Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 353 (2006) 118–164

Theoretical Computer Science

www.elsevier.com/locate/tcs

A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols☆

John C. Mitchell[a], Ajith Ramanathan[a], Andre Scedrov[b,*,1], Vanessa Teague[a,2]

[a]Department of Computer Science, Stanford University, 353 Serra Mall, Stanford, CA 94305-9025, USA

[b]Department of Mathematics, University of Pennsylvania, 209 South 33rd Street, Philadelphia, PA 19104-6395, USA

Received 13 April 2002; received in revised form 13 October 2005; accepted 26 October 2005

Communicated by M.W. Mislove

Abstract

We prove properties of a process calculus that is designed for analysing 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 all 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 El Gamal 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.

© 2005 Elsevier B.V. All rights reserved.

Keywords: Process Algebra; Observational equivalence; Probabilistic bisimulation; Security 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.
Additional support was provided by OSB/AFOSR MURI "Cooperative Networked Control of Dynamical Peer-to-Peer Vehicle Systems" Grant.

* Corresponding author.

E-mail addresses: mitchell@stanford.edu (J.C. Mitchell), ajith@cs.stanford.edu (A. Ramanathan), scedrov@cis.upenn.edu (A. Scedrov), vteague@cs.stanford.edu (V. Teague).

[1] Additionally supported by NSF Grants CCR-9800785 and CCR-0098096.

[2] Additionally supported by a Stanford Graduate Fellowship.

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.10.044

0. Introduction

There are many methods used in the analysis of security protocols. The main systematic or formal approaches include specialised 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 [61,43,46,63,51]. 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 [53], allows a protocol adversary to nondeterministically choose among possible actions (see [19]). This convenient idealisation 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 analyse probabilistic protocols.

In this paper we describe some technical properties of a process calculus that was proposed earlier [41,42,45,50,52] 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 [59] contains material excerpted from this paper. The framework relies on a language for defining communicating polynomial-time processes [50]. The reason we restrict processes to probabilistic polynomial time is 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 analyse probabilistic as well as deterministic encryption functions and protocols. Without a probabilistic framework, it would not be possible to analyse an encryption function such as El Gamal [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 [50] and further example protocols considered in [42]. Some portions of this paper are summarised in [52,59,60]. Subsequent to the publication of [59] an error was found in the operational semantics. In fixing this problem, some rules in the proof system given in [59] were found to be false. Fortunately, none of our major results were found to be untrue. The closest, independent 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., [40,65,11]. 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]. This is an orthogonal approach since in this work expressions represent non-terminating entities. So, when two expressions are said to be asymptotically equivalent, it means that the probabilistic behaviour of the two expressions approach each other over the course of their (infinite) evaluation i.e., the two expressions converge over time.

In our setting, expressions denote families of terminating processes indexed by a security parameter, since we wish to model security protocols. Security properties are specified as observational equivalences. Specifically, $\mathcal{P} \cong \mathcal{Q}$ means that for any context $\mathcal{C}[\,\cdot\,]$, the behaviour of expression $\mathcal{C}[\mathcal{P}]$ is asymptotically (in the security parameter) computationally indistinguishable from the behaviour of expression $\mathcal{C}[\mathcal{Q}]$. If $\mathcal{P}$ is a protocol of interest, and $\mathcal{Q}$ is an idealised form of the expression 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 realised 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 [41,42,45,52] 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 before public communications. Second, we develop a form of probabilistic bisimulation that, while not a complete characterisation 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 7 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 characterisation 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.

0.1. Preliminaries

In this section we establish several basic notions that we will require throughout this monograph. We start by introducing a notion of probabilistic function tailored to our needs. Next we recapitulate standard treatments and establish notation and terminology for probabilistic Turing machines, equivalence classes and multisets.

Probabilistic functions. Let $f: X \times Y \to [0, 1]$ be a function satisfying:

(1) $\forall x \in X: \sum_{y \in Y} f(x, y) \leqslant 1$ and
(2) $\forall x \in X: |\{y \in Y \mid f(x, y) > 0\}| \in \mathbb{N}$.

Then f is a *probabilistic function from X to Y*, written as $f: X \dashrightarrow Y$. The second condition in the definition ensures that the sum in the first condition is well-defined by stipulating that only a finite number of terms in the sum in the first condition are non-zero.

We will say that the *domain* of f is the set X, the *codomain* of f is the set Y, and the *range* of f is the set $\{y \in Y \mid \exists x \in X: f(x, y) > 0\}$. We will write $f(x) \overset{p}{\dashrightarrow} y$ or $\text{Prob}\,[f(x) = y] = p$ just when $f(x, y) = p$. A *stochastic* probabilistic function $f: X \dashrightarrow Y$ is one in which $\forall x \in X: \sum_{y \in Y} f(x, y) = 1$.

The *composition* $g \circ f: X \times Z \to [0, 1]$ *of two probabilistic functions* $f: X \dashrightarrow Y$ and $g: Y \dashrightarrow Z$ is defined as the function satisfying

$$\forall x \in X. \forall z \in Z: (g \circ f)(x, z) = \sum_{y \in Y} f(x, y) \cdot g(y, z).$$

It is easy to verify that the composition $g \circ f: X \times Z \to [0, 1]$ of two probabilistic functions $f: X \dashrightarrow Y$ and $g: Y \dashrightarrow Z$ is a probabilistic function.

Probabilistic Turing machines. Our presentation follows standard treatments (see e.g., [54,6]). A *random Turing machine* (RTM) is a Turing machine with an extra random-tape and three extra states q_{rand}, q_{one} and q_{zero}. Initially, the machine starts with the input on the working tape and an infinite sequence of bits on the random-tape. When the machine enters state q_{rand}, control passes to state q_{one} if the bit to the right of the current position of the random-tape head contains a 1 and to state q_{zero} if the bit to the right contains a 0. Given a probabilistic Turing machine M we will write $M(\vec{r}, \vec{a})$ for the result of M on input $\vec{a}$ using random bits given by $\vec{r}$.

The RTM M runs in *probabilistic poly-time* if there exists a polynomial $q(\vec{x})$ such that $M(\vec{r}, \vec{a})$ halts in time $q(|\vec{a}|)$ for all sequences $\vec{r}$ of bits on the random-tape. We note that if M runs in probabilistic polynomial time, then M reads at most $q(\vec{x})$ bits of the random-tape. We can view this M as a *probabilistic poly-time Turing machine* (PPTM) if we choose the bits on the random-tape uniformly at random from the space of bitstrings that M can read in its timebound of $q(\vec{x})$. In particular, we will write that $M(\vec{a}) = a$ *with probability* p if, choosing a sequence of $q(\vec{a})$ bits uniformly at random, the probability that $M(\vec{r}, \vec{a}) = a$ is p.

We say that a PPTM M *computes a probabilistic function* $f: X\text{-->}Y$ if for all inputs $x \in X$ for all outputs $y \in Y$, we have $\text{Prob}[f(x) = y] = \text{Prob}[M(x) = y]$. A probabilistic function $f: X\text{-->}Y$ is *poly-time computable*, or just poly-time, if it is computed by a PPTM. We note that every function computed by a PPTM satisfies condition 2 in the definition of a probabilistic function and is, therefore, a probabilistic function.

Equivalence relations: An *equivalence relation R on X* is a subset R of $X \times X$ such that:

(1) R is reflexive i.e., $\forall x \in X: \langle x, x\rangle \in R$;
(2) R is transitive i.e., $\forall x, y, z \in X: \langle x, y\rangle, \langle y, z\rangle \in R \Longrightarrow \langle x, z\rangle \in R$; and
(3) R is symmetric i.e., $\forall x, y \in X: \langle x, y\rangle \in R \Longrightarrow \langle y, x\rangle \in R$.

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 rep E for a *representative element* of E i.e., any $x \in E$.

1. A probabilistic poly-time process calculus

We assume a countable set **Var** of variables, a distinguished variable η not in **Var**, a countable set **Channel** of channel names partitioned into two countable sets **Unbindable** and **Bindable**, a set $\textbf{Poly} = \{q: \mathbb{N} \to \mathbb{N} \mid \forall a \in \mathbb{N}: q(a) \geqslant 0\}$ of positive polynomials, a total function *width*: $\textbf{Channel} \to \textbf{Poly}$, and a symbol $\equiv$ denoting syntactic identity.

1.1. 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 $\hookrightarrow: \Theta \times \mathbb{N}^* \to \mathbb{N}$ called *basic term reduction*, such that:

(1) if θ is a basic term with k arguments, then there exists a probabilistic poly-time Turing machine M_θ with k inputs such that $M_\theta(a_1, \ldots, a_k)$ returns a with probability p iff $\theta(a_1, \ldots, a_k) \overset{p}{\hookrightarrow} a$ with probability p; and,
(2) for each probabilistic poly-time function $f: \mathbb{N}^k \to \mathbb{N}$, there exists a k-ary basic term θ such that M_θ computes f.

The first condition guarantees that all basic terms are computable in polynomial time, while the second condition guarantees that any probabilistic poly-time function of type $\mathbb{N}^k \to \mathbb{N}$ can be expressed by some basic term. One example of such class of terms is the term calculus OSLR studied in [50] (based in turn on [9,37]). As a consequence of these two conditions, we will, henceforth, freely move between terms and functions. Whenever we need to explicitly specify a basic term, we will use a notation styled on λ-calculus. For example, $\lambda x.\lambda y.(x + y)$ denotes the basic term that computes the sum of its two inputs.

Definition 1 (*Terms*). Letting θ range over basic terms and x range over $\textbf{Var} \cup \{\eta\}$, the class **Term** of *terms* is given by the grammar:

$$\mathrm{T} ::= x|\eta|\text{rand}|(\theta)\ \mathrm{T}_1, \ldots, \mathrm{T}_k \quad \text{where } \theta \text{ is a basic term of } k \text{ arguments.}$$

We emphasise that the distinguished variable η is treated just like any ordinary free variable of the term. We can define the *term reduction* of terms with no variables inductively:

(1) The term rand reduces to either 0 or 1 with uniform probability. We use rand as a source of truly random bits.
(2) The term $(\theta)\ \mathrm{T}_1, \ldots, \mathrm{T}_k$ is reduced by first reducing $\mathrm{T}_1, \ldots, \mathrm{T}_k$ yielding the values $a_1, \ldots, a_k$ and then reducing $\theta(a_1, \ldots, a_k)$.

Given a term T with variables $x_1, \ldots, x_k$ (where there might exist i such that x_i is η) we write $[a_1, \ldots, a_k/x_1, \ldots, x_k]$T for the term obtained by substituting a_i $(1 \leqslant i \leqslant k)$ for each free occurrence of x_i in T. We will write $\mathrm{T}(a_1, \ldots, a_k) \xrightarrow{p}_{\text{term}} a$ just when $[a_1, \ldots, a_k/x_1, \ldots, x_k]$T reduces to a with probability p.

A term T is an *atom* just when it has no free variables and reduces to itself with probability 1. It is easy to see that the terms representing the natural numbers are all atoms. Given a term T all of whose variables are among $x_1, \ldots, x_k$, it is easy to construct a probabilistic polynomial-time Turing machine M_{T} with k inputs such that $M_{\mathrm{T}}(a_1, \ldots, a_k) = a$ with probability p just when $\mathrm{T}(a_1, \ldots, a_k) \xrightarrow{p}_{\text{term}} a$.

Since terms are essentially substitution instances of basic terms, it is easy to see that all terms always terminate in polynomial time (since the basic terms are precisely the set of probabilistic polynomial time functions).

1.2. Syntax

Definition 2 (*Expressions*). Letting T range over **Term**, x range over **Var**, c range over **Channel**, c range over **Bindable**, and $q(\cdot)$ range over **Poly**, the class **Expr** of *expressions* of the probabilistic poly-time calculus (PPC) is given by the grammar:

$$\begin{array}{lr}
\mathcal{P} ::= \oslash & \text{(empty)} \\
\quad\; \nu(c).(\mathcal{P}) & \text{(channel binding)} \\
\quad\; in(c, x).(\mathcal{P}) & \text{(input)} \\
\quad\; out(c, \mathrm{T}).(\mathcal{P}) & \text{(output)} \\
\quad\; [\mathrm{T}].(\mathcal{P}) & \text{(match)} \\
\quad\; (\mathcal{P} \mid \mathcal{P}) & \text{(parallel composition)} \\
\quad\; !_{q(\eta)}.(\mathcal{P}) & \text{(bounded replication).}
\end{array}$$

Intuitively, $\oslash$ is the *empty process* that does nothing. The *channel binding* $\nu(c).(\mathcal{P})$ binds the channel name c in $\mathcal{P}$. Only channels in **Bindable** can be bound to a ν operator. Essentially, we have two kinds of channel names: channels in **Unbindable** that cannot be bound which we call *unbindable channels*, and channels in **Bindable** that can be bound which we call *bindable channels*. We will use bindable channels to create various secure primitives (such as secure authenticated channels). To this end, we will stipulate that messages on bindable channels occur with higher priority than messages on unbindable channels, i.e., bindable channels will be used to construct internal channels of protocols. Additionally, when a bindable channel is bound to a ν, no entity outside the scope of the ν will be able to interact with the channel, i.e., a bound channel produces no observable behaviour and cannot be read from or written to by an adversary. We emphasise that all free channels (which include bindable channels that have not been bound) produce observable behaviour. For simplicity we will assume that all bound channel names are distinct from each other and from unbound channel names. The *input expression* $in(c, x).(\mathcal{P})$ binds all free occurrences of the variable x in $\mathcal{P}$. Intuitively, the input expression receives an input value a on the channel c and then substitutes a for all free occurrences of the variable x in $\mathcal{P}$. The *output expression* $out(c, \mathrm{T}).(\mathcal{P})$ first reduces T to some atom a and then transmits that value on the channel c before proceeding with $\mathcal{P}$. We note that both the input operator and the output operator act as guards on the expression affixed as a suffix since, for example, $\mathcal{P}$ in the expression $out(c, \mathrm{T}).(\mathcal{P})$ cannot be evaluated until the output $out(c, \mathrm{T})$ is performed. In PPC all computation is performed using terms; the process expressions simply move values around between the various "islands of computation" that the terms embody. The picture here is meant to conform closely to the picture of a network (modelled by PPC) connecting various computers (modelled by terms). The *match expression* $[\mathrm{T}].(\mathcal{P})$ guards the expression $\mathcal{P}$ by only allowing the evaluation of $\mathcal{P}$ if the guarding term T reduces to the atom 1. Otherwise, the match evaluates to $\oslash$. The expression $\nu(c).(out(c, \mathrm{T}_1 \overset{?}{=} \mathrm{T}_2).(\oslash) \mid in(c, x).([\mathrm{x} \overset{?}{=} 1].(\mathcal{P}) \mid [\mathrm{x} \overset{?}{=} 0].(\mathcal{Q})))$ can be used to implement an "if–then–else–if" construct. The *parallel composition* $(\mathcal{P}_1 \mid \mathcal{P}_2)$ allows the expression on the left to evaluate independently of the expression on the right. Alternatively, the two expressions can communicate with each other by one sending a message (via an output) to the other (who receives it via an input on the same channel). We assume that $\mid$ associates to the left. Finally, the *bounded replication* $!_{q(\eta)}.(\mathcal{P})$ is simply the $q(\eta)$-fold parallel composition of $\mathcal{P}$. The parameter η is supposed to signify a *security parameter*, a basic notion in cryptography that allows one to characterise the 'strength' of a cryptosystem (see [31,44]). We assume that the value for the security parameter is given in unary—this is standard practice in the community and has the nice effect of ensuring that $|\eta| = \eta$.

