. In future, we hope to demonstrate that a range of other foundational cryptographic primitives, such as pseudorandom function families, are all similarly expressible in our language. By capturing a variety of essential cryptographic notions in our framework, we hope to lay the groundwork for a methodology by which we reason "backwards" from a desired property of a particular protocol to the properties that the various primitives in the protocol must possess in order to establish the desired protocol property. We expect some interesting foundational questions to arise in the formulation of security properties such as authentication and secrecy.

Second, we provided a formal proof of the equivalence of the semantic security of ElGamal encryption and the Decision Diffie-Hellman assumption. While this fact is well-known to the cryptographic community, it is tremendously encouraging that we can offer a simple and entirely mechanical proof of a non-trivial cryptographic fact in our calculus. In future, we hope to extend our toolkit of equivalences so that we may offer similarly mechanical formal proofs of properties of more complex security protocols. It may also be possible to develop model-checking procedures along the lines of these already explored for probabilistic temporal logics [21, 35, 36, 38]. In fact, we hope to be able to develop automated reasoning procedures for use in a network security setting using techniques developed in our study of the properties of our process calculus.

Acknowledgements: We thank M. Abadi, R. Amadio, D. Boneh, R. Canetti, A. Datta, C. Dwork, R. van Glabbeek, M. Hofmann, A. Jeffrey, S. Kannan, B. Kapron, P. Lincoln, A. Martin, R. Milner, I. Mironov, M. Mitchell, M. Naor, P. Panangaden, D. Sarenac, and P. Selinger for helpful discussions and advice on relevant literature.

References

[1] Abadi, M., and Fournet, C. Mobile values, new names, and secure communication. In *28th ACM Symposium on Principles of Programming Languages* (2001), pp. 104–115.

[2] Abadi, M., and Gordon, A. D. A bisimulation method for cryptographic protocol. In *Proc. ESOP 98* (1998), Lecture notes in Computer Science, Springer.

[3] Abadi, M., and Gordon, A. D. A calculus for cryptographic protocols: the spi calculus. *Information and Computation 143* (1999), 1–70. Expanded version available as SRC Research Report 149 (January 1998).

[4] Abadi, M., and Jürjens, J. Formal eavesdropping and its computational interpretation. In *Proc. Fourth International Symposium on Theoretical Aspects of Computer Software (TACS2001)* (Tohoku University, Sendai, Japan, 2001), Lecture Notes in Computer Science, Springer.

[5] Abadi, M., and Rogaway, P. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Proc. First IFIP International Conference on Theoretical*

Computer Science (Sendai, Japan, 2000), no. 1872 in Lecture Notes in Computer Science, Springer-Verlag, pp. 3–22.

[6] Atallah, M. J., Ed. *Algorithms and Theory of Computation Handbook.* CRC Press LLC, 1999, ch. 24, pp. 19–28.

[7] Backes, M., Pfitzmann, B., and Waidner, M. Reactively secure signature schemes. In *Proceedings of 6th Information Security Conference* (2003), vol. 2851 of *Lecture Notes in Computer Science*, Springer, pp. 84–95.

[8] Backes, M., Pfitzmann, B., and Waidner, M. A general composition theorem for secure reactive systems. In *Proceedings of 1st Theory of Cryptography Conference* (2004), vol. 2951 of *Lecture Notes in Computer Science*, Springer.

[9] Bellantoni, S. *Predicative Recursion and Computational Complexity.* PhD thesis, University of Toronto, 1992.

[10] Bellare, M., Desai, A., Pointcheval, D., and Rogaway, P. Relations among notions of security for public-key encrytion schemes. In *Proc. CRYPTO 1998* (Santa Barbara, California, 1998), H. Krawczyk, Ed., vol. 1462 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 26–45.

[11] Bernardo, M., and Gorrieri, R. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science 202* (1998), 1–54.

[12] Boneh, D. The decision Diffie-Hellman problem. *Proceedings of the Third Algorithmic Number Theory Symposium 1423* (1998), 48–63.

[13] Burrows, M., Abadi, M., and Needham, R. A logic of authentication. *Proceedings of the Royal Society, Series A 426*, 1871 (1989), 233–271. Also appeared as SRC Research Report 39 and, in a shortened form, in ACM Transactions on Computer Systems 8, 1 (February 1990), 18-36.

[14] Canetti, R. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology 13*, 1 (2000), 143–202.

[15] Canetti, R. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symp. on the Foundations of Computer Science* (2001), IEEE. Full version available at `http://eprint.iacr.org/2000/067/`.

[16] Canetti, R., and Krawczyk, H. Universally composable notions of key exchange and secure channels. Cryptology ePrint Archive, Report 2002/059, 2002. `http://eprint.iacr.org/`.

[17] Canetti, R., and Rabin, T. Universal composition with joint state. Cryptology ePrint Archive, Report 2002/047, 2002. `http://eprint.iacr.org/`.

[18] Canetti, R., and Rabin, T. Universal composition with joint state. In *Proc. CRYPTO 2003* (Santa Barbara, California, 2003), D. Boneh, Ed., vol. 2729 of *Lecture Notes in Computer Science*, Springer, pp. 265–281.

[19] Cervesato, I., Durgin, N. A., Lincoln, P. D., Mitchell, J. C., and Scedrov, A. A meta-notation for protocol analysis. In *12th IEEE Computer Security Foundations Workshop* (1999), IEEE Computer Society Press.

[20] Cramer, R., and Shoup, V. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. *Lecture Notes in Computer Science 1462* (1998), 13–25.

[21] de Alfaro, L. Temporal logics for the specification of performance and reliability. In *STACS '97* (1997), vol. 1200 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 165–176.

[22] Desharnais, J., Edalat, A., and Panangaden, P. Bisimulation for labelled Markov processes. *Information and Computation 179*, 2 (2002), 163–193.

[23] Desharnais, J., Gupta, V., Jagadeesan, R., and Panangaden, P. Approximating labeled Markov processes. In *Logic in Computer Science* (2000), pp. 95–106.

[24] Diffie, W., and Hellman, M. E. New directions in cryptography. *IEEE Transactions on Information Theory 22* (November 1976), 644–654.

[25] Dolev, D., Dwork, C., and Naor, M. Non-malleable cryptography (extended abstract). In *Proc. 23rd Annual ACM Symposium on the Theory of Computing* (1991), pp. 542–552.

[26] Dolev, D., and Yao, A. C.-C. On the security of public-key protocols. In *Proc. 22nd Annual IEEE Symp. Foundations of Computer Science* (1981), pp. 350–357.

[27] Durgin, N. A., Mitchell, J. C., and Pavlovic, D. A compositional logic for protocol correctness. In *14th IEEE Computer Security Foundations Workshop* (Cape Breton, Nova Scotia, Canada, June 2001).

[28] El Gamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory 31* (1985), 469–472.

[29] Gennaro, R. An improved pseudo-random generator based on discrete log. In *Proc. CRYPTO 2000* (Santa Barbara, California, 2000), vol. 1880 of *Lecture Notes in Computer Science*, Springer, pp. 469–481. Revised version available at `www.research.ibm.com/people/r/rosario/`.

[30] Goldreich, O. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Springer-Verlag, 1999.

[31] Goldreich, O. *The Foundations of Cryptography*, vol. 1. Cambridge University Press, June 2001.

[32] Goldreich, O. *The Foundations of Cryptography*, vol. 2. June 2003. Manuscript under preparation; latest version available at `http://www.wisdom.weizmann.ac.il/~oded/foc-vol2.html`.

[33] Goldwasser, S., and Bellare, M. *Lecture Notes on Cryptography.* 2003. Lecture notes for a class taught by the authors at the MIT (1996–2001); available online at `http://www.cs.nyu.edu/courses/fall01/G22.3033-003/`.

[34] Goldwasser, S., and Micali, S. Probabilistic encryption. *Journal of Computer and System Sciences 28*, 2 (1984), 270–299. Previous version in *STOC 1982*.

[35] Hansson, H. *Time and Probabilities in Formal Design of Distributed Systems.* Real-Time Safety Critical Systems. Elsevier, 1994.