Example 3. Here are some sample expressions.

(1) We assume that we have terms rsaParams that generates the public values of an RSA cryptosystem parameterised by η and randMsg that generates a random message of length determined by η. The expression

$$(in(c_1, x).(in(c_2, y).(out(d, \mathrm{rsa}(\mathrm{x}, \mathrm{y})).(\oslash))) \\ \mid (out(c_1, \mathrm{rsaParams}).(\oslash) \mid out(c_2, \mathrm{randMsg}).(\oslash)))$$

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

(2) The expression

$$!_{2\eta}.(out(c, \mathrm{rand}).(\oslash))$$

can potentially transmit 2η random bits given a suitable number of inputs on the channel c to receive the bits.

(3) Writing $\|$ for concatenation, we can guess η-length keys using

$$(out(c, \mathrm{rand}).(\oslash) \mid !_{\eta}.(in(c, x).(out(c, \mathrm{x}\|\mathrm{rand}).(\oslash))))$$

Definition 4 (*Contexts*). Letting T range over **Term**, x range over **Var**, c range over **Channel**, c range over **Bindable**, and $q(\cdot)$ range over **Poly**, the class **CExpr** of *context expressions* of PPC is given by the grammar:

$$\begin{aligned} \mathcal{C}[\,\cdot\,] ::= {} & [\,\cdot\,] \qquad\qquad\qquad \text{(hole)} \\ & \nu(c).(\mathcal{C}[\,\cdot\,]) \\ & in(c, x).(\mathcal{C}[\,\cdot\,]) \\ & out(c, \mathrm{T}).(\mathcal{C}[\,\cdot\,]) \\ & [\mathrm{T}].(\mathcal{C}[\,\cdot\,]) \\ & (\mathcal{P} \mid \mathcal{C}[\,\cdot\,]) \\ & (\mathcal{C}[\,\cdot\,] \mid \mathcal{P}) \\ & !_{q(\eta)}.(\mathcal{C}[\,\cdot\,]). \end{aligned}$$

We define the *substitution of the expression $\mathcal{P}$ into the context expression* $\mathcal{C}[\,\cdot\,]$, written as $\mathcal{C}[\mathcal{P}]$, inductively in an entirely straightforward manner: for the basis we stipulate that $[\mathcal{P}] \equiv \mathcal{P}$. Similarly, we can define the substitution of context expressions into context expressions. Given two expressions $\mathcal{P}$ and $\mathcal{Q}$ we will say that $\mathcal{Q}$ is a *subexpression* of $\mathcal{P}$ just when there exists a context expression $\mathcal{C}[\,\cdot\,]$ such that $\mathcal{P} \equiv \mathcal{C}[\mathcal{Q}]$. Similarly, we can define context subexpressions.

Given an expression $\mathcal{Q}$, when we find a $\mathcal{C}[\,\cdot\,]$ for some $\mathcal{P}$ such that $\mathcal{C}[\mathcal{P}] \equiv \mathcal{Q}$, we note that, in addition to witnessing that $\mathcal{P}$ is a subexpression of $\mathcal{Q}$, $\mathcal{C}[\,\cdot\,]$ also *locates $\mathcal{P}$ in $\mathcal{Q}$*. That is to say, a context can be used to uniquely identify the location of a subexpression in another expression. In practice, it will be useful to simultaneously identify the location of several subexpressions. For example, if we wish that a particular input receive a message from a particular output, it will be useful to write a context with two holes that pick out the designated input and output. By allowing derivations of the form $(\mathcal{C}[\,\cdot\,] \mid \mathcal{C}[\,\cdot\,])$, we can construct such *multi-holed context expressions*. We assume that the holes in a multi-holed context expression are uniquely numbered so as to render substitution unambiguous. Given a k-holed context expression $\mathcal{C}[\,\cdot_1, \ldots, \cdot_k]$ we denote the expression obtained by plugging in the expressions $\mathcal{P}_1, \ldots, \mathcal{P}_k$ into the holes $[\,\cdot\,]_1, \ldots, [\,\cdot\,]_k$, respectively, by $\mathcal{C}[\mathcal{P}_1, \ldots, \mathcal{P}_k]$. Naturally, substituting $j < k$ expressions into a k-holed context expression yields a $(k-j)$-holed context expressions. Given $\mathcal{C}[\,\cdot_1, \ldots, \cdot_k]$ and $\mathcal{P}_1, \ldots, \mathcal{P}_k$ and $\mathcal{Q}$ such that $\mathcal{C}[\mathcal{P}_1, \ldots, \mathcal{P}_k] \equiv \mathcal{Q}$, we will say that $\mathcal{C}[\,\cdot_1, \ldots, \cdot_k]$ *locates* $\mathcal{P}_1, \ldots, \mathcal{P}_k$ *in* $\mathcal{Q}$. In a similar manner (and for a similar reason) we also define *multi-holed contexts*. We denote the set of all multi-holed context expressions (resp. multi-holed contexts) by **MultiCExpr** (resp. **MultiCon**). Clearly, **CExpr** $\subset$ **MultiCExpr** and **Con** $\subset$ **MultiCon**. We will drop the numbering of holes in multi-holed context expressions and contexts whenever we can safely do so.

We define the *free variables* of an expression $\mathcal{P}$, denoted by $FreeVar(\mathcal{P})$, inductively:

$$\begin{aligned}
FreeVar(\oslash) &\stackrel{\text{def}}{=} \emptyset,\\
FreeVar(\nu(c).(\mathcal{P})) &\stackrel{\text{def}}{=} FreeVar(\mathcal{P}),\\
FreeVar(in(c,x).(\mathcal{P})) &\stackrel{\text{def}}{=} FreeVar(\mathcal{P}) - \{x\},\\
FreeVar(out(c,\mathrm{T}).(\mathcal{P})) &\stackrel{\text{def}}{=} FreeVar(\mathrm{T}) \cup FreeVar(\mathcal{P}),\\
FreeVar([\mathrm{T}].(\mathcal{P})) &\stackrel{\text{def}}{=} FreeVar(\mathrm{T}) \cup FreeVar(\mathcal{P}),\\
FreeVar((\mathcal{P}_1 \mid \mathcal{P}_2)) &\stackrel{\text{def}}{=} FreeVar(\mathcal{P}_1) \cup FreeVar(\mathcal{P}_2),\\
FreeVar(!_{q(\eta)}.(\mathcal{P})) &\stackrel{\text{def}}{=} FreeVar(\mathcal{P}).
\end{aligned}$$

An expression is *variable-closed* when it has no free variables, and *variable-open* otherwise. We let $Channel(\mathcal{P})$ be the set of *channels names* appearing in $\mathcal{P}$. The set of *public channels of* $\mathcal{P}$, denoted by $PubChan(\mathcal{P})$, is $\{c \in Channel(\mathcal{P}) \mid c \in \textbf{Unbindable}\}$. The set of *private channels of* $\mathcal{P}$, denoted by $PrivChan(\mathcal{P})$, is $\{c \in Channel(\mathcal{P}) \mid c \in \textbf{Bindable}\}$. The set of *bound channels of* $\mathcal{P}$, denoted by $BoundChan(\mathcal{P})$, is the largest subset of the channels of $\mathcal{P}$ whose members are all bound by a ν in $\mathcal{P}$. The *free channels of* $\mathcal{P}$, denoted by $FreeChan(\mathcal{P})$, is the given by $Channel(\mathcal{P}) \setminus BoundChan(\mathcal{P})$. Clearly, $PubChan(\mathcal{P}) \subset FreeChan(\mathcal{P})$ and $BoundChan(\mathcal{P}) \subset PrivChan(\mathcal{P})$. Similarly, we can define the sets of channels, free channels, bound channels, public channels and private channels of a context expression.

The reader will recall that each channel name has a polynomial in one variable associated with it by the map *width*. The argument to this *bandwidth* polynomial is given by the choice of value for η (which, we remind the reader, is equal to the size of η since it is written in unary). Intuitively, the bandwidth of a channel controls how much data can be sent along a channel in one message. Limiting the bandwidth of a channel to a polynomial in η will be crucial in proving that variable-closed expressions can be evaluated in time polynomial in the size of η (see Theorem 28). If channels could transmit messages of unbounded size, then we could construct exponentially long messages using a technique like repeated squaring. If these messages were then used as inputs to polynomial-time functions, the result would be an expression that takes exponential time to evaluate.

Our goal will be to develop an equational reasoning system for processes. In order to obtain an easy mechanism of generic compositionality, we will require that process equivalence be a congruence. The intuition here is that we can represent a security protocol as an expression, and an adversarial environment in which that protocol executes as a context. If we can show that an expression representing a protocol is congruent to some expression representing an ideal specification, we will have shown that no adversary can distinguish (i.e., attack) the protocol with non-negligible advantage. As a result, process equivalence will give us both generic compositionality (from its congruence properties) and correctness of security protocols (by generating equivalences with ideal specifications).

Henceforth, when writing (context) expressions we will drop any parentheses that are not required for giving an unambiguous parse to the (context) expression. We will also drop the $\oslash$ expression affixed as suffixes to input and output expressions.

Processes and contexts. We now discuss the distinguished variable η that appears in the definitions of expressions, terms, and in the bandwidths associated with channel names. This variable is supposed to represent the security parameter of a cryptographic scheme or security protocol. Given an expression $\mathcal{P}$ we can substitute a number i for the security parameter to obtain the *process* $[i/\eta]\mathcal{P}$. Thus every expression $\mathcal{P}$ can be viewed as defining the uniform family of processes $\{[i/\eta]\mathcal{P} \mid i \in \mathbb{N}\}$. We can similarly define context processes or *contexts* from context expressions. All the definitions given in the previous section for expressions and context expressions can be easily extended to processes and contexts. We will write $width([i/\eta]\mathcal{P}, c)$ to denote the value $2^{width(c)(i)}$. This is one more than maximum value that can be transmitted on the channel c in the process $[i/\eta]\mathcal{P}$—any message transmitted on the channel c in the process $[i/\eta]\mathcal{P}$ is reduced modulo $width([i/\eta]\mathcal{P}, c)$.

Wherever we can safely do so, we will drop the explicit indication of the value chosen for the security parameter and simply write P for a process or $C[\,\cdot\,]$ for a context. We note that processes do not contain replications since, by fixing a value for the security parameter, we can replace replications by parallel compositions. We define **Proc** and **Con** to be the set of processes and contexts, respectively. Since **Proc** (resp. **Con**) consists of expressions (resp. context expressions) in which the security parameter does not appear, it follows that **Proc** $\subset$ **Expr** and **Con** $\subset$ **CExpr**.

1.3. An abstract operational semantics for PPC

We start by giving the operational semantics for variable-closed processes; we will later outline an extension to variable-open processes and expressions. Intuitively, we envision the evaluation of variable-closed processes as a series of alternating reduction and communication steps. During a reduction step we evaluate as many terms and match expressions as we can. This step is supposed to embody all the (internal) computation a process can perform without requiring a communication step. Hence, we stipulate that reduction steps occur with higher priority than communication steps. A communication step consists of an input receiving a message from an output on the same channel. The communication step is chosen according to some probabilistic schedule from the set of all possible communications available to the process. This procedure of alternating reduction and communication steps continues until no further communication steps are possible.

There are several subtleties in this intuitive notion of process evaluation. The most important is the decision to make use of probabilistic scheduling. Our goal in this project is to develop a framework for reasoning about the computational model of security. Hence, we do not make Dolev–Yao-style simplifying assumptions. On the other hand, if we retain the traditional nondeterministic scheduling seen in [1–3,13,61–63], the adversary has exponential computing power. Consider the process

$$\begin{aligned} &in(d, y).in(c_1, x_1).\cdots.in(c_k, x_k).out(c, \mathrm{Decrypt}_{\mathrm{x_1\|\cdots\|x_k}}(\mathrm{y})) \mid \\ &\qquad \mid out(c_1, 0) \mid \cdots \mid out(c_k, 0) \mid out(c_1, 1) \mid \cdots \mid out(c_k, 1) \end{aligned}$$

With nondeterministic scheduling, the adversary can guess a k-length key by concatenating k bits. Since we do not make Dolev–Yao assumptions, the adversary is fully justified in manipulating bit-level representations of keys. In practice though, the adversary can only successfully reconstruct the key with probability at most 2^{-k} which, for sufficiently large k, is considered negligible. Hence, abandoning the Dolev–Yao model drives us to probabilistic scheduling.

However, we cannot a priori fix a probabilistic schedule when we give the operational semantics for PPC. Since we will wish to define process equivalence with respect to all adversaries, and adversaries are modelled as having control over the network, we will need to quantify over all probabilistic schedules. Consequently, we will define for each process a labelled transition system where the probabilities annotating edges do not take into account a particular schedule: we will have a *reactive* labelled transition system (see [40,65]) in that the transition system for a process represents the behaviour of the process in the presence of an external entity that selects the communication step to perform at every step of evaluation. In particular, our transition rules will associate with each process a set of distributions indexed by actions. Thus, while the labelled transition system will yield probabilistic behaviour given an particular action label, we can only guarantee that the process displays probabilistic behaviour over all choices of actions once we fix a schedule. This will be done by applying a *scheduler* to the labelled transition system (Section 3.1). Strictly speaking the values that will be associated with transitions are not probabilities in the sense that the overall behaviour of a process is not guaranteed to be probabilistic. However, for convenience, we will refer to these numbers as probabilities because, given a scheduler, they yield probabilistic behaviour.

Another issue stems from what counts as a communication step. Intuitively, a communication step is just an input–output pair on the same channel: the output transmits the value used by the input. However, we would like to develop a syntactic treatment of compositionality. Consequently, we need some way of modelling the evaluation of a process in any context. In order to do this, we will make use of partial steps that are single inputs and outputs representing a communication with an arbitrary context in which the process is being evaluated. The result is that we develop an *abstract* operational semantics that gives the behaviour of a process in an arbitrary context.