[36] Hansson, H., and Jonsson, B. A framework for reasoning about time and reliability. In *Proc. of Real Time Systems Symposium* (1989), IEEE, pp. 102–111.

[37] Hofmann, M. *Type Systems for Polynomial-Time Computation.* Habilitation Thesis, Darmstadt; see `www.dcs.ed.ac.uk/home/mxh/papers.html`, 1999.

[38] Huth, M., and Kwiatkowska, M. Z. Quantitative analysis and model checking. In *LICS '97* (1997), pp. 111–122.

[39] Kemmerer, R. A., Meadows, C., and Millen, J. K. Three systems for cryptographic protocol analysis. *Journal of Cryptology 7*, 2 (1994), 79–130.

[40] Larsen, K. G., and Skou, A. Bisimulation through probabilistic testing. *Information and Computation 94*, 1 (1991), 1–28.

[41] Lincoln, P. D., Mitchell, J. C., Mitchell, M., and Scedrov, A. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security* (San Francisco, California, 1998), M. K. Reiter, Ed., ACM Press, pp. 112–121.

[42] Lincoln, P. D., Mitchell, J. C., Mitchell, M., and Scedrov, A. Probabilistic polynomial-time equivalence and security protocols. In *Formal Methods World Congress, vol. I* (Toulouse, France, 1999), J. M. Wing, J. Woodcock, and J. Davies, Eds., no. 1708 in Lecture Notes in Computer Science, Springer, pp. 776–793.

[43] Lowe, G. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In *2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems* (1996), T. Margaria and B. Steffen, Eds., vol. 1055 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 147–166.

[44] Luby, M. *Pseudorandomness and Cryptographic Applications.* Princeton Computer Science Notes. Princeton University Press, 1996.

[45] Mateus, P., Mitchell, J. C., and Scedrov, A. Composition of cryptographic protocols in a probabilistic polynomial-time process calculus. In *14th International Conference on Concurrency Theory* (Marseille, France, 2003), R. M. Amadio and D. Lugiez, Eds., vol. 2761 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 327–349.

[46] Meadows, C. Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches. In *Proc. European Symposium On Research In Computer Security* (1996), Springer Verlag, pp. 351–364.

[47] Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. *Handbook of Applied Cryptography.* CRC Press, 2001.

[48] Micali, S., Rackoff, C., and Sloan, B. The notion of security for probabilistic cryptosystems. *SIAM Journal of Computing 17* (1988), 412–426.

[49] Milner, R. *A Calculus of Communicating Systems.* Springer, 1980.

[50] Milner, R. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 1989.

[51] Mitchell, J. C., Mitchell, M., and Scedrov, A. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th Annual IEEE Symposium on the Foundations of Computer Science* (Palo Alto, California, 1998), IEEE, pp. 725–733.

[52] Mitchell, J. C., Mitchell, M., and Stern, U. Automated analysis of cryptographic protocols using Murφ. In *Proc. IEEE Symposium on Security and Privacy* (1997), pp. 141–151.

[53] Mitchell, J. C., Ramanathan, A., Scedrov, A., and Teague, V. A probabilistic polynomial-time calculus for the analysis of cryptographic protocols (preliminary report). In *17th Annual Conference on the Mathematical Foundations of Programming Semantics, Arhus, Denmark* (May 2001), S. Brookes and M. Mislove, Eds., vol. 45, Electronic notes in Theoretical Computer Science.

[54] Needham, R., and Schroeder, M. Using encryption for authentication in large networks of computers. *Communications of the ACM 21*, 12 (1978), 993–999.

[55] Paulson, L. C. Mechanized proofs for a recursive authentication protocol. In *10th IEEE Computer Security Foundations Workshop* (1997), pp. 84–95.

[56] Paulson, L. C. Proving properties of security protocols by induction. In *10th IEEE Computer Security Foundations Workshop* (1997), pp. 70–83.

[57] Pfitzmann, B., and Waidner, M. Composition and integrity preservation of secure reactive systems. In *7th ACM Conference on Computer and Communications Security* (Athens, November 2000), ACM Press, pp. 245–254. Preliminary version: IBM Research Report RZ 3234 (# 93280) 06/12/00, IBM Research Division, Zürich, June 2000.

[58] Pfitzmann, B., and Waidner, M. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy* (Washington, 2001), pp. 184–200.

[59] Pnueli, A. Linear and branching structures in the semantics and logics of reactive systems. In *Proceedings of the 12th International Colloquium on Automata, Languages, and Programming (ICALP)* (Napflion, Greece, 1985), W. Brauer, Ed., vol. 194 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 15–32.

[60] Ramanathan, A., Mitchell, J. C., Scedrov, A., and Teague, V. Probabilistic bisimulation and equivalence for security analysis of network protocols. In *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004* (Barcelona, Spain, 2004), I. Walukiewicz, Ed., vol. 2987 of *Lecture Notes in Computer Science*, Springer, pp. 468–483.

[61] Roscoe, A. W. Modeling and verifying key-exchange protocols using CSP and FDR. In *CSFW 8* (1995), IEEE Computer Society Press, p. 98.

[62] Ryan, P. Y. A., and Schneider, S. A. An attack on a recursive authentication protocol—A cautionary tale. *Information Processing Letters 65*, 1 (1998), 7–10.

[63] Schneider, S. Security properties and CSP. In *IEEE Symposium on Security and Privacy* (Oakland, California, 1996).

[64] Tsiounis, Y., and Yung, M. On the security of El Gamal-based encryption. *Lecture Notes in Computer Science 1431* (1998), 117–134.

[65] van Glabbeek, R. J., Smolka, S. A., and Steffen, B. Reactive, generative, and stratified models of probabilistic processes. *International Journal on Information and Computation 121*, 1 (August 1995).

[66] Yao, A. C.-C. Protocols for secure computations. In *Proc. 23rd IEEE Symp. on the Foundations of Computer Science* (1982), IEEE, pp. 160–164.

[67] Yao, A. C.-C. Theory and applications of trapdoor functions. In *IEEE Foundations of Computer Science* (1982), pp. 80–91.

Appendix A. Proof of Lemma 3.27

We start with a little notation. Given a process P and a set of processes $\mathfrak{R}$, we will write $\mathrm{Prob}\big[P \xrightarrow{\alpha} \mathfrak{R}\big]$ for $\sum_{R\in\mathfrak{R}} \mathrm{Prob}\big[P \xrightarrow{\alpha} R\big]$ *i.e.*, the total probability that P can take an α-transition into one of the processes in $\mathfrak{R}$.

The following lemma will also prove useful.

Lemma A.1. *Let P and Q be blocked processes such that $P \simeq Q$. Then for all public α we have $N(P,\alpha) = N(Q,\alpha)$.*

Proof. We assume that $N(P,\alpha) < N(Q,\alpha)$ in order to obtain a contradiction. An examination of the definition of N (Defn. 3.12) reveals that $N(P,\alpha)$ counts the number of transition labelled by α that P could possibly take. Thus if $N(P,\alpha) < N(Q,\alpha)$, it must be that some public $\beta \sim \alpha$ must occur more often in Q than in P.

Since we define $\simeq$ by quantifying over all contexts and schedulers, in order to achieve the contradiction with the assumption that $P \simeq Q$ it suffices to produce a scheduler S under which $\mu(P,\beta,\mathfrak{R},S) \neq \mu(Q,\beta,\mathfrak{R},S)$. Consider the scheduler that only schedules actions β equivalent to α if the transition set contains β. If P has m occurrences of actions equivalent to α, after m transitions P will reduce to a process that cannot take something equivalent to α, but after m transitions Q will reduce to a process that can still take something equivalent to α. Since α-transitions are public, P cannot be bisimilar to Q whence the desired contradiction is obtained. □

Corollary A.2. *Let $P_1 \simeq P_2$ be processes such that $P_1 \mid Q$ and $P_2 \mid Q$ are blocked processes. Then for all public α we have $N(P_1 \mid Q,\alpha) = N(P_2 \mid Q,\alpha)$.*

Proof. We note that $N(P_1 \mid Q,\alpha) = N(P_1,\alpha) + N(Q,\alpha) + \hat{N}(P_1 \mid Q,\alpha)$. Furthermore $\hat{N}(P_1 \mid Q,\alpha)$ is defined purely in terms of $N(P_1,\beta)$ and $N(Q,\beta)$ (see Defn. 3.12). The rest follows by applying Lemma A.1. □

Lemma 3.27. *Given processes P_1, P_2, and Q such that $P_1 \simeq P_2$, we have that $\forall\mathfrak{R} \in Proc/_{\simeq}.\forall S \in Sched.\forall\alpha \in Act$:*

(1) $\mu(P_1 \mid Q,\alpha,\mathfrak{R},S) = \mu(P_2 \mid Q,\alpha,\mathfrak{R},S)$*, and,*
(2) $\mu(Q \mid P_1,\alpha,\mathfrak{R},S) = \mu(Q \mid P_2,\alpha,\mathfrak{R},S)$*.*

The proof in an induction on the maximum length of α-paths from $P_i \mid Q$ into $\mathfrak{R}$. The basis is established by a case-analysis that distinguishes between actions that occur 'across' the $|$-operator (*i.e.*, that affect both $P_i \mid Q$) and actions that affect only one side of the $|$.

Proof. We will just show that

$$\mu(P_1 \mid Q,\alpha,\mathfrak{R},S) = \mu(P_2 \mid Q,\alpha,\mathfrak{R},S) \tag{$*$}$$

since the proof of the other claim is similar. We start by simplifying the problem. We note that no transition can eliminate the parallel composition operator $|$ from a process. Therefore, $\mu(P_1 \mid Q,\alpha,\mathfrak{R}) > 0$ iff $\mathfrak{R}$ contains processes of the form $R_1 \mid R_2$. Hence, to prove $(*)$ it suffices to show

$$\mu(P_1 \mid Q,\alpha,\mathfrak{R}_1 \mid \mathfrak{R}_2,S) = \mu(P_2 \mid Q,\alpha,\mathfrak{R}_1 \mid \mathfrak{R}_2,S) \tag{✠}$$

To show (✠) in the case that P_1, P_2, and Q are blocked, it suffices to demonstrate that

$$\mu(P_1 \mid Q,\alpha,\mathfrak{R}_1 \mid \mathfrak{R}_2,S) = p \implies \mu(P_2 \mid Q,\alpha,\mathfrak{R}_1 \mid \mathfrak{R}_2,S) = p$$

At this point we begin a case analysis depending on whether $P_1 \mid Q$ can take a transition equivalent to α to reach a process in the equivalence class of processes $\mathfrak{R}_1 \mid \mathfrak{R}_2$. The first case is when $\mu(P_1 \mid Q,\alpha,\mathfrak{R}_1 \mid \mathfrak{R}_2,S) = 0$ *i.e.*, α does not take $P_1 \mid Q$ into $\mathfrak{R}_1 \mid \mathfrak{R}_2$. Then either P_1 fails to reach $\mathfrak{R}_1$ or Q fails to reach $\mathfrak{R}_2$. From the definition of $\simeq$ we have that $\mu(P_1,\beta,\mathfrak{R}_1,S) = \mu(P_2,\beta,\mathfrak{R}_1,S)$. Therefore, either P_2 must fail to reach $\mathfrak{R}_1$ or Q must fail to reach $\mathfrak{R}_2$. Hence, we can conclude that $\mu(P_2 \mid Q,\alpha,\mathfrak{R}_1 \mid \mathfrak{R}_2,S) = 0$.

The second case is when $\mu(P_2 \mid Q, \alpha, \mathfrak{R}_1 \mid \mathfrak{R}_2, S) > 0$ *i.e.*, α does take $P_1 \mid Q$ into $\mathfrak{R}_1 \mid \mathfrak{R}_2$. We will prove this case by an induction on k, the maximum length of paths from $P_1 \mid Q$ into $\mathfrak{R}_1 \mid \mathfrak{R}_2$. The basis ($k = 1$) will be established by a case analysis. We will write μ_k to denote the cumulative probability distribution function assuming path lengths not exceeding k. We will differentiate the case where α, the action in question, occurs only on one side of the parallel composition from the case where α affects both sides (as the reader will recall, the probability of both components in a parallel communication taking an action is computed differently from the probability of just one component taking an action).

(1) We assume that the action α affects both parties in the parallel composition. Furthermore, we need only consider the case where P_1 can take a β-transition into $\mathfrak{R}_1$ and Q can take a γ-transition into $\mathfrak{R}_2$ with $\beta \cdot \gamma \sim \alpha$. To see why consider what happens when $P_1 \mid Q$ takes an α-transition into $\mathfrak{R}_1$. Then, since no transition can eliminate $|$-operators, there must exist processes R_{1a} and R_{1b} such that P_1 takes a β-transition to $[R_{1a}]_{\simeq}$ and Q takes a γ-transition to $[R_{1b}]_{\simeq}$ with:
   (a) $\beta \cdot \gamma \sim \alpha$; and,
   (b) $[R_{1a} \mid R_{1b}]_{\simeq} = \mathfrak{R}_1$.
   But then

$$\mu(P_1 \mid Q, \alpha, \mathfrak{R}_1 \mid \mathfrak{R}_2, S) = \mu(P_1 \mid Q, \alpha, [R_{1a}]_{\simeq} \mid [R_{1b} \mid \mathrm{rep}\, \mathfrak{R}_2]_{\simeq}, S)$$

   and we can work with $\mathfrak{R}_1 = [R_{1a}]_{\simeq}$ and $\mathfrak{R}_2 = [R_{1b} \mid \mathrm{rep}\, \mathfrak{R}_2]_{\simeq}$. Similarly, we can eliminate from consideration the case where $P_1 \mid Q$ takes an α-transition into $\mathfrak{R}_2$.

   To continue, let $\alpha \sim \tau$. Then

$$\mu_1(P_1 \mid Q, \tau, \mathfrak{R}_1 \mid \mathfrak{R}_2, S) = \mu_1(P_1, \tau, \mathfrak{R}_1, S) \cdot \mu_1(Q, \tau, \mathfrak{R}_2, S)$$

   Since we assume that the maximum length of silent paths into $\mathfrak{R}_1 \mid \mathfrak{R}_2$ is 1, we can continue as follows:

$$\begin{aligned} &= \mu_1(P_2, \tau, \mathfrak{R}_1, S) \cdot \mu_1(Q, \tau, \mathfrak{R}_2, S) \\ &= \mu_1(P_2 \mid Q, \tau, \mathfrak{R}_1 \mid \mathfrak{R}_2, S) \end{aligned}$$

   So much for the case that α is a silent action. If α is not silent, it must be equivalent to $\texttt{in}\langle c, a\rangle \cdot \texttt{out}\langle c, a\rangle$ since we assume that α affects both P_1 and Q. To aid succinctness we will write $\mathfrak{R}^P$, with $\mathfrak{R} \in \mathit{Proc}/_{\simeq}$, to denote the set $\{R \in \mathfrak{R} | \operatorname{Prob}[P \xrightarrow{\alpha} R] > 0\}$. Then $\mu_1(P_1 \mid Q, \alpha, \mathfrak{R}_1 \mid \mathfrak{R}_2, S)$ is

$$\frac{1}{N(P_1 \mid Q, \alpha)} \cdot \operatorname{Prob}\big[S(Trans(P_1 \mid Q)) = [\alpha]\big] \cdot \\ \Bigg(\sum_{R_1 \in \mathfrak{R}_1{}^{P_1}} \operatorname{Prob}\big[P_1 \xrightarrow{\mathtt{in}\langle c,a\rangle} R_1\big] \cdot N(P_1, \mathtt{in}\langle c,a\rangle) \cdot \\ \sum_{R_2 \in \mathfrak{R}_2{}^{Q}} \operatorname{Prob}\big[Q \xrightarrow{\mathtt{out}\langle c,a\rangle} R_2\big] \cdot N(Q, \mathtt{out}\langle c,a\rangle) + \\ \sum_{R_1 \in \mathfrak{R}_1{}^{P_1}} \operatorname{Prob}\big[P_1 \xrightarrow{\mathtt{out}\langle c,a\rangle} R_1\big] \cdot N(P_1, \mathtt{out}\langle c,a\rangle) \cdot \\ \sum_{R_2 \in \mathfrak{R}_2{}^{Q}} \operatorname{Prob}\big[Q \xrightarrow{\mathtt{in}\langle c,a\rangle} R_2\big] \cdot N(Q, \mathtt{in}\langle c,a\rangle) \Bigg)$$

The reader will notice that we only sum over the processes in $\mathfrak{R}_1$ reachable from P_1 via an input a on the channel c and the processes in $\mathfrak{R}_2$ reachable from Q via an output a on the channel c. Since the only actions that affect both parties combined with a $|$ are actual actions on the channel c, the only actions that the two participants can take are an input and an output respectively. Hence we do not need to some over actions equivalent to $\mathtt{in}\langle c,a\rangle$ and $\mathtt{out}\langle c,a\rangle$.