Finally, there is the question of what to do with communications on bindable channels versus unbindable channels. As already mentioned, bindable channels will be used to build primitives like authenticated secure channels. Their primary goal is to model channels which cannot be seen by the adversary. In order to do this, it is necessary to make communications on bindable channels occur with higher priority than communications on unbindable channels. Additionally, an important element in constructing these secure primitives is that the adversary cannot interact with the channel. For this reason, we require the notion of syntactic scope provided by the ν operator. Any actual communication consisting of an input and output on a bound channel should produce no observable. However, a message on an free bindable channel is observable since the adversary can interact with it. Hence, all communications steps on bindable channels (including partial ones) generate observable behaviour in exactly the same way as communication steps on

$$\emptyset \xrightarrow{1}_{\text{redxn}} \emptyset \tag{RZ}$$

$$in(c, x).P \xrightarrow{1}_{\text{redxn}} in(c, x).P \tag{RI}$$

$$\frac{p = \sum_{\{b \in \mathbb{N} \mid a = b \bmod width(P,c)\}} \text{Prob}\left[\text{T} \rightarrow_{\text{term}} b\right],\ P \xrightarrow{q}_{\text{redxn}} P'}{out(c, \text{T}).P \xrightarrow{pq}_{\text{redxn}} out(c, \text{a}).P'} \tag{RO}$$

$$\frac{P \xrightarrow{p}_{\text{redxn}} P',\ \text{T} \xrightarrow{q}_{\text{term}} 1}{[\text{T}].P \xrightarrow{pq}_{\text{redxn}} P'} \tag{RMP}$$

$$\frac{\text{T} \xrightarrow{q}_{\text{term}} 1}{[\text{T}].P \xrightarrow{1-q}_{\text{redxn}} \emptyset} \tag{RMF}$$

$$\frac{P \xrightarrow{p}_{\text{redxn}} P',\ Q \xrightarrow{q}_{\text{redxn}} Q'}{P \mid Q \xrightarrow{pq}_{\text{redxn}} P' \mid Q'} \tag{RPC}$$

$$\frac{P \xrightarrow{p}_{\text{redxn}} P'}{\nu(c).P \xrightarrow{p}_{\text{redxn}} \nu(c).P'} \tag{RCB}$$

Fig. 1. The reduction rules for variable-closed PPC processes.

unbindable channels. Once a bindable channel is bound, within its scope no partial steps can occur (since the arbitrary context in which we evaluate the process cannot communicate with the bound channel) and all normal communication steps produce no observable behaviour. We remind the reader that we have three different levels of priorities for actions:

(1) reduction actions which occur with the highest priority,
(2) actions on bindable channels which occur with medium priority, and,
(3) actions on unbindable channels which occur with low priority.

The operational semantics for PPC in Fig. 2 require definitions of the notions of process reduction (Section 1.3.1), an action (Section 1.3.2), and the normalisation function, *Norm* (Section 1.3.3), used in computing the probabilities of transitions under parallel composition. We proceed by defining these notions in turn before we discuss the operational semantics (Section 1.3.4). A treatment of probabilistic scheduling can be deferred until later (Section 3.1) since, as mentioned, we do not require a particular probabilistic schedule in providing the operational semantics.

1.3.1. Process reduction

We define *process reduction* via the inference rules of Fig. 1. Intuitively, reduction is meant to capture the computation a process does between communication steps. Since a free variable in a process can be viewed as a resource that needs to be delivered to the process via a communication step, it stands to reason that a variable-open process cannot do any computation (since it is waiting on a communication to deliver the missing resource). So reduction is defined only over variable-closed processes.

Given a variable-closed process, we evaluate all terms that have no free variables (in keeping with the just-given intuition). Since the only binding operator on free variables are inputs, we reduce all terms that do not appear in the scope of an input operator. We say that an output operator or a match operator is *exposed* if it does not appear in the scope of an input operator. Similarly, we say that a non-atomic term appearing in an exposed output[3] or any term appearing in an exposed match is exposed as well. We note that since we are considering only variable-closed processes, all exposed terms have no free variables. To reduce a process, we need to evaluate all the exposed terms. We will say that a variable-closed process is *blocked* just when it has no exposed terms, and *unblocked* otherwise.[4] We will say that Q is a *reduct* of P just when $\text{Prob}\left[P \rightarrow_{\text{redxn}} Q\right] > 0$. Finally, for a set of processes $\mathbf{R}$:

$$\text{Prob}\left[P \rightarrow_{\text{redxn}} \mathbf{R}\right] \overset{\text{def}}{=} \sum_{R \in \mathbf{R}} \text{Prob}\left[P \rightarrow_{\text{redxn}} R\right].$$

Lemma 5. $\text{Prob}\left[P \rightarrow_{\text{redxn}} \mathbf{Proc}\right] \leqslant 1$.

Proof. A trivial induction on the structure of processes. $\square$

As an immediate corollary we see that for any set of processes $\mathbf{R}$, we have $\text{Prob}\,[P] \rightarrow_{\text{redxn}} \mathbf{R} \leqslant 1$. We conclude this section with some remarks on some features of the reduction rules. Intuitively, an output $out(c, \mathrm{T}).P$ represents the sending of the message a computed by the term T. However, the bandwidth on channels means that the value transmitted by the output might be truncated if T evaluates to a large value. We handle this truncation mechanism by ensuring that our reduction rules reduce an output term to a suitably truncated value. Hence, the probability of the output reduction must be taken over all elements in the codomain of the term that, when truncated, result in the same value. We remark that our reduction rules imply that if $P \overset{1}{\rightarrow}_{\text{redxn}} \oslash$ then $[\mathrm{T}].P \overset{1}{\rightarrow}_{\text{redxn}} \oslash$. We remind the reader that the reduction rules of Fig. 1 make no mention of replications since processes do not contain any replications.

Lemma 6. $P \overset{1}{\rightarrow}_{\text{redxn}} P$ *iff* P *has no exposed terms i.e.*, P *is blocked.*

Proof. ($\Longrightarrow$) An examination of the inference rules of Fig. 1 shows that the only rules that can directly cause a syntactic change to a process are (RO), (RMP), and (RMF). Assuming $P \overset{1}{\rightarrow}_{\text{redxn}} P$ implies that the match cases do not occur and the term in the output case reduce to an atom with probability 1 (i.e., the term is an atom). Hence, P must contain no exposed terms.

($\Longleftarrow$) This case is obvious since (1) a lack of exposed terms implies that the match rules are never invoked, and (2) the terms in exposed outputs are atomic. $\square$

1.3.2. Actions

An action is an abstraction of a communication step. There will be four kinds of actions in our calculus: (1) actual actions, which are actions consisting of an input and an output and are supposed to represent the intuitive notion of a communication step; (2) input actions, each of which is just an input and represents communications where a message is received from an arbitrary evaluation context; (3) output actions, each of which is just an output and represents messages sent to an arbitrary evaluation context; and, (4) silent actions which are actions that the adversary cannot see or schedule. We define three sets:

$$\begin{aligned} \Lambda_{in} &\overset{\text{def}}{=} \{\langle c, a\rangle \mid c \in \mathbf{Channel}, a \in \mathbb{N}\}, \\ \Lambda_{out} &\overset{\text{def}}{=} \{\langle \bar{c}, a\rangle \mid c \in \mathbf{Channel}, a \in \mathbb{N}\}, \\ \Lambda_{silent} &\overset{\text{def}}{=} \{\langle \tau_c, a\rangle \mid c \in \mathbf{Bindable}\} \cup \{\varepsilon\}. \end{aligned}$$

We define a partial function $\cdot : \Lambda_{in} \cup \Lambda_{out} \times \Lambda_{in} \cup \Lambda_{out} \rightarrow \Lambda_{in} \times \Lambda_{out}$ such that $\forall c \in \mathbf{Channel}.\forall \alpha \in \mathbf{Act}\colon \langle c, a\rangle \cdot \langle \bar{c}, a\rangle = \langle \bar{c}, a\rangle \cdot \langle c, a\rangle$ and $\cdot$ is injective everywhere else. We can then define a fourth set:

$$\Lambda_{actual} \overset{\text{def}}{=} \{\langle c, a\rangle \cdot \langle \bar{c}, a\rangle \mid c \in \mathbf{Channel}, a \in \mathbb{N}\}.$$

[3] Since atomic terms are constant expressions, they do not represent computation that needs to be done. Hence, we will not refer to an atomic term in an exposed output as exposed.

[4] Our choice of the terms ‘blocked’ and ‘unblocked’ is guided by the intuition that a blocked process cannot perform any local computation and, hence, must wait for a communication step to occur.

We will call Λ_{in} the set of *input actions*, Λ_{out} the *output actions*, Λ_{actual} the *actual actions*, Λ_{silent} the *silent actions*, and ε the *reduction action*. The symbol ε represents a reduction step i.e., the application of the rules of Fig. 1 to a process. Since ε does not occur on any channel, and so cannot produce any observable behaviour, we treat it as a kind of silent action.

The set of *actions* of PPC, denoted by **Act**, is the set $\Lambda_{in} \cup \Lambda_{out} \cup \Lambda_{actual} \cup \Lambda_{silent}$. For any action α, we define *Channel*(α) as either the channel on which α takes place in the case in the case that $\alpha \neq \varepsilon$, or ε in the case that $\alpha = \varepsilon$. The channels in the set **PubAct**, defined as $\{\alpha \mid \alpha \in \mathbf{Act} \setminus \Lambda_{silent}\}$, are the *public actions* of PPC. The set $\Lambda_{in} \cup \Lambda_{out}$ is the set of *partial actions* of PPC. An actual action $\alpha \cdot \beta$ is the *simultaneous occurrence* of the actions α and β. Ignoring ε, a silent action is just some actual action that occurs on some bound channel.

It will be convenient to define the 'opposite' of an action, i.e., define an action β which generates the complementary observable behaviour of an action α. Given $\alpha, \beta \in \Lambda_{in} \cup \Lambda_{out}$, we will say that β is the *coaction* of α, written $\beta = \bar{\alpha}$, exactly when either

(1) $\alpha = \langle c, a\rangle$ and $\beta = \langle \bar{c}, a\rangle$ or
(2) $\alpha = \langle \bar{c}, a\rangle$ and $\beta = \langle c, a\rangle$.

In the rest of this paper, we use α to range over all actions, and $\bar{\alpha}$ to range over all coactions. We will also use τ to range over silent actions. It is easy to see that Λ_{actual} is just the set of all actions of the form $\alpha \cdot \bar{\alpha}$.

1.3.3. The normalisation function

We inductively define a *normalisation function*, *Norm*: $\mathbf{Proc} \times \mathbf{Act} \rightarrow \mathbb{Z}$, that computes the number of ways an unblocked process can take a particular action:

(1) $\forall \alpha \in \mathbf{Act}$: $Norm(\oslash, \alpha) = 0$. The zero process cannot perform any actions.
(2) $\forall a \in [0..width(P, c) - 1]$: $Norm(in(c, x).P, \langle c, a\rangle) = 1$. Recall that messages are truncated by the channel bandwidth.
(3) $Norm(out(c, \mathrm{a}).P, \langle \bar{c}, a \bmod width(P, c)\rangle) = 1$. Recall that messages are truncated by the channel bandwidth.
(4) $\forall \alpha \in \mathbf{Act}$: $(c \neq Channel(\alpha)) \Longrightarrow Norm(v(c).P, \alpha) = Norm(P, \alpha)$. The private channel operator does not affect the number of ways in which actions that do not occur on the channel c can be taken.
(5) $Norm(v(c).P, \langle \tau_c, a\rangle) = Norm(P, \langle c, a\rangle \cdot \langle \bar{c}, a\rangle)$. The expression $v(c).P$ can take a silent action on the channel c in the same number of ways that P can take actual actions on the channel c.
(6) $\forall \alpha \in \mathbf{Act} - \Lambda_{actual}$: $Norm(P \mid Q, \alpha) = Norm(P, \alpha) + Norm(Q, \alpha)$. We compute the number of ways in which partial or silent actions α can be taken by $P \mid Q$ by adding the number of ways α can be taken by P and Q individually. We note that in the case of silent actions, either $Norm(P, \alpha)$ or $Norm(Q, \alpha)$ will be equal to zero since we assume that all private channels are renamed apart.
(7) $Norm(P \mid Q, \alpha \cdot \bar{\alpha}) = Norm(P \mid Q, \alpha) \cdot Norm(P \mid Q, \bar{\alpha})$. To compute the number of ways that $P \mid Q$ can take an actual action $\alpha \cdot \bar{\alpha}$, we multiply the total number of ways that P and Q can take α and $\bar{\alpha}$, respectively.

For all other cases, the normalisation function evaluates to zero. The normalisation factor computed by this function will be used to normalise the probability of an action happening when combining processes via the parallel composition operator. This ensures that the total probability of an process being able to take a particular action never exceeds 1.

1.3.4. The operational semantics for variable-closed PPC processes

In Fig. 2 we give an operational semantics for variable-closed PPC processes. These operational semantics will be used to construct a labelled transition system for each process (Section 1.3.5). We extend this semantics to variable-open PPC processes in the obvious way by defining a set of labelled transitions systems for each variable-open PPC process (see Section 1.3.5).

We will write

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

iff it can be derived from the rules of Fig. 2. We refer to $P \xrightarrow{\alpha[p]} P'$ as an α-*transition* or just *transition* and its intuitive meaning is that if P takes the action labelled α, then with probability p it will become P'. We emphasise that this probability p is over all α-labelled transitions that P can take, not over all transitions (regardless of label) that P can

$$\forall\, a \in [0..width(P,c)-1]\colon\ in(c,x).P \xrightarrow{\langle c,a\rangle[1]} [a/x]P \tag{I}$$

$$out(c,\mathrm{a}).P \xrightarrow{\langle \bar{c}, a \bmod width(P,c)\rangle[1]} P \tag{O}$$

$$\frac{P \text{ unblocked},\ P \overset{p}{\to}_{\text{redxn}} P'}{P \xrightarrow{\varepsilon[p]} P'} \tag{R}$$

$$\frac{P, Q \text{ blocked},\ P \xrightarrow{\alpha[p]} P',\quad Channel(\alpha) \in \mathbf{Bindable}\ \vee\ Q \text{ has no actions on bindable channels}}{P \mid Q \xrightarrow{\alpha\left[p\cdot\frac{Norm(P,\alpha)}{Norm(P\mid Q,\alpha)}\right]} P' \mid Q} \tag{CL}$$

$$\frac{P, Q \text{ blocked},\ Q \xrightarrow{\alpha[q]} Q',\quad Channel(\alpha) \in \mathbf{Bindable}\ \vee\ P \text{ has no actions on bindable channels}}{P \mid Q \xrightarrow{\alpha\left[q\cdot\frac{Norm(Q,\alpha)}{Norm(P\mid Q,\alpha)}\right]} P \mid Q'} \tag{CR}$$

$$\frac{P, Q \text{ blocked},\ P \xrightarrow{\alpha[p]} P',\ Q \xrightarrow{\bar{\alpha}[q]} Q',\quad Channel(\alpha) \in \mathbf{Bindable}\ \vee\ P, Q \text{ have no actions on bindable channels}}{P \mid Q \xrightarrow{\alpha\cdot\bar{\alpha}\left[\left(p\cdot\frac{Norm(P,\alpha)}{Norm(P\mid Q,\alpha)}\right)\cdot\left(q\cdot\frac{Norm(Q,\bar{\alpha})}{Norm(P\mid Q,\bar{\alpha})}\right)\right]} P' \mid Q'} \tag{C}$$

$$\frac{P \text{ blocked},\ P \xrightarrow{\tau[p]} P'}{\nu(c).P \xrightarrow{\tau[p]} \nu(c).P'} \tag{P1}$$

$$\frac{P \text{ blocked},\ P \xrightarrow{\langle c,a\rangle\cdot\langle\bar{c},a\rangle[p]} P'}{\nu(c).P \xrightarrow{\langle\tau_c,a\rangle[p]} \nu(c).P'} \tag{P2}$$

$$\frac{P \text{ blocked},\ P \xrightarrow{\alpha[p]} P',\ \alpha \notin \Lambda_{silent},\ Channel(\alpha) \neq c}{\nu(c).P \xrightarrow{\alpha[p]} \nu(c).P'} \tag{P3}$$

Fig. 2. The operational semantics for variable-closed PPC processes.

take. Thus, the transition rules do not define probabilistic behaviour over all transitions, but only over all transitions of a given label. The choice of label is made by an external agent so as to capture the behaviour of the process under any (potentially nasty) scheduling of actions. We remind the reader that applying a particular schedule to the transition system will yield probabilistic behaviour (see Lemma 19); in light of this fact, we will continue to call these values annotating transitions as probabilities even though, strictly speaking, they are not. We will write Prob $\left[P \overset{\alpha}{\to} P'\right]$ to denote the probability of an α-transition taking P to P'. We will say that P *has an* α*-transition* if there exists P' such that Prob $\left[P \overset{\alpha}{\to} P'\right] > 0$. We will say that P' is an α*-child of* P if Prob $\left[P \overset{\alpha}{\to} P'\right] > 0$ and write $Children_\alpha(P)$ for the set of α-children of P. The *children of* P is the set $\bigcup_{\alpha\in\mathbf{Act}} Children_\alpha(P)$ and is denoted by $Children(P)$.