Rearranging terms we get

$$\frac{1}{N(P_1 \mid Q, \alpha)} \cdot \operatorname{Prob}\big[S(Trans(P_1 \mid Q)) = [\alpha]\big] \cdot \\ \Bigg(\frac{N(P_1, \mathtt{in}\langle c,a\rangle) \cdot N(Q, \mathtt{out}\langle c,a\rangle)}{\operatorname{Prob}\big[S(Trans(P_1)) = [\mathtt{in}\langle c,a\rangle]\big] \cdot \operatorname{Prob}\big[S(Trans(Q)) = [\mathtt{out}\langle c,a\rangle]\big]} \cdot \\ \mu_1(P_1, \mathtt{in}\langle c,a\rangle, \mathfrak{R}_1, S) \cdot \mu_1(Q, \mathtt{out}\langle c,a\rangle, \mathfrak{R}_2, S) \\ + \\ \frac{N(P_1, \mathtt{out}\langle c,a\rangle) \cdot N(Q, \mathtt{in}\langle c,a\rangle)}{\operatorname{Prob}\big[S(Trans(P_1)) = [\mathtt{out}\langle c,a\rangle]\big] \cdot \operatorname{Prob}\big[S(Trans(Q)) = [\mathtt{out}\langle c,a\rangle]\big]} \cdot \\ \mu_1(P_1, \mathtt{out}\langle c,a\rangle, \mathfrak{R}_1, S) \cdot \mu_1(Q, \mathtt{in}\langle c,a\rangle, \mathfrak{R}_2, S) \Bigg) \quad (\dagger)$$

Since $Trans(P_1 \mid Q) \sim Trans(P_2 \mid Q)$[12], it immediately follows for each α in *Act* that $\operatorname{Prob}\big[S(Trans(P_1 \mid Q)) = [\alpha]\big] = \operatorname{Prob}\big[S(Trans(P_2 \mid Q)) = [\alpha]\big]$. From Corollary A.2 we have that $N(P_1 \mid Q, \alpha) = N(P_2 \mid Q, \alpha)$ and

[12]This is easy to see since the actions available to $P \mid Q$ are actions of P, actions of Q, and actions constructed from input (resp. output) actions of P and output (resp. input) actions of Q. In particular, from $P_1 \simeq P_2$ it follows that

$$\mu_1(P_1, \alpha, \mathfrak{R}_1, S) = \mu_1(P_2, \alpha, \mathfrak{R}_1, S)$$

Therefore $[\alpha]_\sim \in Trans(P_1)$ iff $[\alpha]_\sim \in Trans(P_2)$ *i.e.*, $Trans(P_1) \sim Trans(P_2)$. Thus, from the way we construct the transition sets for processes constructed with $|$, we have that $Trans(P_1 \mid Q) \sim Trans(P_2 \mid Q)$.

from Lemma A.1 we have that $N(P_1, \alpha) = N(P_2, \alpha)$ for any $\alpha \in \mathit{Act}$. This allows us to substitute P_2 for P_1 in (†) and obtain

$$\frac{1}{N(P_2 \mid Q, \alpha)} \cdot \mathrm{Prob}\big[S(\mathit{Trans}(P_2 \mid Q)) = [\alpha]\big] \cdot$$

$$\Bigg(\frac{N(P_2, \mathtt{in}\langle c,a\rangle) \cdot N(Q, \mathtt{out}\langle c,a\rangle)}{\mathrm{Prob}\big[S(\mathit{Trans}(P_2)) = [\mathtt{in}\langle c,a\rangle]\big] \cdot \mathrm{Prob}\big[S(\mathit{Trans}(Q)) = [\mathtt{out}\langle c,a\rangle]\big]} \cdot$$

$$\mu_1(P_2, \mathtt{in}\langle c,a\rangle, \mathfrak{R}_1, S) \cdot \mu_1(Q, \mathtt{out}\langle c,a\rangle, \mathfrak{R}_2, S)$$

$$+$$

$$\frac{N(P_2, \mathtt{out}\langle c,a\rangle) \cdot N(Q, \mathtt{in}\langle c,a\rangle)}{\mathrm{Prob}\big[S(\mathit{Trans}(P_2)) = [\mathtt{out}\langle c,a\rangle]\big] \cdot \mathrm{Prob}\big[S(\mathit{Trans}(Q)) = [\mathtt{out}\langle c,a\rangle]\big]} \cdot$$

$$\mu_1(P_2, \mathtt{out}\langle c,a\rangle, \mathfrak{R}_1, S) \cdot \mu_1(Q, \mathtt{in}\langle c,a\rangle, \mathfrak{R}_2, S)\Bigg)$$

which is just $\mu_1(P_2 \mid Q, \alpha, \mathfrak{R}_1 \mid \mathfrak{R}_2, S)$.

(2) We assume that the action α only affects one party of the parallel composition. We will tackle the case where α only affects P_1 since the other case follows by a similar argument.

Here we do not need to consider the case where P_1 takes an α-transition into $\mathfrak{R}_1 \mid \mathfrak{R}_2$. If it did, it would mean that P_1 is of the form $P_1' \mid P_2'$ such that P_1' takes a β-transition into $\mathfrak{R}_1$ and P_2' takes a γ-transition into $\mathfrak{R}_2$ with $\beta \cdot \gamma \sim \alpha$. But then, since no transition can lose |-operators, it would be impossible for $P_1 \mid Q$ to take the α-transition into $\mathfrak{R}_1 \mid \mathfrak{R}_2$ since the process fragment $\mid Q$ would be lost along the way.

Let $\alpha \sim \tau$. We note that Q cannot have a silent action since we assume that α only affects P_1. Since we do not normalize the probabilities of silent actions when combining processes via a |, we have

$$\begin{aligned}\mu_1(P_1 \mid Q, \alpha, \mathfrak{R}_1 \mid Q, S) &= \mu_1(P_1, \tau, \mathfrak{R}_1, S)\\ &= \mu_1(P_2, \tau, \mathfrak{R}_1, S)\\ &= \mu_1(P_2 \mid Q, \alpha, \mathfrak{R}_1 \mid Q, S)\end{aligned}$$

Now we assume that $\alpha \nsim \tau$. Again Q cannot have silent actions because they would preempt the α that affects P_1. Our proof of this case is via a simple calculation that relies on one fact:

$$\mathrm{Prob}\big[S(\mathit{Trans}(P_1 \mid Q)) = [\alpha]\big] = \mathrm{Prob}\big[S(\mathit{Trans}(P_2 \mid Q)) = [\alpha]\big]$$

We now proceed with the actual calculation. Since we are attempting to prove the basis ($k = 1$), $\mu_1(P_1 \mid Q, \alpha, \mathfrak{R} \mid Q, S)$ is just

$$\mathrm{Prob}\big[S(\mathit{Trans}(P_1 \mid Q)) = [\alpha]\big] \cdot \sum_{R \in \mathfrak{R}^{P_1}} \mathrm{Prob}\big[P_1 \mid Q \xrightarrow{\beta} R \mid Q\big]$$

Rearranging terms as in case 1, we get

$$\frac{N(P_1, \alpha) \cdot \mathrm{Prob}\big[S(\mathit{Trans}(P_1 \mid Q)) = [\alpha]\big]}{N(P_1 \mid Q, \alpha) \cdot \mathrm{Prob}\big[S(\mathit{Trans}(P_1)) = [\alpha]\big]} \cdot \mu_1(P_1, \alpha, \mathfrak{R}, S) \qquad (‡)$$

But we know that

$$\forall \alpha \in \mathit{Act}\colon \mathrm{Prob}\big[S(\mathit{Trans}(P_1 \mid Q)) = [\alpha]\big] = \mathrm{Prob}\big[S(\mathit{Trans}(P_2 \mid Q)) = [\alpha]\big]$$

Also from Corollary A.2 we have that $N(P_1 \mid Q, \alpha) = N(P_2 \mid Q, \alpha)$ and from Lemma A.1 we have that $N(P_1, \alpha) = N(P_2, \alpha)$ for any $\alpha \in \mathit{Act}$. This allows us to substitute P_2 for P_1 in (‡) and obtain

$$\frac{N(P_2,\alpha)\cdot \mathrm{Prob}\big[S(\mathit{Trans}(P_2 \mid Q)) = [\alpha]\big]}{N(P_2 \mid Q,\alpha)\cdot \mathrm{Prob}\big[S(\mathit{Trans}(P_2)) = [\alpha]\big]} \cdot \mu_1(P_2, \alpha, \mathfrak{R}, S)$$

which is just $\mu_1(P_2 \mid Q, \alpha, \mathfrak{R} \mid Q, S)$.

Since the case analysis is exhaustive, we can assume the statement of the Lemma for $k \leq m$. We note that if $k = m+1$ then $\mu(P_1 \mid P_2, \alpha, \mathfrak{R}_1 \mid \mathfrak{R}_2, S)$ is exactly

$$\sum_{\mathfrak{T}_1 \mid \mathfrak{T}_2 \in \mathit{Proc}/_{\simeq}} \mu_m(P_1 \mid P_2, \tau, \mathfrak{T}_1 \mid \mathfrak{T}_2, S) \cdot \mu_1(\mathrm{rep}\,\mathfrak{T}_1 \mid \mathfrak{T}_2, \alpha, \mathfrak{R}_1 \mid \mathfrak{R}_2, S)$$

To complete the proof we just apply the inductive hypothesis. □

Appendix B. Proof of Lemma 3.28

Lemma B.1. *Let $P \simeq Q$ with P and Q being blocked processes. Then $|\mathit{Sep}_c(P)| = |\mathit{Sep}_c(Q)|$.*

Proof. We note that $N(P, \mathtt{in}\langle c, a\rangle)$ (resp. $N(P, \mathtt{in}\langle c, a\rangle)$) counts the number of ways that P can take a transition labelled by $\mathtt{in}\langle c, a\rangle$ (resp. $\mathtt{out}\langle c, a\rangle$). So, the sum over $a \in \mathbb{N}$ of $N(P, \mathtt{out}\langle c, a\rangle)$ counts the number of outputs in P that can be taken. To compute the number of inputs in P that can be taken we simply compute $N(P, \mathtt{in}\langle c, a\rangle)$ for some particular a—since an input can receive any value, the number of inputs receiving, say, 0 is the number of inputs in P that are ready to go. Now $|\mathit{Sep}_c(P)|$ is a function of the number of ways that P can take actual actions on the channel c *i.e.*, $|\mathit{Sep}_c(P)|$ is either

$$\frac{\big(\sum_{a\in\mathbb{N}} N(P, \mathtt{out}\langle c, a\rangle)\big)!}{\big(\sum_{a\in\mathbb{N}} N(P, \mathtt{out}\langle c, a\rangle) - N(P, \mathtt{in}\langle c, 0\rangle)\big)!}$$

or

$$\frac{N(P, \mathtt{in}\langle c, 0\rangle)!}{\big(N(P, \mathtt{in}\langle c, 0\rangle) - \sum_{a\in\mathbb{N}} N(P, \mathtt{out}\langle c, a\rangle)\big)!}$$

depending on whether the number of outputs or inputs are greater. Since $P \simeq Q$ we can use Lemma A.1 to obtain that $N(P, \mathtt{in}\langle c, 0\rangle) = N(Q, \mathtt{in}\langle c, 0\rangle)$ and that $\forall a \in \mathbb{N}\colon N(P, \mathtt{out}\langle c, a\rangle) = N(Q, \mathtt{out}\langle c, a\rangle)$. Whence $|\mathit{Sep}_c(P)| = |\mathit{Sep}_c(Q)|$. □

Lemma 3.28. *Let P, Q be processes such that $P \simeq Q$. Then,*

$$\forall \mathfrak{R} \in \mathit{Proc}/_{\simeq}.\forall S \in \mathit{Sched}.\forall \alpha \in \mathit{Act}\colon \mu(\nu_c(P), \alpha, \mathfrak{R}, S) = \mu(\nu_c(Q), \alpha, \mathfrak{R}, S)$$

Proof. We start by observing that $\mu(\nu_c(P), \alpha, \mathfrak{R}, S) = \mu(\nu_c(Q), \alpha, \mathfrak{R}, S) = 0$ whenever there exists $a \in \mathbb{N}$ such that either $\mathtt{in}\langle c, a\rangle \in \mathrm{car}\,\alpha$ or $\mathtt{out}\langle c, a\rangle \in \mathrm{car}\,\alpha$. Thus, we need only consider actions that do not happen on the channel c. Similarly, We also note that

(1) Every scheduler has to schedule all private actions before any public actions are scheduled. Hence, every path in the process graph for $\nu_c(P)$ has to schedule public actions on the channel c which are now private before it schedules any other public actions.
(2) Also, any path must schedule a c-action (*i.e.*, an actual action on the channel c) simultaneously with other private actions if a c-action is available (more on this when we establish the basis).

We will proceed with an induction on the maximum length of α-paths from $\nu_c(P)$ into $\mathfrak{R}$ and show that $\mu(\nu_c(P), \alpha, \mathfrak{R}, S) = p \implies \mu(\nu_c(Q), \alpha, \mathfrak{R}, S) = p$. We will write μ_k to indicate that we are assuming that the maximum α-path length in the graph for P is k. The basis occurs when the maximum length of an α-path in P is 1. Let us assume that $\alpha \sim \tau$. A private action of $\nu_c(P)$ is a private action of P occurring simultaneously with all c-actions of P that can go simultaneously. This means that $\nu_c(P)$ can take a τ-action to $\nu_c(R)$ with non-zero probability iff there exists a $2k$-holed c-separator $C[\ \ ] \in \mathit{Sep}_c(P)$ and $2k$ processes $R_1, \ldots, R_{2k}$ such that

(1) $C[R_1, \ldots, R_{2k}] \equiv P$
(2) $\mathrm{Prob}\big[C[\oslash, \ldots, \oslash] \xrightarrow{\tau} C'[\oslash, \ldots, \oslash]\big] > 0$,
(3) We have for each even i with $0 < i \leq 2k$ that $\mathrm{Prob}\big[C[R_1, \ldots, R_{2k}] \xrightarrow{\alpha} C[R_1, \ldots, R_{i-2}, R'_{i-i}, R'_i, R_{i+1}, \ldots, R_{2k}]\big] > 0$.

Additionally, since ν-operators are never removed by reduction or communication steps, we need only consider equivalence classes of actions containing processes of the form $\nu_c(Q)$ (the probability that $\nu_c(P)$ can take any kind of transition to a process that has no outermost ν-operator binding the channel c is zero).

Let us continue by assuming that P is blocked. We note that a c-separator must consist of holes in parallel composition with the other holes and the remainder of the process since each hole in a c-separator picks out an exposed input or an exposed output. Then, it follows that if P takes any τ-transition, the result must be of form $C'[R_1, \ldots, R_{2k}]$ where $C'[R_1, \ldots, R_{2k}]$ is reached from $C[R_1, \ldots, R_{2k}]$ via a τ-action. Since each of the R_i is an input or an output expression, every process in the equivalence class $[C[R_1, \ldots, R_{2k}]]_{\simeq}$ must be of the form $D[A_1, \ldots, A_{2k}]$ where $\forall i \in [1..2k]\colon A_i \simeq R_i$.