Let us discuss the inference rules. The full list is given in Fig. 2. The first group consists of a pair of axioms for inputs and outputs:

$$\forall\, a \in [0..width(P,c)-1]\colon\ in(c,x).P \xrightarrow{\langle c,a\rangle[1]} [a/x]P, \tag{I}$$

$$out(c,\mathrm{a}).P \xrightarrow{\langle \bar{c}, a \bmod width(P,c)\rangle[1]} P. \tag{O}$$

The axiom (I) states that an input on the channel c binding the variable x can receive any value that can ‘fit’ on the channel. This value is the substituted for all free occurrences of x in the scope of the input. Similarly, axiom (O) states that an output containing an atomic term on the channel c can transmit a suitably truncated version of that atom before proceeding with the process being guarded by the output.

The rule (R) deals with unblocked processes:

$$\frac{P \text{ unblocked}, P \xrightarrow{p}_{\text{redxn}} P'}{P \xrightarrow{\varepsilon[p]} P'}. \tag{R}$$

Our operational semantics will give the highest priority to reduction transitions since these represent local computation. Hence all other rules will require as a precondition that the processes being considered be blocked. The only way to ‘block’ an unblocked process is via this rule. We note that axioms (I) and (O) deal with blocked processes and so do not need the precondition just described. Rule (R) simply states that if P reduces to P' with probability p then there is an ε-transition from P to P' with probability p.

We now examine the group of rules dealing with $|$, parallel composition:

$$\frac{\begin{array}{c}P, Q \text{ blocked}, P \xrightarrow{\alpha[p]} P',\\ \mathit{Channel}(\alpha) \in \textbf{Bindable} \ \vee\ Q \text{ has no actions on bindable channels}\end{array}}{P \mid Q \xrightarrow{\alpha\left[p \cdot \frac{\mathit{Norm}(P,\alpha)}{\mathit{Norm}(P \mid Q,\alpha)}\right]} P' \mid Q}, \tag{CL}$$

$$\frac{\begin{array}{c}P, Q \text{ blocked}, Q \xrightarrow{\alpha[q]} Q',\\ \mathit{Channel}(\alpha) \in \textbf{Bindable} \ \vee\ P \text{ has no actions on bindable channels}\end{array}}{P \mid Q \xrightarrow{\alpha\left[q \cdot \frac{\mathit{Norm}(Q,\alpha)}{\mathit{Norm}(P \mid Q,\alpha)}\right]} P \mid Q'}, \tag{CR}$$

$$\frac{\begin{array}{c}P, Q \text{ blocked}, P \xrightarrow{\alpha[p]} P', Q \xrightarrow{\bar{\alpha}[q]} Q',\\ \mathit{Channel}(\alpha) \in \textbf{Bindable} \ \vee\ P, Q \text{ have no actions on bindable channels}\end{array}}{P \mid Q \xrightarrow{\alpha\cdot\bar{\alpha}\left[\left(p \cdot \frac{\mathit{Norm}(P,\alpha)}{\mathit{Norm}(P \mid Q,\alpha)}\right)\cdot\left(q \cdot \frac{\mathit{Norm}(Q,\bar{\alpha})}{\mathit{Norm}(P \mid Q,\bar{\alpha})}\right)\right]} P' \mid Q'}. \tag{C}$$

Rules (CL) and (CR) state that if a process P can take an action α, then so can the processes $P \mid Q$ and $Q \mid P$ for any process Q. However, the probabilities change since Q might be able to take the same action. To reflect this modified probability we simply re-normalise using the normalisation factor $\mathit{Norm}(P \mid Q, \alpha)$. Additionally, we must preserve the priorities on channels—$P \mid Q$ and $Q \mid P$ can take α only if Q has no actions of higher priority. Rule (C) allows two processes composed by $|$ to communicate with each other. Again, we need to re-normalise probabilities and preserve priorities on actions. We note that in (C), the actions cannot be silent because the scope given by ν cannot extend over both processes. Furthermore, in (CL) and (CR), if α is silent we effectively do not re-normalise since the silent actions in P occur on different channels from the private channels of Q whence $\mathit{Norm}(Q \mid P, \tau) = \mathit{Norm}(P \mid Q, \tau) = \mathit{Norm}(P, \tau)$.

Finally, we consider the group of rules dealing with ν, the channel-binding operator:

$$\frac{P \text{ blocked}, P \xrightarrow{\tau[p]} P'}{\nu(c).P \xrightarrow{\tau[p]} \nu(c).P'}, \tag{P1}$$

$$\frac{P \text{ blocked}, P \xrightarrow{\langle c,a\rangle\cdot\langle \bar{c},a\rangle[p]} P'}{\nu(c).P \xrightarrow{\langle \tau_c,a\rangle[p]} \nu(c).P'}, \tag{P2}$$

$$\frac{P \text{ blocked}, P \xrightarrow{\alpha[p]} P', \alpha \notin \Lambda_{silent}, \mathit{Channel}(\alpha) \neq c}{\nu(c).P \xrightarrow{\alpha[p]} \nu(c).P'}. \tag{P3}$$

Intuitively, a ν just defines the scope within which the bound channel is visible. Thus, it does not affect scheduling of channels. It simply eliminates partial actions on the bound channel (since they cannot communicate with the abstract context in which the process is being evaluated) and makes the actual actions on the bound channel invisible (since no entity outside the scope of the bound channel can see the bound channel let alone messages on it). Rule P1 states that a ν does not affect already extant silent actions. Rule P2 states that if P has a p-probability transition labelled by an actual action on the channel c to P', then $\nu(c).P$ has, essentially, the same transition with the same probability. Rule P3 states that ν does not affect public actions that do not occur on the channel c.

Lemma 7. *Let P be a variable-closed process and α an action. Then*:

(1) $\{Q \in \mathbf{Proc} \mid \mathrm{Prob}\,[P \xrightarrow{\alpha} Q] > 0\}$ *is finite and*
(2) $\sum_{Q \in \mathbf{Proc}} \mathrm{Prob}\,[P \xrightarrow{\alpha} Q] \leqslant 1$.

Proof. Both claims follow from entirely routine inductions on the structure of variable-closed processes. □

Lemma 8. *Let P be a blocked process. Then*

$$\mathrm{Prob}\left[P \xrightarrow{\alpha\cdot\bar{\alpha}} Q\right] = \frac{1}{\mathit{Norm}(P, \alpha\cdot\bar{\alpha})}.$$

Proof. Since P is blocked, every input and output that can possibly communicate must not be in the scope of a match, and every output that can possibly communicate with an input must be of the form $\mathit{out}(c, \mathrm{a}).Q$. Then, we continue by induction on the number of exposed (i.e., not in the scope of an input) $|$-operators in the process. Assuming that the lemma holds for up to k $|$ operators, we can show that

$$\mathrm{Prob}\left[P_1 \mid P_2 \xrightarrow{\langle c,a\rangle} Q_1 \mid Q_2\right] = \frac{1}{\mathit{Norm}(\langle c, a\rangle, P_1 \mid P_2)},$$

since the inductive hypothesis allows us to calculate that

$$\frac{\mathrm{Prob}\left[P_i \xrightarrow{\langle c,a\rangle} Q_i\right]\cdot \mathit{Norm}(P_i, \langle c, a\rangle)}{\mathit{Norm}(P_1 \mid P_2, \langle c, a\rangle)} = \frac{1}{\mathit{Norm}(P_1 \mid P_2, \langle c, a\rangle)}.$$

We can perform a similar calculation for all actions of the form $\langle \bar{c}, a\rangle$. It then is immediate that the lemma holds for actions of the form $\alpha\cdot\bar{\alpha}$. □

1.3.5. Labelled transition systems

Using the operational semantics given in Fig. 2 we define the labelled transition system for a variable-closed process P.

Definition 9. A *labelled transition system* (LTS) is a 3-tuple $\langle R, T, I\rangle$ where:

(1) R is a set of states each labelled by a variable-closed process,
(2) $T \subseteq R \times \mathbf{Act} \times [0, 1] \times R$ is a set of arrows such that $T(r, \alpha, p, r')$ iff $\hat{r} \xrightarrow{\alpha[p]} \hat{r}'$ where $\hat{r}$ denotes the process labelling the state r; and
(3) I is the initial state and is labelled by P.

An LTS is just the graph induced by the operational semantics for PPC. It is clear from the definition of an LTS that a process can have several different LTSs. For example, given the LTS $\langle R, T, I\rangle$ for P we can construct another LTS $\langle R', T, I\rangle$ for P by adding a state $u \notin R$ labelled by a process Q that is not reachable from the initial state.

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. Then we will denote the unique LTS for a variable-closed process P, obtained by discarding unreachable states and identifying states labelled with the same process, by $\mathbb{L}(P)$. For simplicity we will, henceforth, identify states and the processes labelling them. Letting **LTS** be the set of labelled transitions systems and **VarClosedProcs** the set of variable-closed processes, we can also define the mapping $\mathbb{L}$: **VarClosedProcs** $\rightarrow$ **LTS**. It is easy to see that $\mathbb{L}(P)$ is a directed acyclic graph.

We extend $\mathbb{L}$ to an interpretation $\hat{\mathbb{L}}$: $\mathbf{Proc} \to 2^{\mathbf{LTS}}$ of all processes by extending $\mathbb{L}$ in the following manner. Given a process P and a valuation ξ of the free variables of P, we denote by $\xi(P)$ the process obtained by substituting $\xi(x)$ for all free occurrences of x in P. With this notation in place, we define $\hat{\mathbb{L}}(P)$ as the set $\{\mathbb{L}(\xi(P)) \mid \xi$ is a valuation of $\mathit{FreeVar}(P)\}$. Thus $\hat{\mathbb{L}}$ is a function that maps processes to sets of LTSs (one for each valuation of the free variables of the process).

We can further extend $\mathbb{L}$ to an interpretation $\tilde{\mathbb{L}}$: $\mathbf{Expr} \to 2^{2^{\mathbf{LTS}}}$ of all expressions in the following fashion. Given an expression $\mathcal{P}$, we define $\tilde{\mathbb{L}}(\mathcal{P})$ as the set $\{\hat{\mathbb{L}}([i/\eta]\mathcal{P}) \mid i \in \mathbb{N}\}$. Analogous to $\hat{\mathbb{L}}$, the function $\tilde{\mathbb{L}}$ maps expressions to sets of sets of LTSs (one set of LTSs for each process in the process family defined by the expression).

Action paths. Let $\mathbb{L}(P)$ be the labelled transition system for a process P. An *action path from P to Q* is a path in $\mathbb{L}(P)$ from P to Q with each edge in the path having non-zero probability. Given $\alpha \in \mathbf{PubAct}$, an *$\alpha$-path from P to Q* is a path all of whose edges except the final one leading to Q are labelled with silent actions. The final edge is labelled with α. A length k path has k edges; a length k α-path has $k-1$ edges labelled by silent actions followed by a single edge labelled by α. The probability of a path π is computed by taking the product of the probabilities of each edge in π. Given a set of processes $\mathbf{R}$, we say that π is a path from P into $\mathbf{R}$ if π is a path from P to some process $R \in \mathbf{R}$. For any public action α, we write $\mathit{Paths}(P, \alpha, \mathbf{R})$ to denote the set of all α-paths from P into $\mathbf{R}$.

2. Probabilistic bisimulation

In this section we adapt weak bisimulation [49] to a probabilistic setting. Following the elegant and economical development of Van Glabbeek et al. [65], we will present our probabilistic bisimulation as an equivalence relation over **Proc**.

Definition 10. We define μ: $\mathbf{VarClosedProcs} \times \mathbf{PubAct} \times 2^{\mathbf{VarClosedProcs}} \to [0, 1]$, the *cumulative mass function* (cMF), by the equation

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

Lemma 11. $\forall P \in \mathbf{VarClosedProcs}. \forall \alpha \in \mathbf{PubAct}$: *$\mu(P, \alpha, \mathbf{VarClosedProcs})$ is finite.*

Proof. By an induction on the maximum length of α-paths. We give a short sketch of the proof. We will write $\mu^k(P, \alpha, \mathbf{R})$ to denote that the value $\mu(P, \alpha, \mathbf{R})$ is calculated over at most k-length paths. For the basis we consider paths with length at most 1, i.e., paths with no silent actions. Then we can directly apply Lemma 7. To justify the inductive hypothesis we note that the value $\mu(P, \alpha, \mathbf{VarClosedProcs})$, when restricted to at most k-length paths, can be calculated as

$$\sum_{Q \in \mathbf{VarClosedProcs}} \mathrm{Prob}\left[P \xrightarrow{\alpha} Q\right] + \sum_{\tau \in \Lambda_{silent}} \sum_{Q \in \mathbf{VarClosedProcs}} \mathrm{Prob}\left[P \xrightarrow{\tau} Q\right] \cdot \mu^{k-1}(Q, \alpha, \mathbf{VarClosedProcs}).$$

From the inductive hypothesis, we know that $\mu^{k-1}(P, \alpha, \mathbf{VarClosedProcs})$ is finite. From Lemma 7, we know that $\forall \tau \in \Lambda_{silent}: \sum_{Q \in \mathbf{VarClosedProcs}} \mathrm{Prob}\left[P \xrightarrow{\tau} Q\right] \leqslant 1$ and $\sum_{Q \in \mathbf{VarClosedProcs}} \mathrm{Prob}\left[P \xrightarrow{\alpha} Q\right] \leqslant 1$. Lemma 7 also states that the outdegree of any process is finite. Therefore,

$$\sum_{\tau \in \Lambda_{silent}} \sum_{Q \in \mathbf{VarClosedProcs}} \mathrm{Prob}\left[P \xrightarrow{\tau} Q\right] \cdot \mu^{k-1}(P, \alpha, \mathbf{VarClosedProcs})$$

is finite. □

As an immediate corollary we have that $\mu(P, \alpha, \mathbf{R})$ is finite for any public α and set of variable-closed processes $\mathbf{R}$. We cannot prove that the cMF for P is a collection of distributions indexed by α and $\mathbf{R}$ (i.e., that $\mu(P, \alpha, \mathbf{R}) \leqslant 1$) since Lemma 7 says that the transition probabilities of a variable-closed process form a well-defined probability distribution for *each* type of silent action. To show that a cMF is a collection of distributions, we would need that the transition probabilities of a variable-closed process form a well-defined distribution over all silent actions. We extend cMFs to all processes by defining the cMF of a variable-closed process as a set of cMFs, one per valuation of the free variables of the process. In particular, given two variable-open processes, P and Q, we say that $\mu(P, \alpha, \mathbf{R}) = \mu(Q, \alpha, \mathbf{R})$ if, under all valuations, the cMFs are pairwise identical.