Then, $\mu_1(\nu_c(P), \tau, [\nu_c(C'[R'_1, \ldots, R'_{2k}])]_{\simeq}, S)$ is given by

$$\mathrm{Prob}\big[S(\mathit{Trans}(\nu_c(P))) = [\tau]\big] \cdot \left(\prod_{\{i \in \mathbb{N} \mid 0 < i \leq 2k\}} \mathrm{Prob}\big[A_i \mid A_{i-1} \xrightarrow{\alpha_i} A'_i \mid A'_{i-1}\big]\right) \cdot \sum_{\nu_c(D[A_1, \ldots, A_{2k}]) \in [\nu_c(C'[R_1, \ldots, R_{2k}])]_{\simeq}} \mathrm{Prob}\big[P \xrightarrow{\tau} D[A_1, \ldots, A_{2k}]\big]$$

But since each of the R_i is an input or an output expression, it follows that

$$\prod_{\{i \in \mathbb{N} \mid 0 < i \leq 2k\}} \mathrm{Prob}\big[A_i \mid A_{i-1} \xrightarrow{\alpha_i} A'_i \mid A'_{i-1}\big] = 1$$

Then, rearranging terms as in the proof of Lemma 3.27, we get

$$\mathrm{Prob}\big[S(\mathit{Trans}(\nu_c(P))) = [\tau]\big] \cdot \frac{N(P, \tau)}{N(\nu_c(P), \tau)} \cdot \mu_1(P, \tau, [C'[R_1, \ldots, R_{2k}]]_{\simeq}, S)$$

But $N(\nu_c(P), \tau) = N(P, \tau) \cdot |\mathit{Sep}_c(P)|$ (see Defn. 3.12) whence we get

$$\mathrm{Prob}\big[S(\mathit{Trans}(\nu_c(P))) = [\tau]\big] \cdot \frac{1}{|\mathit{Sep}_c(P)|} \cdot \mu_1(P, \tau, [C'[R_1, \ldots, R_{2k}]]_{\simeq}, S) \qquad (\dagger)$$

Since $P \simeq Q$ it follows that $\mathit{Trans}(P) \sim \mathit{Trans}(Q)$ whence $\mathrm{Prob}\big[S(\mathit{Trans}(\nu_c(P))) = [\alpha]\big] = \mathrm{Prob}\big[S(\mathit{Trans}(\nu_c(Q))) = [\alpha]\big]$. An application of Lemma B.1 gives us that

$|Sep_c(P)| = |Sep_c(Q)|$. Finally $P \simeq Q$ yields that $\mu_1(P, \alpha, \mathfrak{R}, S) = \mu_1(P, \alpha, \mathfrak{R}, S)$. So (†) can be rewritten as

$$\text{Prob}\big[S(Trans(\nu_c(Q))) = [\tau]\big] \cdot \frac{1}{|Sep_c(Q)|} \cdot \mu_1(Q, \tau, [C'[R_1, \ldots, R_{2k}]]_\simeq, S)$$

which is just $\mu_1(\nu_c(Q), \tau, [\nu_c(C'[R'_1, \ldots, R'_{2k}])]_\simeq, S)$.

If P is unblocked, then the only action that $\nu_c(P)$ can take is a reduction action. But then $\mu_1(\nu_c(P), \tau, \mathfrak{R}, S) = \mu_1(P, \tau, \mathfrak{R}, S) = \mu_1(Q, \tau, \mathfrak{R}, S) = \mu_1(\nu_c(Q), \tau, \mathfrak{R}, S)$ (since we assume that the maximum silent path length into $\mathfrak{R}$ is 1).

Now let us assume that $\alpha \not\sim \tau$. Then $\nu_c(P)$ cannot have any private actions (including any on the channel c) since they would preempt the action α. An examination of the rules of Figure 2 reveals that the ν-operator on the channel c has no effect on other channels. Whence

$$\mu_1(\nu_c(P), \alpha, \mathfrak{R}, S) = \mu_1(P, \alpha, \mathfrak{R}, S) = \mu_1(Q, \alpha, \mathfrak{R}, S) = \mu_1(\nu_c(Q), \alpha, \mathfrak{R}, S)$$

So much for the basis. We assume that the theorem holds for paths of length at most m and then establish the claim by observing that $\mu_{m+1}(\nu_c(P), \alpha, \mathfrak{R}, S)$ is exactly

$$\sum_{\mathfrak{R}' \in Proc/_\simeq} \mu_m(\nu_c(P), \tau, \mathfrak{R}', S) \cdot \mu_1(\text{rep}\, \mathfrak{R}', \alpha, \mathfrak{R}, S)$$

and deploying the inductive hypothesis. □

Appendix C. The Evaluator for Closed Expressions

We will now construct the machine $M_{\mathcal{P}}$.[13] The machine has two input tapes, two working tapes, and two output tapes. It starts with a value i for the security parameter written out in unary[14] on one input tape and a description of the Turing machine implementing a perceptible scheduler S on the other tape—we know such a Turing machine exists because a perceptible scheduler is a poly-time probabilistic function. The input tape starts with the closed expression $\mathcal{P}$ that $M_{\mathcal{P}}$ is supposed to evaluated. The first step that $M_{\mathcal{P}}$ undertakes is to substitute i for $\underline{\mathtt{N}}$ in $\mathcal{P}$. It then copies $P^{\underline{\mathtt{N}} \leftarrow i}$ onto one of the working tapes making sure to leave enough space around the terms as specified by the assumption on the size of terms (we will use the symbol ␣ as a blank symbol that we use to delete symbols by overwriting them as well as "pad out" terms so that we leave sufficient space as indicated by the discussion on Section 6.1). As we reduce $P^{\underline{\mathtt{N}} \leftarrow i}$ and perform communication steps, we will rewrite $P^{\underline{\mathtt{N}} \leftarrow i}$ on this working tape. At this point $M_{\mathcal{P}}$ is ready to begin evaluation of the process.

An evaluation step consists of a reduction step followed by a communication step. We note that, due to the idempotence of reduction (see Lemma 3.7), performing a reduction step on a blocked process does not alter the probability distribution on observables induced by $\mathcal{P}$. To perform a reduction step $M_{\mathcal{P}}$ must evaluate each exposed term and each exposed match in $P^{\underline{\mathtt{N}} \leftarrow i}$ (it is easy to determine if a term or match is exposed—every time an input operator, say $\mathtt{in}\langle c, x\rangle.Q^{\underline{\mathtt{N}} \leftarrow i}$, is

[13]The rest of this section can be safely omitted in a first reading since it merely details the intuitive construction implied by the comments and assumptions above.

[14]For technical reasons, it is convenient that the security parameter be written out in unary. In particular, the running time of a process will turn out to be $p(\underline{\mathtt{N}}) \cdot q(|\underline{\mathtt{N}}|)$ for p and q polynomials in one variable. We would like the running time to be polynomial in the size of the security parameter and so we write the security parameter in unary to ensure that the property is respected.

encountered, we consider every subexpression of $Q^{\underline{\mathrm{N}}\leftarrow i}$ as not being exposed). In order to evaluate the term T we will make use of the Turing machine M_T associated with T by the properties established by the conditions on terms given in Section 3.1. We note that no evaluation sequence (*i.e.*, an alternating sequence of reductions and communication steps) can create terms. Thus, the set of terms that could possibly be reduced during evaluation of $\mathcal{P}$ can be determined by inspecting $\mathcal{P}$. Let $Terms\colon Expr \to 2^{Term}$ be a function from closed expression to sets of terms such that $Terms(\mathcal{P})$ is the set of terms that appear as syntactic elements of $\mathcal{P}$. Since every exposed term must be a substitution instance of terms from the set $Terms(\mathcal{P})$ (recall that $\mathcal{P}$ is assumed to be closed, and communication steps create λ-substitution instances of terms), we can evaluate each substitution instance $\lambda x_1 \cdots \lambda x_k.T\, a_1 \cdots a_k$ by simply running the Turing machine M_T at inputs $a_1, \ldots, a_k$. In the case that the term evaluated appears in an output on the channel c, we write down the $\sigma(c)(i)$ least significant bits. If the term is part of a match, we use a working tape to record the entire value of the term. In this manner, we can compute a reduction step by simply invoking the appropriate Turing machines at the appropriate values and overwriting matches with blank symbols and/or the $\oslash$ process. We note that the distribution on process induced by performing a reduction step on $P^{\underline{\mathrm{N}}\leftarrow i}$ is exactly the same as that specified by the definition of the reduction function ρ (see Section 3.3) *i.e.*, $M_{\mathcal{P}}$ really does compute a reduction step in stage of evaluation.

We then need to perform a communication step. We start by computing the set of input/output pairs and silent actions that $P^{\underline{\mathrm{N}}\leftarrow i}$ can take. This can be done by simply collecting all the input and outputs that are not in the scope of other inputs and outputs, and then building the set E of all pairs of equivalent to $\mathtt{in}\langle c, a\rangle \cdot \mathtt{out}\langle c, a\rangle$ for some c and a (whether c is private or not). We augment this set of actions with locating contexts (see Section 3.10) that we will use to identify the particular inputs and outputs of a communication step. If E contains no silent actions, then we have the set of eligible actions for $\mathcal{P}$. If E has a silent action, then the eligible set consists of sets of silent actions. Each of the sets in the eligible set for $\mathcal{P}$ consists of the silent actions that one of the components of $\mathcal{P}$ can possibly take. We then use the given perceptible scheduler S to select one of the equivalence classes of the eligible set induced by $\sim$ (in the case that the eligible set consists of silent actions, we skip this step since the scheduler must elect to perform a silent action). Having selected an equivalence class, we then pick one of the actions from that equivalence class uniformly at random and perform the indicated substitution(s) if the eligible set contained no silent actions. Otherwise, we pick one silent action from each of the sets in the eligible set. This corresponds to having each component take a silent action independently (with components with no silent actions taking the trivial silent action to itself). If the action is a public one, we record the appropriate observable on the output tape—this tape will then have all the observables generated during the evaluation of the process. Clearly, at the termination of this step $M_{\mathcal{P}}$ has correctly performed an action available to $\mathcal{P}$. However, has it done so with the right probability? An examination of the inference rules of Figure 2 coupled with a blocked induction reveals that the probability of a particular input (resp. output) action is simply one divided by the number of input (resp. output) actions equivalent to it that are found in $\mathcal{P}$. Thus our decision to select the action uniformly at random from the chosen equivalence class is consistent with the probabilities induced by the rules of Figure 2. Thus the communication step performed by $M_{\mathcal{P}}$ respects the operational semantics of PPC.

Finally, $M_{\mathcal{P}}$ repeats reduction-and-communication-step pairs until there are no more actual actions to take. We note that this means that there will be a final reduction step followed by no communication step, but since that reduction step produces no observables, it can be ignored.

Our construction is specific to a particular closed expression. Thus each closed expression has an evaluator associated with it. The reader will no doubt appreciate that a single Turing machine able to evaluate any closed expression is a simple extension. The input, in this case, will simply be the expression to be evaluated, a value for the security parameter, a description of the Turing machine associated with the scheduler, and descriptions of the Turing machines associated with each of the terms in the process to be evaluated. In fact we can even extend this evaluation machine to open expressions by including in the input a valuation for the free variables of the process. We leave the details of these extensions to the interested reader.

Appendix D. Miscellaneous Proofs

D.1. **CPDFs are Well-Defined.**

Lemma 3.20.

$$\forall P \in \mathit{Proc}. \forall \alpha \in \mathit{Act}. \forall \mathfrak{R} \subseteq \mathit{Proc}. \forall S \in \mathit{Sched}\colon \mu(P, \alpha, \mathfrak{R}, S) \leq 1$$

Proof. We note that the contribution of an empty path to μ is 0 if we are computing $\mu(P, \alpha, \mathfrak{R}, S)$ with α public, and the contribution of an empty path when we compute $\mu(P, \tau, \mathfrak{R}, S)$ is 1 iff there are no positive length silent paths from P into $\mathfrak{R}$. Hence we need only check the value of μ over positive length paths. In what follows we will write $\mathit{Paths}^k(P, \alpha, \mathfrak{R})$ to be the set of α-paths from P into $\mathfrak{R}$ with length k. Similarly, $\mathit{Paths}^{\leq k}(P, \alpha, \mathfrak{R})$ denotes the set of α-paths from P into $\mathfrak{R}$ with length at most k.

We note that $\mu(P, \alpha, \mathfrak{R}, S)$ is given by

$$\mu(P, \alpha, \mathfrak{R}, S) = \sum_{\pi \in \mathit{Paths}(P, \alpha, \mathfrak{R})} \mathrm{Prob}\big[\pi_S\big]$$

We can reorder this sum as

$$\sum_{k=1}^{\infty} \left(\sum_{\pi \in \mathit{Paths}^k(P, \alpha, \mathfrak{R})} \mathrm{Prob}\big[\pi_S\big] \right)$$

It is important to consider this reordering carefully. In particular, multiple initial segments of a single path will not appear in this sum since the only silent paths we allow are paths of maximum length. Since paths are defined by their supports, and every silent path must have a maximum support, it follows that $\mathit{Paths}^k(P, \tau, \mathfrak{R})$ contains only silent paths of minimal length. Thus, the total probability of getting to $\mathfrak{R}$ via a τ-path of length at most $k - 1$ is not over-estimated since the set of paths we sum over will not contain a path π and a different path ψ such that π in a initial path of ψ.

We proceed by induction on the maximum length of α-paths from P to some member of $\mathfrak{R}$. The basis occurs when this path is at most length 1. The cumulative probability of taking an α-path from P into $\mathfrak{R}$, where the maximum length of the α-path is 1, is just

$$\left(\sum_{\pi \in Paths^{k}(P,\alpha,\mathfrak{R})} \mathrm{Prob}\big[\pi_S\big]\right) = \mathrm{Prob}\big[S(Trans(P)) = [\alpha]_{\sim}\big] \cdot \sum_{\substack{\beta\sim\alpha,\\ R\in\mathfrak{R}}} \mathrm{Prob}\big[P \xrightarrow{\beta} R\big]$$

From the definition of process graphs (Defn. 3.16) we know that

$$\sum_{\substack{\beta\sim\alpha,\\ R\in\mathfrak{R}}} \mathrm{Prob}\big[P \xrightarrow{\beta} R\big] \leq 1$$

whence it follows that the entire term has value at most 1 (since schedulers are stochastic probabilistic functions).

We take as our inductive hypothesis that the statement holds when the longest path is of length at most $k-1$. Then the cumulative probability of taking an α-path from P into $\mathfrak{R}$, where the maximum length of the α-path is k, is just the sum of the probability of taking a silent path of length at most $k-1$ and then getting to $\mathfrak{R}$ via a single α-step (see ‡) plus the probability of getting to $\mathfrak{R}$ directly via an α-step (see †). Thus, there are two terms to consider:

$$\mathrm{Prob}\big[S(Trans(P)) = [\alpha]_{\sim}\big] \cdot \sum_{\substack{\beta\sim\alpha\\ R\in\mathfrak{R}}} \mathrm{Prob}\big[P \xrightarrow{\beta} R\big] \qquad (\dagger)$$

and

$$\sum_{\pi \in Paths^{\leq k-1}(P,\tau,\mathfrak{R}')} \mathrm{Prob}\big[\pi_S\big] \cdot \mathrm{Prob}\big[S(Trans(\mathrm{rep}\,\mathfrak{R}')) = [\alpha]_{\sim}\big] \cdot \sum_{\substack{\beta\sim\alpha\\ R\in\mathfrak{R}}} \mathrm{Prob}\big[\mathrm{rep}\,\mathfrak{R}' \xrightarrow{\beta} R\big] \qquad (\ddagger)$$

We use the definition of process graphs to show that the term

$$\sum_{\substack{\beta\sim\alpha\\ R\in\mathfrak{R}}} \mathrm{Prob}\big[P \xrightarrow{\beta} R\big] \qquad (*)$$

in (†) is at most one. We then use the inductive hypothesis and the definition of process graphs to show that (‡) is at most one. But we can rewrite (‡) as

$$\mathrm{Prob}\big[S(Trans(P)) = [\tau]_{\sim}\big] \cdot \sum_{Q\in Proc} \mathrm{Prob}\big[P \xrightarrow{\tau} Q\big] \cdot \sum_{\pi \in Paths^{\leq k-2}(Q,\tau,\mathfrak{R}')} \mathrm{Prob}\big[\pi_S\big] \cdot \mathrm{Prob}\big[S(Trans(\mathrm{rep}\,\mathfrak{R}')) = [\alpha]_{\sim}\big] \cdot \sum_{\substack{\beta\sim\alpha\\ R\in\mathfrak{R}}} \mathrm{Prob}\big[\mathrm{rep}\,\mathfrak{R}' \xrightarrow{\beta} R\big] \qquad (\maltese)$$

Finally we use the fact that a scheduler is a stochastic probabilistic function to combine $(*)$ and $(\maltese)$ and, thus, establish the result. $\square$

D.2. Generation Probabilities are Well-Defined.