Definition 12. An equivalence relation $B \subseteq \mathbf{Proc} \times \mathbf{Proc}$ is a *weak probabilistic bisimulation* or *bisimulation* just when $\langle P, Q\rangle \in B$ implies that:

(1) $\forall \mathbf{R} \in \mathbf{Proc}/_B . \forall \alpha \in \mathbf{PubAct}\colon \mu(P, \alpha, \mathbf{R}) = \mu(Q, \alpha, \mathbf{R})$,
(2) if P and Q are blocked, then $\forall \alpha \in \mathbf{Act}\colon \mathit{Norm}(P, \alpha) = \mathit{Norm}(Q, \alpha)$,
(3) if either P or Q is unblocked, then:

$$\forall \mathbf{R} \in \mathbf{Proc}/_B : \mathrm{Prob}\left[P \rightarrow_{\mathrm{redxn}} \mathbf{R}\right] = \mathrm{Prob}\left[Q \rightarrow_{\mathrm{redxn}} \mathbf{R}\right].$$

Two processes, P and Q, are *bisimulation-equivalent* (denoted by $P \sim Q$) if there exists a weak probabilistic bisimulation B such that $\langle P, Q\rangle \in B$, i.e.,

$$\sim \overset{\text{def}}{=} \bigcup \{B \mid B \text{ is a weak probabilistic bisimulation}\}.$$

As a short-hand, we will write $P \sim_B Q$ just when the bisimulation-equivalence of P and Q is witnessed by the bisimulation B.

We remind the reader that if P and Q are variable-open, then we say that they are bisimilar if they are pairwise bisimilar under all valuations of their free variables.

Lemma 13. *Given a collection of bisimulations B_k $(k \in K)$, the following are all bisimulations*:

(1) $Id \overset{\text{def}}{=} \{\langle P, P\rangle \mid P \in \mathbf{Proc}\}$.
(2) $B_1 B_2 \overset{\text{def}}{=} \{\langle P, R\rangle \mid \exists Q\colon \langle P, Q\rangle \in B_1 \wedge \langle Q, R\rangle \in B_2\}$.
(3) $B_1^{-1} \overset{\text{def}}{=} \{\langle Q, P\rangle \mid \langle P, Q\rangle \in B_1\}$.
(4) $\bigcup_{k\in K} B_k$ *where K where k is a countable index set.*

Proof. Claims (1)–(3) follow from the fact that every bisimulation is an equivalence relation. To show claim (4) we proceed as follows. Let $\langle P, Q\rangle \in \bigcup_{k\in K} B_k$. Then, by the construction of $\bigcup_{k\in K} B_k$, there must exist $i \in K$ such that B_i witnesses the bisimilarity of P and Q. Since $B_i \subset \bigcup_{k\in K} B_k$, it follows that $\bigcup_{k\in K} B_k$ witnesses the bisimilarity of P and Q. □

Corollary 14. *If B_k $(k \in K)$ is a collection of bisimulations, then their reflexive, transitive closure $(\bigcup_k B_k)^*$ is a bisimulation. Furthermore, $\sim$ is an equivalence relation over* **Proc**.

Proof. For the first claim we apply the fact that bisimulations are closed under reflexivity, transitivity and union. Since each of the relations B_k are symmetric, $(\bigcup_k B_k)^*$ must also be symmetric. Whence $(\bigcup_k B_k)^*$ must be an equivalence relation. Furthermore, claim (4) of Lemma 13 asserts that countable unions of bisimulations are bisimulations. Since $\sim$ is the union of all bisimulations and since the reflexive, transitive closure of all bisimulations is a bisimulation, it follows that $\sim$ is an equivalence relation. □

2.1. Probabilistic bisimulation is a congruence

In this section we prove a congruence theorem inspired by Milner [49]. In order to simplify the proof, we adapt the approach of Van Glabbeek et al. [65]. Essentially, we reason in terms of the cMFs rather than in terms of the underlying transitions.

Main Theorem 15. *Probabilistic bisimulation is a congruence, i.e.*,

$$\forall C[\,\cdot\,] \in \mathbf{Con}\colon P \sim Q \Longrightarrow C[P] \sim C[Q].$$

Proof. We want to show that $B \overset{\text{def}}{=} \{\langle C[P], C[Q]\rangle \mid C[\,\cdot\,] \in \mathbf{Con},\, P \sim Q\}$ is a weak probabilistic bisimulation. We start by noting that any variable in $C[P]$ either occurs bound in P, free in P but bound in $C[\,\cdot\,]$, or free even in $C[P]$. Since we defined bisimulation equivalence on variable-open processes $C[P]$ and $C[Q]$ by considering $C[P]$ and $C[Q]$ under all possible valuations, we can eliminate from consideration variables that appear free in $C[P]$ and $C[Q]$ as well as variables that appear free in just $C[\,\cdot\,]$.

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, writing $\mathbf{Con}(P, Q)$ to mean the set of contexts such that $C[P]$ and $C[Q]$ have no free variables, we just assume as our *first inductive hypothesis* that

$$\forall C[\,\cdot\,] \in \mathbf{Con}(P, Q) \colon C[P] \sim_B C[Q] \tag{2.1}$$

for any P and Q such that $P \sim_B Q$ and the maximum on the number of free variables in P and Q is k.

We establish the first inductive hypothesis by an induction on the structure of $C[\,\cdot\,]$. We start by defining $\mu^n : \mathbf{Proc} \times \mathbf{PubAct} \times 2^{\mathbf{Proc}} \to [0, 1]$ as

$$\mu^n(P, \alpha, \mathbf{R}) = \sum_{\pi \in \mathit{Paths}^n(P,\alpha,\mathbf{R})} \mathrm{Prob}\,[\pi]\,,$$

where $\mathit{Paths}^n(P, \alpha, \mathbf{R})$ denotes the set of all α-paths into $\mathbf{R}$ from P that can be derived by a proof tree of height at most n from the rules of Fig. 2. Intuitively, $\mu^n(P, \alpha, \mathbf{R})$ measures the cumulative mass of reaching a process in $\mathbf{R}$ from P via an α-action preceded by at most $n-1$ silent actions. We will adopt the convention that $\mu^0(P, \alpha, \mathbf{R}) = 0$. Let us write $P \sim_B^n$ when, given the equivalence relation $B \subseteq \mathbf{Proc} \times \mathbf{Proc}$, we have:

(1) $\forall \mathbf{R} \in \mathbf{Proc}/_B . \forall \alpha \in \mathbf{PubAct} \colon \mu^n(P, \alpha, \mathbf{R}) = \mu^n(Q, \alpha, \mathbf{R})$,
(2) if P and Q are blocked, then $\forall \alpha \in \mathbf{Act} \colon \mathit{Norm}(P, \alpha) = \mathit{Norm}(Q, \alpha)$,
(3) if either P or Q is unblocked, then

$$\forall \mathbf{R} \in \mathbf{Proc}/_B \colon \mathrm{Prob}\left[P \rightarrow_{\text{redxn}} \mathbf{R}\right] = \mathrm{Prob}\left[Q \rightarrow_{\text{redxn}} \mathbf{R}\right].$$

Since $\mu(P, \alpha, \mathbf{R}) = \lim_{n\to\infty} \mu^n(P, \alpha, \mathbf{R})$ we need just show, via an induction on n, that for all $n \geqslant 0$, for all P and Q such that $P \sim Q$ and the maximum on the number of free variables in P and Q is $k+1$, we have

$$\forall C[\,\cdot\,] \in \mathbf{Con}(P, Q) \colon C[P] \sim_B^n C[Q]. \tag{2.2}$$

For our *second inductive hypothesis*, we may assume (2.2) for some $n \geqslant 0$ since the basis (when $n = 0$) is trivial. In proving (2.2) for $n+1$ we undertake a case analysis depending on the topmost operator of $C[\,\cdot\,]$ and only show the $\leqslant$ direction of the first condition on $\sim_B^n$ (since the reverse direction is completely symmetric).

(1) $C[\,\cdot\,] \equiv D[\,\cdot\,]$. It is easy to see that $\mu^{n+1}(C[P], \alpha, \mathbf{R}) = \mu^n(D[P], \alpha, \mathbf{R}) \leqslant \mu(D[Q], \alpha, \mathbf{R}) = \mu(C[Q], \alpha, \mathbf{R})$. Furthermore, if $C[P], C[Q]$ are blocked then so are $D[P], D[Q]$. Therefore, we can calculate that $\mathit{Norm}(C[P], \alpha) = \mathit{Norm}(D[P], \alpha) = \mathit{Norm}(D[Q], \alpha) = \mathit{Norm}(C[Q], \alpha)$. If either $C[P]$ or $C[Q]$ are unblocked, then $\left[C[P] \rightarrow_{\text{redxn}} \mathbf{R}\right] = \mathrm{Prob}\left[D[P] \rightarrow_{\text{redxn}} \mathbf{R}\right] = \mathrm{Prob}\left[D[Q] \rightarrow_{\text{redxn}} \mathbf{R}\right] = \mathrm{Prob}\left[C[Q] \rightarrow_{\text{redxn}} \mathbf{R}\right]$.

(2) $C[\,\cdot\,] \equiv \mathit{in}(c, x).D[\,\cdot\,]$. For all α not of the form $\langle c, a\rangle$ we have that $\mu^{n+1}(C[P], \alpha, \mathbf{R}) = \mu^{n+1}(C[P], \alpha, \mathbf{R}) = 0$. The same holds true for any α of the form $\langle c, a\rangle$ where $a \geqslant \mathit{width}(C[P], c)$.

Let us consider α of the form $\langle c, a\rangle$ where $a < \mathit{width}(C[P], c)$. The second inductive hypothesis yields that $D[P] \sim D[Q]$ and the first inductive hypothesis gives us that $[a/x]D[P] \sim [a/x]D[Q]$. Whence $\mu^{n+1}(C[P], \alpha, \mathbf{R}) = \mu^{n+1}(C[Q], \alpha, \mathbf{R}) \leqslant \mu(C[Q], \alpha, \mathbf{R})$ since the limit of $\mu^{n+1}(C[Q], \alpha, \mathbf{R})$ as n approaches ∞ is $\mu(C[Q], \alpha, \mathbf{R})$.

Finally, since $\mathit{in}(c, x).D[P]$ and $\mathit{in}(c, x).D[Q]$ are blocked we must check that $\mathit{Norm}(\mathit{in}(c, x).D[P], \alpha) = \mathit{Norm}(\mathit{in}(c, x).D[Q], \alpha)$. For any α of the form $\langle c, a\rangle$, where $a < \mathit{width}(C[P], c)$, we see that $\mathit{Norm}(C[P], \alpha) = \mathit{Norm}(C[Q], \alpha) = 1$ and for all other α, we see that $\mathit{Norm}(C[P], \alpha) = \mathit{Norm}(C[Q], \alpha) = 0$.

(3) $C[\,\cdot\,] \equiv out(c, \mathrm{T}).D[\,\cdot\,]$. Proceeding as in (2) we can disregard all actions except those of the form $\langle \bar{c}, a\rangle$ where $a < width(C[P], c)$. For such α, we can calculate that $\mu^{n+1}(C[P], \alpha, \mathbf{R})$ is

$$\sum_{\{b\in\mathbb{N}\,|\,b=a \bmod width(C[P],c)\}} \mathrm{Prob}\left[\mathrm{T} \to_{\mathrm{term}} b\right].$$

We finish as in (2).

If $D[P]$ and $D[Q]$ are blocked, and $C[\,\cdot\,] \equiv out(c, \mathrm{a}).[\,\cdot\,]$, then the only action available to $C[P]$ and $C[Q]$ is of the form $\langle \bar{c}, a \bmod width(C[P], c)\rangle$. But then $Norm(C[P], \alpha) = Norm(C[Q], \alpha) = 1$ when α is of the form $\langle \bar{c}, a \bmod width(C[P], c)\rangle$ and $Norm(C[P], \alpha) = Norm(C[Q], \alpha) = 0$ otherwise. If either $C[P]$ or $C[Q]$ are unblocked then we note that any reduct of $C[P]$ and $C[Q]$ must be of the form $out(c, \mathrm{a}).R$ for some process R. Then we calculate the reduction probabilities as follows:

$$\begin{aligned}
&\mathrm{Prob}\left[out(c, \mathrm{T}).D[P] \to_{\mathrm{redxn}} out(c, \mathrm{a}).\mathbf{R}\right]\\
&\quad = \mathrm{Prob}\left[\mathrm{T} \to_{\mathrm{term}} \mathrm{a}\right] \cdot \mathrm{Prob}\left[D[P] \to_{\mathrm{redxn}} \mathbf{R}\right]\\
&\quad = \mathrm{Prob}\left[\mathrm{T} \to_{\mathrm{term}} \mathrm{a}\right] \cdot \mathrm{Prob}\left[D[Q] \to_{\mathrm{redxn}} \mathbf{R}\right]\\
&\quad = \mathrm{Prob}\left[out(c, \mathrm{T}).D[Q] \to_{\mathrm{redxn}} out(c, \mathrm{a}).\mathbf{R}\right].
\end{aligned}$$

(4) $C[\,\cdot\,] \equiv [\mathrm{T}].D[\,\cdot\,]$. Proceeding as in (2) we see that $\mu^{n+1}(C[P], \alpha, \mathbf{R}) = \mathrm{Prob}\left[\mathrm{T} \to_{\mathrm{term}} 1\right] \cdot \mu^{n}(D[P], \alpha, \mathbf{R}) \leqslant \mathrm{Prob}\left[\mathrm{T} \to_{\mathrm{term}} 1\right] \cdot \mu^{n}(D[Q], \alpha, \mathbf{R}) = \mu^{n+1}(C[Q], \alpha, \mathbf{R})$ since the only action available to any unblocked process is the reduction action. We finish as in case (2).