Lemma D.1.

$$\forall S \in \mathit{Sched}\colon \sum_{\substack{R \in \mathit{Proc} \\ \alpha}} \mu(P, \alpha, \{R\}, S) \leq 1$$

Proof. Pick an arbitrary $S \in \mathit{Sched}$. We proceed by induction on the maximum length of paths. The basis occurs when the maximum length of paths leaving P is 1. Then,

$$\sum_{\substack{R \in \mathit{Proc} \\ \alpha}} \mu(P, \alpha, \{R\}, S) = \sum_{\alpha} \mathrm{Prob}\big[S(\mathit{Trans}(P)) = [\alpha]_{\sim}\big] \cdot \sum_{\beta \sim \alpha} \mathrm{Prob}\big[P \xrightarrow{\beta} R\big] \quad (\dagger)$$

From the definition of process graphs (Defn. 3.16), we know that for any α in Act it is the case that $\sum_{\beta \sim \alpha, R \in \mathit{Proc}} \mathrm{Prob}\big[P \xrightarrow{\beta} R\big] \leq 1$. Furthermore, S is a stochastic probabilistic function (*i.e.*, the sum over all action-types of probability that S picks that action-type is either 0 or 1). Whence it follows that ($\dagger$) is at most 1.

We can, therefore, assume as our inductive hypothesis that the lemma holds for paths of at most length m. Let us denote the cPDF over paths of length at most m by the symbol μ^m. The inductive step occurs when the maximum length of paths leaving P is $m + 1$. In this case we compute

$$\sum_{\substack{R \in \mathit{Proc} \\ \alpha}} \mu^{m+1}(P, \alpha, \{R\}, S)$$

as

$$\sum_{\alpha \not\sim \tau} \mathrm{Prob}\big[S(\mathit{Trans}(P)) = [\alpha]_{\sim}\big] \cdot \sum_{\substack{\beta \sim \alpha, \\ R \in \mathit{Proc}}} \mathrm{Prob}\big[P \xrightarrow{\beta} R\big] +$$
$$\mathrm{Prob}\big[S(\mathit{Trans}(P)) = [\tau]_{\sim}\big] \cdot \sum_{U \in \mathit{Proc}} \mathrm{Prob}\big[P \xrightarrow{\tau} U\big] \cdot$$
$$\sum_{\mathfrak{R}' \in \mathit{Proc}/_{\simeq}} \mu^{m-1}(U, \tau, \mathfrak{R}', S) \cdot \mu^1(\mathrm{rep}\,\mathfrak{R}', \alpha, \mathfrak{R}, S)$$

We know that

$$\sum_{\substack{\beta \sim \alpha, \\ R \in \mathit{Proc}}} \mathrm{Prob}\big[P \xrightarrow{\beta} R\big] \leq 1$$

We then deploy the inductive hypothesis with the fact that S is a stochastic probabilistic function to complete the proof. □

Lemma 4.3. $\mathit{Prob}\big[P \rightsquigarrow_S o\big] \leq 1$.

Proof. Let $\alpha_1, \ldots, \alpha_k$, with α_i an α_i-path, be an *evaluation sequence*. We proceed via an induction on k the length of evaluation sequences. The basis occurs when P generates $o = \langle c, a\rangle$ under scheduler S via an evaluation sequence of at most length 1. Then

$$\mathrm{Prob}\big[P \rightsquigarrow_S o\big] = \sum_{\mathfrak{R} \in \mathit{Proc}/_{\simeq}} \mu(P, \mathtt{in}\langle c, a\rangle \cdot \mathtt{out}\langle c, a\rangle, \mathfrak{R}, S)$$

which, by Lemma 3.20, is at most one. This establishes the basis. We now establish the inductive hypothesis (when the length of the evaluation sequence is at most $k+1$). We compute $\mathrm{Prob}\big[P \rightsquigarrow_S o\big]$ as

$$\sum_{R \in Proc} \mu(P, \mathtt{in}\langle c,a\rangle \cdot \mathtt{out}\langle c,a\rangle, \{R\}, S) + \sum_{\substack{\{\beta \not\sim \mathtt{in}\langle c,a\rangle \cdot \mathtt{out}\langle c,a\rangle \in Act^{\times}\} \\ R \in Proc}} \mu(P, \beta, \{R\}, S) \cdot \mathrm{Prob}\big[R \rightsquigarrow_S o\big] \quad (\dagger)$$

The inductive hypothesis gives us that $\mathrm{Prob}\big[R \rightsquigarrow_S o\big] \leq 1$. Thus

$$\sum_{R \in Proc} \mu(P, \mathtt{in}\langle c,a\rangle \cdot \mathtt{out}\langle c,a\rangle, \{R\}, S) + \sum_{\substack{\{\beta \not\sim \mathtt{in}\langle c,a\rangle \cdot \mathtt{out}\langle c,a\rangle \in Act^{\times}\} \\ R \in Proc}} \mu(P, \beta, \{R\}, S) \quad (\ddagger)$$

is an upper-bound on (†). We then complete the proof by using Lemma D.1 to show that (‡) is at most 1. □

D.3. **Paths in Process Graphs are of Finite Length.**

Lemma D.2. *Let P be a closed process. Then, under any scheduler, the length of any path from root to leaf in the process graph of P is finite.*

Proof. We note that no action creates new inputs or outputs. Furthermore, each action removes at least one input or output whence no path can contain two edges labelled by the same action (using locating contexts to color actions uniquely). Consequently, since P is of finite length, the number of inputs, outputs, and input-output pairs (which include all private and actual actions) that can possibly label actions along a single path from root to leaf must be finite.

Now a node labelled by an unblocked process must only have outgoing reduction actions and a blocked process has no outgoing reduction actions. Thus, there is at most one edge labelled by a reduction action preceding any edge labelled by a non-reduction action (since actions take blocked processes to unblocked processes).

Finally, since no path can contain two edges labelled by the same action and the set of possible labelling actions is finite, the path must be of finite length. Since this proof does not depend on the scheduler's behavior, the property holds under all schedulers. □

Definition D.3. Let G be the process graph of P. The graph G' obtained by relabelling each node labelled with the process Q with $[Q]_{\simeq}$ is called the *$\simeq$-graph induced by P.*

The maximum length of paths from the root of a process graph to a leaf is a rough measure of how long it takes to evaluate the process. We will use the $\simeq$-graph to study the length of paths in a process graph.

Lemma D.4. *Let G be the $\simeq$-graph induced by P. Let π be a path from the root to a leaf. Then π does not contain a cycle.*

Proof. If π contains a cycle, there must exist two nodes on the path whose labels are in the same equivalence class $\mathfrak{R}$ under $\simeq$. Whence it must be that for some $\alpha \in \mathit{Act}$ and some scheduler S, we have that $\mu(\mathrm{rep}\,\mathfrak{R}, \alpha, \mathfrak{R}, S) > 0$. But then, under that scheduler, there must be an infinite path in the graph since any process in $\mathfrak{R}$ can return to $\mathfrak{R}$ via an α-path with non-negligible probability under S. But this contradicts Lemma D.2. $\square$