Since $C[P]$ and $C[Q]$ are unblocked, we check the reduction probabilities

$$\begin{aligned}
&\mathrm{Prob}\left[[\mathrm{T}].D[P] \to_{\mathrm{redxn}} \mathbf{R}\right] = \mathrm{Prob}\left[\mathrm{T} \to_{\mathrm{term}} \mathrm{a}\right] \cdot \mathrm{Prob}\left[D[P] \to_{\mathrm{redxn}} \mathbf{R}\right]\\
&\quad = \mathrm{Prob}\left[\mathrm{T} \to_{\mathrm{term}} \mathrm{a}\right] \cdot \mathrm{Prob}\left[D[Q] \to_{\mathrm{redxn}} \mathbf{R}\right] = \mathrm{Prob}\left[[\mathrm{T}].D[Q] \to_{\mathrm{redxn}} \mathbf{R}\right].
\end{aligned}$$

(5) $C[\,\cdot\,] \equiv \nu(c).D[\,\cdot\,]$. Since no rule in the operational semantics eliminates ν operator, we can restrict our attention to equivalence classes of processes of the form $\nu(c).\mathbf{R}$. For any $\alpha \in \mathbf{PubAct}$, the cumulative probability that an α-path takes $C[P]$ to $\nu(c).\mathbf{R}$ is the sum of the cumulative probability that an α-path takes $D[P]$ to $\mathbf{R}$ directly plus the cumulative probability of a $\langle c, a\rangle \cdot \langle \bar{c}, a\rangle$-path taking $D[P]$ to some $\mathbf{R}'$ and then from there to $\mathbf{R}$ via an α-path. We compute this probability as

$$\begin{aligned}
&\mu^{n+1}(C[P], \alpha, \nu(c).\mathbf{R})\\
&\quad = \mu^{n}(D[P], \alpha, \mathbf{R}) + \sum_{\mathbf{R}'\in\mathbf{Proc}/_B} \mu^{n}(D[P], \langle c, a\rangle \cdot \langle \bar{c}, a\rangle, \mathbf{R}') \cdot \mu^{n}(\mathrm{rep}\,\mathbf{R}', \alpha, \mathbf{R}).
\end{aligned} \tag{†}$$

Applying the second inductive hypothesis yields that (†) is upper-bounded by

$$\mu^{n}(D[Q], \alpha, \mathbf{R}) + \sum_{\mathbf{R}'\in\mathbf{Proc}/_B} \mu^{n}(D[Q], \langle c, a\rangle \cdot \langle \bar{c}, a\rangle, \mathbf{R}') \cdot \mu^{n}(\mathrm{rep}\,\mathbf{R}', \alpha, \mathbf{R}) = \mu^{n+1}(C[Q], \alpha, \nu(c).\mathbf{R}).$$

Let us assume that at least one of $C[P]$ and $C[Q]$ is unblocked. We note that any reduct of $\nu(c).D[P]$ (resp. $\nu(c).D[Q]$) must be of the form $\nu(c).R$ where R is a reduct of $D[P]$ (resp. $D[Q]$). Then,

$$\begin{aligned}
\mathrm{Prob}\left[\nu(c).D[P] \to_{\mathrm{redxn}} \nu(c).\mathbf{R}\right] &= \mathrm{Prob}\left[D[P] \to_{\mathrm{redxn}} \mathbf{R}\right]\\
&= \mathrm{Prob}\left[D[Q] \to_{\mathrm{redxn}} \mathbf{R}\right] = \mathrm{Prob}\left[\nu(c).D[Q] \to_{\mathrm{redxn}} \nu(c).\mathbf{R}\right]
\end{aligned}$$

On the other hand, if both $C[P]$ and $C[Q]$ are blocked then we have three cases to consider:

(a) α is a partial action or actual action on the bound channel c. Then $Norm(C[P], \alpha) = Norm(C[Q], \alpha) = 0$.
(b) $\alpha = \langle \tau_c, a\rangle$. Then, $Norm(C[P], \alpha) = Norm(D[P], \langle c, a\rangle \cdot \langle \bar{c}, a\rangle) = Norm(D[Q], \langle c, a\rangle \cdot \langle \bar{c}, a\rangle) = Norm(C[Q], \alpha)$.
(c) α is any other action. Then. $Norm(C[P], \alpha) = Norm(D[P], \alpha) = Norm(D[Q], \alpha) = Norm(C[Q], \alpha)$.

(6) $C[\,\cdot\,] \equiv D[\,\cdot\,] \mid S$ or $C[\,\cdot\,] \equiv S \mid D[\,\cdot\,]$. We only study the case where $C[\,\cdot\,] \equiv D[\,\cdot\,] \mid S$ since the same argument is employed in the other case. Since no transition rule in the operational semantics eliminates the $\mid$ operator, we can confine our attention to $\mathbf{R}$ of the form $\mathbf{R}_1 \mid \mathbf{R}_2$.

We start by showing that if $D[P]$, $D[Q]$, and S are all blocked, then $\forall\alpha \in \mathbf{Act}$: $Norm(D[P] \mid S, \alpha) = Norm(D[Q] \mid S, \alpha)$. This is immediate since for any $\alpha \in \mathbf{Act} - \Lambda_{actual}$ we have $Norm(D[P] \mid S, \alpha) = Norm(D[P], \alpha) + Norm(S, \alpha) = Norm(D[Q], \alpha) + Norm(S, \alpha) = Norm(D[Q] \mid S, \alpha)$. In case that $\alpha \in \Lambda_{actual}$ we note that the result follows by applying the previous argument after observing that $Norm(D[P] \mid S, \alpha \cdot \bar{\alpha}) = (Norm(D[P], \alpha) + Norm(S, \alpha)) \cdot (Norm(D[P], \bar{\alpha}) + Norm(S, \bar{\alpha}))$.

However, if either $D[P]$, $D[Q]$, or S is unblocked, we check the reduction probabilities by applying the inductive hypothesis after noting that $\mathrm{Prob}\left[D[P] \mid S \rightarrow_{\mathrm{redxn}} \mathbf{R}_1 \mid \mathbf{R}_2\right] = \mathrm{Prob}\left[D[P] \rightarrow_{\mathrm{redxn}} \mathbf{R}_1\right] \cdot \mathrm{Prob}\left[S \rightarrow_{\mathrm{redxn}} \mathbf{R}_2\right]$.

We finish by showing that $\forall\alpha \in \mathbf{PubAct}$: $\mu^{n+1}(D[P] \mid S, \alpha, \mathbf{R}_1 \mid \mathbf{R}_2) = \mu^{n+1}(D[Q] \mid S, \alpha, \mathbf{R}_1 \mid \mathbf{R}_2)$ via an induction on the maximum length of α-paths from $D[P] \mid S$ and $D[Q] \mid S$ to $\mathbf{R}_1 \mid \mathbf{R}_2$. The basis occurs when considering α-paths of maximum length 2 (i.e., paths which can have at most 1 silent action as a prefix). There are several cases to consider:

(a) Neither $D[P] \mid S$ nor $D[Q] \mid S$ have any silent actions (including reduction actions). In the case that α affects only $D[P]$ we see that

$$\begin{aligned}\mu^{n+1}(D[P] \mid S, \alpha, \mathbf{R}_1 \mid \mathbf{R}_2) &= \frac{Norm(D[P], \alpha)}{Norm(D[P] \mid S, \alpha)} \mu^{n}(D[P], \alpha, \mathbf{R}_1 \mid \mathbf{R}_2) \\ &\leqslant \frac{Norm(D[Q], \alpha)}{Norm(D[Q] \mid S, \alpha)} \mu^{n}(D[Q], \alpha, \mathbf{R}_1 \mid \mathbf{R}_2) = \mu^{n+1}(D[Q] \mid S, \alpha, \mathbf{R}_1 \mid \mathbf{R}_2).\end{aligned}$$

If α affects only S then

$$\begin{aligned}\mu^{n+1}(D[P] \mid S, \alpha, \mathbf{R}_1 \mid \mathbf{R}_2) &= \frac{Norm(S, \alpha)}{Norm(D[P] \mid S, \alpha)} \mu^{n}(S, \alpha, \mathbf{R}_1 \mid \mathbf{R}_2) \\ &= \frac{Norm(S, \alpha)}{Norm(D[Q] \mid S, \alpha)} \mu^{n}(S, \alpha, \mathbf{R}_1 \mid \mathbf{R}_2) = \mu^{n+1}(D[Q] \mid S, \alpha, \mathbf{R}_1 \mid \mathbf{R}_2).\end{aligned}$$

If α of the form $\beta \cdot \bar{\beta}$ with β affecting $D[P]$ and $\bar{\beta}$ affecting S then

$$\begin{aligned}&\mu^{n+1}(D[P] \mid S, \alpha, \mathbf{R}_1 \mid \mathbf{R}_2) \\ &\quad= \frac{Norm(D[P], \beta)}{Norm(D[P] \mid S, \beta)} \mu^{n}(D[P], \beta, \mathbf{R}_1 \mid \mathbf{R}_2) \cdot \frac{Norm(S, \bar{\beta})}{Norm(D[P] \mid S, \bar{\beta})} \mu^{n}(S, \bar{\beta}, \mathbf{R}_1 \mid \mathbf{R}_2) \\ &\quad\leqslant \frac{Norm(D[Q], \beta)}{Norm(D[Q] \mid S, \bar{\beta})} \mu^{n}(D[Q], \beta, \mathbf{R}_1 \mid \mathbf{R}_2) \cdot \frac{Norm(S, \bar{\beta})}{Norm(D[Q] \mid S, \bar{\beta})} \mu^{n}(S, \bar{\beta}, \mathbf{R}_1 \mid \mathbf{R}_2) \\ &\quad= \mu^{n+1}(D[Q] \mid S, \alpha, \mathbf{R}_1 \mid \mathbf{R}_2).\end{aligned}$$

(b) Both $D[P] \mid S$ and $D[Q] \mid S$ have silent actions. Therefore, each path in either LTS is of the form $D[P/Q] \mid S \xrightarrow{\tau} \mathbf{R}_1' \mid \mathbf{R}_2' \xrightarrow{\alpha} \mathbf{R}_1 \mid \mathbf{R}_2$. If the silent transition is not a reduction action, then we know that τ must be a transition of either $D[P]$ or S but not both and the channel on which τ occurs is a channel of either $D[P]$ or S but not both (since the scope of the binding ν does not extend over both components). Thus

$$\mathrm{Prob}\left[D[P] \mid S \xrightarrow{\tau} \mathbf{R}_1' \mid \mathbf{R}_2'\right] = \begin{cases} \mathrm{Prob}\left[D[P] \xrightarrow{\tau} \mathbf{R}_1'\right] & \text{if } \tau \text{ affects } D[P] \text{ and} \\ \mathrm{Prob}\left[S \xrightarrow{\tau} \mathbf{R}_2'\right] & \text{if } \tau \text{ affects } S. \end{cases}$$

In the case that τ is a reduction action, we have already calculated that $\mathrm{Prob}\left[D[P] \mid S \rightarrow_{\mathrm{redxn}} \mathbf{R}_1' \mid \mathbf{R}_2'\right] = \mathrm{Prob}\left[D[P] \mid S \rightarrow_{\mathrm{redxn}} \mathbf{R}_1' \mid \mathbf{R}_2'\right]$. Thus we do not have to re-normalise the probabilities of silent actions when calculating the cMFs of processes combined with a $\mid$ operator; we need only re-normalise the probabilities of the final public action. To do so, we simply apply the arguments of the previous case.

(c) Either $D[P] \mid S$ or $D[Q] \mid S$ have a silent action (but not both). We dispose of the case where the silent action is not a reduction action using the observation of the previous case: the channel on which the silent action occurs is not shared between $D[P]$ (resp. $D[Q]$) and S.

If it is a reduction step, then we make use of the idempotence of reduction (Lemma 6) to argue as in the previous case. In particular, one component of the parallel composition will have a probability one reduction to itself and a probability zero reduction to any other process. Thus, as in the previous case, we need not re-normalize probabilities of silent actions.

So much for the basis. We will assume $\forall \alpha \in \mathbf{PubAct}\colon \mu^{n+1}(D[P] \mid S, \alpha, \mathbf{R}_1 \mid \mathbf{R}_2) = \mu^{n+1}(D[Q] \mid S, \alpha, \mathbf{R}_1 \mid \mathbf{R}_2)$ for all paths of length at most k. To show the $k+1$ case we note that an at-most-$(k+1)$-length α-path consists of at most a single silent action followed by an at-most-k-length α-path. We finish by applying the arguments of the basis and using the inductive hypothesis at the appropriate points. $\square$

3. The evaluation of expressions

The abstract operational semantics given in Section 1.3.4 is overly general in two respects: (1) it allows a process to take actions that lack either an input or an output, and (2) it does not impose a distribution on the choice of action to perform. This generality allows the operational semantics to capture the behaviour of the process in any arbitrary environment and under any possible schedule. Hence, it provides a good basis for studying the security of protocols written as processes. However, in practice, when evaluating a process, we will impose a probabilistic schedule on actions and we will require that each evaluation step consists of an input and a matching output. In this section we discuss how scheduling actions is handled and define the evaluation of a process using a scheduler.

3.1. Probabilistic scheduling

Our goal is to provide a language-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. 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 LTS of a process without specifying a particular schedule. When we apply a scheduler to a LTS, we get probabilistically well-defined behaviour: the scheduler selects a label and then a particular transition labelled with the chosen label occurs with probability given by the LTS.

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 (i.e., action labels) rather than the messages themselves (the individual communications or transitions labelled by those actions). 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 make decisions solely on the basis of the actions given (rather than the communications that are labelled by those actions), we blind the scheduler to all information derived from the syntax of PPC (as opposed to the network behaviour 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), we argue that the adversary still has significant power. If each action available to a process labels exactly one transition, then the scheduler essentially picks individual transitions from the set of transitions according to some arbitrary distribution (dictated by the scheduler's behaviour). We believe that any security protocol can be systematically rewritten so that at each step of the evaluation, each *actual action* labels at most one transition. 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 transitions labelled by the same actual action. We note that, in general, we cannot make a guarantee of this sort for partial actions: what would it mean to disambiguate input actions in this fashion? However, since we will reason about properties of processes solely in terms of the observable behaviour generated by a sequence of actual actions (see Definition 31), it seems reasonable to believe that reasoning in such terms captures security under all possible adversaries who can schedule individual communications (as embodied by transitions). In fact, the examples that we study in this paper all have the property that each action labels at most one transition. In short, restricting the scheduler to operating on labels rather than specific transitions seems to be a reasonable solution since we believe that security protocols can be cast into a form that essentially allows the scheduler to pick individual communications.

Definition 16. A *scheduler* $\mathsf{S}\colon (2^{\mathbf{Act}} \setminus \emptyset) \to \mathbf{Act}$ is a probabilistic poly-time Turing machine that takes as input a non-empty set A of actions and selects one of those actions such that $\sum_{a\in A} \mathrm{Prob}\,[\mathsf{S}(A) = a] = 1$. We define **Sched** as the set of all schedulers.

The condition $\sum_{a\in A} \mathrm{Prob}\,[\mathsf{S}(A) = a] = 1$ forces the scheduler to always pick some action out of its input i.e., this condition forces the scheduler to make progress each time it is invoked. We note that the time bound on the running time of a scheduler is polynomial in the size of the set of actions given to it as input.

We remark that the scheduler need not give priority to reduction actions or actions on bindable channels since the operational semantics is constructed to ensure that the priorities on actions are correctly observed. We define **Sched** to be the set of all schedulers.

3.2. Evaluating variable-closed processes with schedulers

Here we show how to evaluate a variable-closed PPC process using a scheduler S. We extend this definition to all PPC processes by quantifying over all valuations of the free variables. To evaluate a variable-closed process under a scheduler, we simply compute the set of possible actions that the process can take, have the scheduler pick one of those actions, and then take a transition labelled by that action according to the distribution given by the LTS for the process.

We formalise this idea in two stages. First, we define the set of possible actions available to a variable-closed process. Then, we define what it means to take a transition under a particular scheduler.

We define the set $\{\alpha \mid \exists Q \in \mathbf{Proc} \mid \mathrm{Prob}\,[P \xrightarrow{\alpha} Q] > 0\}$ to be the *action set of a variable-closed process* P, denoted by $\Lambda(P)$. We say that a variable-closed process P*, with probability p, evaluates to Q via an α-transition under scheduler* S, written $P \xrightarrow{\alpha[p]}_{\mathsf{S}} Q$, just when:

(1) There is a transition labelled α from P to Q and
(2) $p = \mathrm{Prob}\,[\mathsf{S}(\Lambda(P)) = \alpha] \cdot \mathrm{Prob}\,[P \xrightarrow{\alpha} Q]$.

Basically, the probability that P can evaluate to Q under scheduler S via an α-transition is simply the product of the probability that P can take an α-transition to Q and the probability that the scheduler S elects to perform an α action.

Lemma 17. $\sum_{\alpha\in\mathbf{Act}} \sum_{Q\in\mathbf{Proc}} \mathrm{Prob}\left[P \xrightarrow{\alpha}_{\mathsf{S}} Q\right] \leqslant 1$.

Proof. Reordering the sums and expanding $\xrightarrow{\alpha}_{\mathsf{S}}$, we get

$$\sum_{\alpha\in\mathbf{Act}} \mathrm{Prob}\,[\mathsf{S}(\Lambda(P)) = \alpha] \cdot \sum_{Q\in\mathbf{Proc}} \mathrm{Prob}\,[P \xrightarrow{\alpha} Q]\,.$$

By Lemma 7 we know that $\forall\alpha \in \mathbf{Act}$ we have $\mathrm{Prob}\,[P \xrightarrow{\alpha} \mathbf{Proc}] \leqslant 1$ and, from the definition of a scheduler, we know that $\sum_{\alpha\in\mathbf{Act}} \mathrm{Prob}\,[\mathsf{S}(\Lambda(P)) = \alpha] = 1$. □

Let us write $e \cdot \pi$ to mean the path obtained by prefixing a path π with the edge e. Writing source e, label e, and Prob $[e]$, respectively, for the source process, action-label, and probability of an edge, we can then define *the probability of the path* π *under the scheduler* S, written $\mathrm{Prob}\,[\mathsf{S}(\pi)]$, inductively:

(1) $\mathrm{Prob}\,[\mathsf{S}(e)] = \mathrm{Prob}\,[\mathsf{S}(\Lambda(\text{source } e)) = \text{label } e] \cdot \mathrm{Prob}\,[e]$ and
(2) $\mathrm{Prob}\,[\mathsf{S}(e \cdot \pi)] = \mathrm{Prob}\,[\mathsf{S}(e)] \cdot \mathrm{Prob}\,[\mathsf{S}(\pi)]$.

We can then extend the definition of cMFs to account for the behaviour of a scheduler.

Definition 18. We define $\omega\colon \mathbf{Proc} \times \mathbf{PubAct} \times 2^{\mathbf{Proc}} \times \mathbf{Sched}$, *the cumulative probability distribution function* (cPDF), by the equation

$$\omega(P, \alpha, \mathbf{R}, \mathsf{S}) = \sum_{\pi \in Paths(P,\alpha,\mathbf{R})} \mathrm{Prob}\,[\mathsf{S}(\pi)]$$

We prove that the cPDF for a process P, unlike the cMF of that process, is a probability distribution on transitions.

Lemma 19. $\forall P \in \mathbf{Proc}.\forall \mathsf{S} \in \mathbf{Sched}\colon \sum_{\alpha \in \mathbf{PubAct}} \omega(P, \alpha, \mathbf{Proc}, \mathsf{S}) \leqslant 1.$

Proof. The proof is very similar to the proof of Lemma 11 except we make use of Lemma 17. Let us write $\omega^k(P, \alpha, \mathbf{R}, \mathsf{S})$ to denote that the value $\omega(P, \alpha, \mathbf{R}, \mathsf{S})$ is computed over at most k-length paths. To justify the inductive hypothesis we note that the value $\sum_{\alpha \in \mathbf{PubAct}} \omega(P, \alpha, , \mathsf{S})$, when restricted to at most k-length paths, can be calculated as

$$\sum_{\alpha \in \mathbf{PubAct}} \sum_{Q \in \mathbf{Proc}} \mathrm{Prob}\left[P \xrightarrow{\alpha} \mathsf{Q}\right] + \sum_{\tau \in \Lambda_{silent}} \sum_{Q \in \mathbf{Proc}} \mathrm{Prob}\left[P \xrightarrow{\tau} Q\right] \cdot \sum_{\alpha \in \mathbf{PubAct}} \omega^{k-1}(Q, \alpha, \mathbf{Proc}, \mathsf{S})).$$

The inductive hypothesis gives us that $\omega^{k-1}(Q, \alpha, \mathbf{Proc}, \mathsf{S}) \leqslant 1$. From Lemma 17 we know that

$$\sum_{\alpha \in \mathbf{PubAct}} \sum_{Q \in \mathbf{Proc}} \mathrm{Prob}\left[P \xrightarrow{\alpha} \mathsf{Q}\right] + \sum_{\tau \in \Lambda_{silent}} \sum_{Q \in \mathbf{Proc}} \mathrm{Prob}\left[P \xrightarrow{\tau} Q\right] \leqslant 1. \qquad \square$$

The following corollary is immediate.

Corollary 20. $\forall P \in \mathbf{Proc}.\forall \mathbf{R} \subseteq \mathbf{Proc}.\forall \mathsf{S} \in \mathbf{Sched}\colon \sum_{\alpha \in \mathbf{PubAct}} \omega(P, \alpha, \mathbf{R}, \mathsf{S}) \leqslant 1.$

Unlike the proof that cMF is well-defined, we can prove that the cPDF is a distribution on transitions because the scheduler assigns a distribution to actions while the LTS assigns a distributions to transitions labelled with the same action.

4. PPC evaluates in polynomial-time

In this section we construct a Turing machine $M_{\mathcal{P}}$ that evaluates the variable-closed expression $\mathcal{P}$. This machine takes as input a value for the security parameter and the program for the scheduler and then evaluates the expression by selecting the indicated process and performing alternating reduction and communication steps. The communication steps are chosen by the scheduler given as input. We will also prove that this mechanical evaluator runs in time polynomial in the length of the security parameter. Our proof of the polynomial time-bound will proceed in three 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 length of the security parameter. Then, we will show that each action along that path takes time polynomial in the length of the security parameter to evaluate. Finally, we will show that the scheduling decision also takes time polynomial in the length of the security parameter.

Before we continue, it will be useful to define a pair of metrics—number of inputs and number of outputs—on expressions. Let $\mathcal{P}$ be a variable-closed expression. We inductively define $\mathsf{Inputs}(\mathcal{P})$, the *number of inputs in* $\mathcal{P}$, as

$$\begin{aligned}
\mathsf{Inputs}(\emptyset) &= 0,\\
\mathsf{Inputs}(\nu(c).\mathcal{Q}) &= \mathsf{Inputs}(\mathcal{Q}),\\
\mathsf{Inputs}(in(c, x).\mathcal{Q}) &= 1 + \mathsf{Inputs}(\mathcal{Q}),\\
\mathsf{Inputs}(out(c, \mathrm{T}).\mathcal{Q}) &= \mathsf{Inputs}(\mathcal{Q}),\\
\mathsf{Inputs}([\mathrm{T}].\mathcal{Q}) &= \mathsf{Inputs}(\mathcal{Q}),\\
\mathsf{Inputs}(\mathcal{Q}_1 \mid \mathcal{Q}_2) &= \mathsf{Inputs}(\mathcal{Q}_1) + \mathsf{Inputs}(\mathcal{Q}_2),\\
\mathsf{Inputs}(!_{q(\eta)}.\mathcal{Q}) &= q(\eta) \cdot \mathsf{Inputs}(\mathcal{Q}),
\end{aligned}$$

and $\mathsf{Outputs}(\mathcal{P})$, the *number of outputs in* $\mathcal{P}$, as

$$\begin{aligned}
\mathsf{Outputs}(\emptyset) &= 0,\\
\mathsf{Outputs}(\nu(c).\mathcal{Q}) &= \mathsf{Outputs}(\mathcal{Q}),\\
\mathsf{Outputs}(in(c,x).\mathcal{Q}) &= \mathsf{Outputs}(\mathcal{Q}),\\
\mathsf{Outputs}(out(c,\mathrm{T}).\mathcal{Q}) &= 1 + \mathsf{Outputs}(\mathcal{Q}),\\
\mathsf{Outputs}([\mathrm{T}].\mathcal{Q}) &= \mathsf{Outputs}(\mathcal{Q}),\\
\mathsf{Outputs}(\mathcal{Q}_1 \mid \mathcal{Q}_2) &= \mathsf{Outputs}(\mathcal{Q}_1) + \mathsf{Outputs}(\mathcal{Q}_2),\\
\mathsf{Outputs}(!_{q(\eta)}.\mathcal{Q}) &= q(\eta)\cdot \mathsf{Outputs}(\mathcal{Q}).
\end{aligned}$$

By definition, it is clear that $\mathsf{Inputs}(\mathcal{P})$ and $\mathsf{Outputs}(\mathcal{P})$ are polynomials in η that are positive for all input values.

4.1. A Turing machine for process evaluation

In this section we will define an *evaluator for the variable-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. The machine then evaluates the variable-closed process $[i/\eta]\mathcal{P}$ as follows:

(1) It first performs a reduction step. If the process expression is blocked, performing a reduction step will not skew any evaluation probabilities since reductions are idempotent (Lemma 6).
(2) It then computes the action set of $[i/\eta]\mathcal{P}$ and invokes the scheduler upon it. The scheduler will select the next action to perform.
(3) Next, it picks a transition of the process labelled by the chosen action and performs the appropriate communication step. It does this by making a list of all the inputs and outputs in the process that can generate the appropriate action (recall that since the scheduler is perceptible the action must consist of an input and an output) and computes all the combinations of inputs and outputs. It then picks one of those input–output pairs uniformly at random. Lemma 8 assures us that picking an input–output pair uniformly at random conforms to the distribution on transitions given by the LTS for the process.
(4) The machine repeats steps 1–3 until the action set contains no silent actions or actual actions, at which point it halts.

We will refer to steps 1–3 as a single *evaluation step* in the evaluation of a variable-closed expression. The evaluation of variable-open expressions is defined as evaluation under a valuation of the free variables of the expression.

The formal construction of the evaluator is given in Section 4.2. The construction, not surprisingly, closely follows the sketch just given. However, in order to simplify the proof that any variable-closed expression can be evaluated in time polynomial in the security parameter, we make a few simplifying assumptions.

λ-substitutions. Since $\mathcal{P}$ is a variable-closed expression, every term with variables must appear in the scope of sufficiently many inputs to bind those variables. For example, the term $\mathrm{T}(x_1,\ldots,x_i)$ must appear in the scope of at least i inputs, each of those inputs binding exactly one of those variables. Furthermore, a term is only reduced when it becomes exposed, i.e., is no longer in the scope of any input. At this point, the term has no variables since values must 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 $\mathrm{T}(x_1,\ldots,x_i)$ and a communication step that substitutes a for x_k, our Turing machine will create the term $\lambda x_k.\mathrm{T}(x_1,\ldots,x_i)\ a$ rather than $\mathrm{T}(x_1,\ldots,x_{k-1},a,x_{k+1},\ldots,x_i)$ (assuming, of course, that $1\leqslant k\leqslant i$). The actual substitution of the value into the variable and subsequent term-reduction to an atom is delayed until a reduction step is performed. This confers two advantages: (1) we can quantify the time required to evaluate some substitution instance of a term in terms on the original term with free variables and (2) we can quantify the change in the length of a term due to the substitution of a value for a variable (without knowing the number of occurrences of a free variable in a term, it would be impossible to quantify the change in a term's length, otherwise).

Padding. Our second assumption has to do with 'padding' the process being evaluated with blank space so that no evaluation step causes the length of the process to increase. There are two places where padding is required: (1) variables (since values will be substituted into them) and (2) terms (since they will be reduced to atoms, and have λs added around them). Let us start with the space required for variables. Each variable needs enough space to write down any possible value that can be substituted into that variable. Now, the values that get substituted come via communication steps i.e., they come over channels. Since each channel has a bandwidth, we can use the bandwidth to determine the size of the variable. In particular, 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 us write $W(\mathcal{P})$ for the set of bandwidth polynomials associated with channel names appearing in $\mathcal{P}$. Clearly, $W(\mathcal{P})$ consists of a finite number (say *M*) of univariate polynomials. We can assume that the maximum number of terms in any of the polynomials in $W(\mathcal{P})$ is *N*, and we will index these terms from 0 to $N-1$. We define a_i $(0 \leqslant i < N)$ as $\max_{j=1}^{M} \left|a_{j,i}\right|$ where $a_{j,i}$ is the coefficient of the *i*th term of the *j*th polynomial in $W(\mathcal{P})$. We then define the coefficient of the *i*th term of the polynomial $\chi'(x)$ as a_i i.e., $\chi'(x) = \sum_{i=0}^{N-1} a_i \cdot x^i$. It is immediate from the construction of $\chi'(x)$ that, $\forall p(x) \in W(\mathcal{P}).\forall i \in \mathbb{N}\colon\ p(i) \leqslant \chi'(i)$ i.e., $\chi'(x)$ is an upper-bound on any polynomial in $W(\mathcal{P})$. Thus, $2^{\chi'(\eta)}$ is an upper-bound on the largest value that can be substituted for any variable in $\mathcal{P}$. However, $\chi'(\eta)$ might be smaller than the number of characters required to write down each variable name. The number of distinct variable names, which we shall denote by $\upsilon(\mathcal{P})$, is, clearly, linear in the length of $\mathcal{P}$. Furthermore, $\upsilon(\mathcal{P})$ must be a polynomial in the security parameter since every syntactic element of an expression takes constant space except for replications which require space polynomial in the security parameter. To ensure that we have enough space to write down any value that can be substituted into a variable and write down the name of the variable itself, we define $\chi(x) = \chi'(x) + \upsilon(P)$ which is, clearly, a univariate polynomial in the security parameter. We then assume that each variable in $\mathcal{P}$ is of size $\chi(\eta)$.

Now for the λ-substitutions that we create during a communication step. Each single λ-substitution instance of a term requires a constant c_λ amount of space for the λ and enough space for the variable name and the value to substitute for the variable name. The amount of space required to write down the variable name is $\chi(\eta)$. It is also the amount of space required to write down the value to be substituted for that variable. Thus each λ-substitution takes $2 \cdot \chi(\eta) + c_\lambda$ space. To determine the total amount of space required for creating λ-substitutions around a term over the entire course of evaluation, we need to determine the maximum number of inputs that can be binding free variables in the term. Since there are $\mathsf{Inputs}(\mathcal{P})$ inputs in $\mathcal{P}$, we need $\mathsf{Inputs}(\mathcal{P}) \cdot (2 \cdot \chi(\eta) + c_\lambda)$ space for each term. However, it may be that the length of the term c_{T} is greater than this value. To make sure that we leave enough space to write down the term (without any λ-substitutions), we stipulate that the space required for a term T be $p_{\mathrm{T}}(x) \overset{\text{def}}{=} c_{\mathrm{T}} + \mathsf{Inputs}(\mathcal{P}) \cdot (2 \cdot \chi(x) + c_\lambda)$. Then, as we perform communication steps, we will not have to push symbols aside in order to make space for the λ-substitution instance we want to create.

Finally, we need to check that if we define the space that a term T takes to be $p_{\mathrm{T}}(\eta)$, then when we reduce the term to an atom during a reduction step, we do not run out of space to write the value of the atom down. If the term appears in an output then, since outputs truncate the message sent according to the bandwidth of the channel on which the output occurs, we need only write down the $\chi(\eta)$ least significant bits of the value obtained. But $\chi(\eta) \leqslant p_{\mathrm{T}}(\eta)$ just when both $\mathsf{Inputs}(\mathcal{P})$ and $\mathsf{Outputs}(\mathcal{P})$ are greater than zero. If either $\mathsf{Inputs}(\mathcal{P})$ or $\mathsf{Outputs}(\mathcal{P})$ is zero then, technically, the inequality $\chi(\eta) \leqslant p_{\mathrm{T}}(\eta)$ is false. However, this is not a significant issue since evaluation cannot proceed if there are either no inputs or no outputs (recall that an evaluation step requires an input and an output). For a term appearing in a match, 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 is sufficient since all we do is to either erase the match (assuming the match was passed) or erase the entire expression (assuming the match was failed).

As a consequence of these assumptions on the length of terms and variables, each reduction step does not increase the length the process. Furthermore, neither do communication steps, since we make sure to leave enough space for the λs that substitutions create. To summarise our assumptions, we delay the substitution of values into variables until a reduction step touches the relevant term, and we assume that every term and variable has enough space around it to ensure that reduction and communication steps never increases the length of a process.

4.2. The evaluator for variable-closed expressions

We will now construct the machine $M_{\mathcal{P}}$. The machine has three input tapes, two working tapes, and two output tapes. It starts with a value *i* for the security parameter written out in unary 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. Writing the security parameter in unary allows us to equate the length of the security parameter with its value, which is convenient in stating the theorems. Additionally, this is standard practice in the cryptographic community (see [31,44]). The third input tape starts with the variable-closed expression $\mathcal{P}$ that $M_{\mathcal{P}}$ is supposed to evaluate. The first step that $M_{\mathcal{P}}$ undertakes is to substitute i for η in $\mathcal{P}$. It then copies $[i/\eta]\mathcal{P}$ 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 $\sqcup$, as a blank symbol that we use to delete symbols by overwriting them as well as add padding to terms and variables so that we leave sufficient space as indicated by the discussion in Section 4.1). As we reduce $[i/\eta]\mathcal{P}$ and perform communication steps, we will rewrite $[i/\eta]\mathcal{P}$ 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 6), 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 $[i/\eta]\mathcal{P}$ (it is easy to determine if a term or match is exposed—every time an input operator, say $in(c, x).[i/\eta]\mathcal{Q}$, is encountered, we consider every subexpression of $[i/\eta]\mathcal{Q}$ 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 fact that terms are exactly the probabilistic polynomial-time functions (see Section 1.1). 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 $[i/\eta]expr\,P$ can be determined by inspecting $\mathcal{P}$. Let $\Delta\colon \mathbf{VarClosedExpr} \rightarrow 2^{\mathbf{Term}}$ be a function from variable-closed expressions to sets of terms such that $\Delta(\mathcal{P})$ is the set of terms that appear as syntactic elements of $\mathcal{P}$. Since every exposed term must be a substitution instance of terms in $\Delta(\mathcal{P})$ (recall that $\mathcal{P}$ is assumed to be variable-closed, and communication steps create λ-substitution instances of terms), we can evaluate each substitution instance $\lambda x_1, \ldots, \lambda x_k.\mathrm{T}\ a_1, \ldots, a_k$ 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 $width([i/\eta]\mathcal{P}, c)$ 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. It is easy to check that the distribution on process induced by performing a reduction step on $[i/\eta]\mathcal{P}$ is exactly the same as that specified by the definition of the reduction function $\rightarrow_{\mathrm{redxn}}$ (see Section 1.3.1), i.e., $M_{\mathcal{P}}$ really does compute a reduction step at this stage of evaluation.

We then need to perform a communication step. We start by computing the set of input/output pairs that $[i/\eta]\mathcal{P}$ 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 $\langle in(c, a), out(c, \mathrm{a})\rangle$ for some c and a (whether c is private or not). This is essentially the action set of the process restricted to actual actions (we get the action set by rewriting $\langle in(c, a), out(c, \mathrm{a})\rangle$ as $\langle c, a\rangle \cdot \langle \bar{c}, a\rangle$). We augment this set of pseudo-actions with locating contexts (see Section 1.2) that we will use to identify the locations of the particular input and output making up a particular possible communication step. We convert E into a set of actions A in the obvious way and use the given perceptible scheduler S to select one of the actions. Having selected an action out of A, we then eliminate all input–output pairs from E that do not produce the same observable as the chosen action. Finally, we pick one of the remaining input–output pairs uniformly at random and perform the indicated substitution. 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. It is easy to decide if an action is public (this takes time linear in the length of the process being evaluated). Clearly, at the termination of this step $M_{\mathcal{P}}$ has correctly performed the substitution indicated by an action available to $\mathcal{P}$. That it has done so with the right probability follows from Lemma 21. 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 variable-closed expression. Thus each variable-closed expression has an evaluator associated with it. The reader will no doubt appreciate that a single Turing machine able to evaluate any variable-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 (in this case the run-time will, of course, be polynomial in the security parameter and the sizes of the values selected for the various free variables of the expression). We leave the details of these extensions to the interested reader.

Lemma 21. *Let $\mathcal{P}$ be a variable-closed expression and $M_{\mathcal{P}}$ be the associated evaluator. Then $M_{\mathcal{P}}(i, \mathsf{S})$ generates the observable o with the same probability that $[i/\eta]\mathcal{P}$ generates the observable o under the perceptible scheduler* S.

It is not difficult to extract the proof from the construction just given for evaluators.

4.3. A bound on the length of any evaluation sequence

Lemma 22. *Let $\mathcal{P}$ be a variable-closed process. Then, for all values i for the security parameter and perceptible schedulers* S, *during any evaluation of* $[i/\eta]\mathcal{P}$, *at most* $\min\{\mathsf{Inputs}(\mathcal{P})(i), \mathsf{Outputs}(\mathcal{P})(i)\}$ *evaluation steps occur.*

Proof. Consider any evaluation step that occurs 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 know that a reduction step cannot increase the number of inputs or outputs in a process (all it can do is to reduce the number of inputs and outputs by causing a match to fail). Since a communication step syntactically eliminates an input/output pair from the expression being evaluation, 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 $\min\{\mathsf{Inputs}(\mathcal{P})(i), \mathsf{Outputs}(\mathcal{P})(i)\}$ communication steps whence we can only have at most that many evaluation steps. □

Corollary 23. *Let $\mathcal{P}$ be a variable-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 22, we define the number of evaluating steps $h(x)$ that occur during the evaluation of a $\mathcal{P}$ as:

$$\min\{\mathsf{Inputs}(\mathcal{P})(x), \mathsf{Outputs}(\mathcal{P})(x)\} \leqslant \mathsf{Inputs}(\mathcal{P})(x) + \mathsf{Outputs}(\mathcal{P})(x)$$

Clearly, $h(\eta)$ is a polynomial in the security parameter. □

4.4. A bound on the time for an evaluation step

Definition 24. Let $\mathcal{P}$ be a variable-closed expression. We inductively define a polynomial $\mathsf{Length}(\mathcal{P})$, *the length of $\mathcal{P}$ including padding*, as follows:

$$\begin{aligned}
\mathsf{Length}(\emptyset) &= 1,\\
\mathsf{Length}(\nu(c).\mathcal{Q}) &= 1 + \mathsf{Length}(\mathcal{Q}),\\
\mathsf{Length}(in(c, x).\mathcal{Q}) &= 1 + \mathsf{Length}(\mathcal{Q}),\\
\mathsf{Length}(out(c, \mathrm{T}).\mathcal{Q}) &= 1 + p_{\mathrm{T}}(\eta) + \mathsf{Length}(\mathcal{Q}),\\
\mathsf{Length}([\mathrm{T}].\mathcal{Q}) &= 1 + p_{\mathrm{T}}(\eta) + \mathsf{Length}(\mathcal{Q}),\\
\mathsf{Length}(\mathcal{Q}_1 \mid \mathcal{Q}_2) &= 1 + \mathsf{Length}(\mathcal{Q}_1) + \mathsf{Length}(\mathcal{Q}_2),\\
\mathsf{Length}(!_{q(\eta)}.\mathcal{Q}) &= q(\eta) \cdot \mathsf{Length}(\mathcal{Q}).
\end{aligned}$$

It is clear from the definition that $\mathsf{Length}(\mathcal{P})$ is an always positive polynomial in η. We remind the reader that this definition accounts for the padding that we add to the expression to ensure that we never need to push symbols aside when we perform communication steps. It is easy to see that if we eliminate the padding, the length of an expression is still polynomial in the security parameter.

Lemma 25. *Let $[i/\eta]\mathcal{P}$ be a process and let the process $[i/\eta]\mathcal{Q}$ be the result of performing some number of evaluation steps on $[i/\eta]\mathcal{P}$. Then $\forall i \in \mathbb{N}\colon \mathsf{Length}(\mathcal{Q})(i) \leqslant \mathsf{Length}(\mathcal{P})(i)$.*

Proof. Each reduction step eliminates matches and rewrites output terms with values. Thus a reduction step cannot increase the length of a process since we have padded terms to ensure that there is plenty of space to write down the value to which the output term reduces.

Also, each communication step removes at least one input and output from $[i/\eta]\mathcal{P}$. Since the contribution to the length of $[i/\eta]\mathcal{P}$ 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 during communication steps, $\mathsf{Length}(\mathcal{Q})(i)$ cannot exceed $\mathsf{Length}(\mathcal{P})(i)$. □

Lemma 26. *Let $\mathcal{P}$ be a variable-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 length of the security parameter.*

Proof. Since $\mathcal{P}$ is blocked, it has no exposed matches or outputs that need reduction. Before the scheduler is invoked, we need to construct the action set of the process. 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 we are only using perceptible schedulers, every action that can be scheduled must be an actual action. Since an actual action consists of an input and an output, the set of actions we need to consider is of size at most $\mathsf{Inputs}(\mathcal{P}) \cdot \mathsf{Outputs}(\mathcal{P})$. Thus, the action set A of a process can be computed in time polynomial in the length of η. Finally, the scheduler will pick an element from A in time polynomial in the size of A. But A is clearly linear in the (padded) length of $\mathcal{P}$ since each action in the action set must consist of an input and an exposed, reduced output of $\mathcal{P}$. Whence the time needed for the entire procedure of running the scheduler and deciding which action to perform is at most a polynomial $s_{\mathcal{P}}(x) \overset{\text{def}}{=} c_{comm} \cdot (\mathsf{Inputs}(\mathcal{P}) \cdot \mathsf{Outputs}(\mathcal{P}) + o(\mathsf{Length}(\mathcal{P})))$ where c_{comm} is a constant and $o(x)$ is a univariate polynomial. Clearly, $s_{\mathcal{P}}$ is a univariate polynomial in the security parameter since the length of an expression is a polynomial in the security parameter. □

Lemma 27. *Let $\mathcal{P}$ be a variable-closed expression. Then the evaluator for $\mathcal{P}$, $M_{\mathcal{P}}$, performs a communication step in time polynomial in the length of $\mathcal{P}$ and η.*

Proof. Whenever $M_{\mathcal{P}}$ touches an exposed term $(\lambda x_1, \ldots, \lambda x_i.\mathrm{T})\, a_1, \ldots, a_k$, it evaluates T by evaluating the algorithm M_{T} at $a_1, \ldots, a_k$ (recall that since $\mathcal{P}$ is variable-closed, all exposed terms have no free variables) The runtime is bounded by the polynomial $q_{\mathrm{T}}(|a_1|, \ldots, |a_k|)$ (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 $\chi(\eta)$ whence the cost of evaluating a term is given by the polynomial $q_{\mathrm{T}}(\chi(x), \ldots, \chi(x))$ which is just a univariate polynomial t_{T} in 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

$$r_{\mathcal{P}}(x) \overset{\text{def}}{=} \mathsf{Length}(\mathcal{P}) \cdot \left(\sum_{k=1}^{m} t_{\mathrm{T}_k}(x) \right),$$

where the expression $\mathcal{P}$ contains just the terms $\mathrm{T}_1, \ldots, \mathrm{T}_m$.

Next, we need to select an action from the action set of the process using the scheduler. From Lemma 26, this can be done in time $s_{\mathcal{P}}(\eta)$ which is a polynomial in η and the length of $\mathcal{P}$. We then select an input–output pair that produces the same observable as the selected action:

(1) First, we make a list of all the input–output pairs eligible for the substitution step. This is done in time polynomial in the length of $\mathcal{P}$.
(2) Then, we select one of those actions uniformly at random. This procedure takes time polynomial in the number of input–output pairs that can produce the same observable as α. This is at most $\mathsf{Inputs}(\mathcal{P}) \cdot \mathsf{Outputs}(\mathcal{P})$ which is a polynomial in η.

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

Next we need to perform the chosen substitution. We can do this in one pass to substitute the value transmitted by the communication step into the free occurrences of the variable bound by the input, and a second pass to erase the selected input and output (using blank symbols to erase the unnecessary symbols). Thus we can perform a communication step in time $c_{\mathcal{P}} = c \cdot \mathsf{Length}(\mathcal{P})$.

Thus the time for an evaluation step is given by the univariate polynomial $e_{\mathcal{P}}(x) \overset{\text{def}}{=} r_{\mathcal{P}}(x) + s_{\mathcal{P}}(x) + d_{\mathcal{P}}(x) + c_{\mathcal{P}}(x)$ which is a polynomial in η (since the length of $\mathcal{P}$ is a polynomial in η). $\square$

4.5. A bound on evaluation time

Main Theorem 28. *Let $\mathcal{P}$ be a variable-closed expression and $M_{\mathcal{P}}$ its evaluator. Then $M_{\mathcal{P}}$ evaluates $\mathcal{P}$ in time polynomial in the security parameter.*

Proof. Before we evaluate the expression, we need to write out the padded form of the process $[i/\eta]\mathcal{P}$. This takes time polynomial in the security parameter since the padding is of length polynomial in the security parameter and the length of an expression without padding is polynomial in the security parameter. We shall call this polynomial $a(x)$.

We know that there are at most $h(x)$ evaluation steps during the evaluation of $\mathcal{P}$ (Lemma 23). Furthermore, each evaluation step takes time no more than $e_{\mathcal{P}}(x)$ (Lemma 27). Thus, the overall cost of evaluating $\mathcal{P}$ is $\Phi_{\mathcal{P}}(x) \overset{\text{def}}{=} a(x) + h(x) \cdot e_{\mathcal{P}}(x)$ which is just a univariate polynomial in the security parameter. $\square$

5. Asymptotic observational equivalence

Given a protocol and a specification, intuitively, the two are equivalent if no third party can reliably decide whether it is interacting with the protocol or the specification. The only information available to the third party as it attempts to determine with whom it is interacting is the observable behaviour of the entity it is studying. This idea lies at the heart of cryptographic intuitions about how to characterise security properties of protocols and primitives.

In this section we define an observational equivalence relation and explore the relationship between it and probabilistic bisimulation. We start by defining an observable. Essentially, an observable is a datum that an entity can 'see' when interacting with another entity.

Definition 29. An *observable o* is a pair $(c, a) \in \textbf{Channel} \times \mathbb{N}$. The set **Obs** is the set of all observables. A process *P generates the observable (c, a) under the scheduler* S, written as $P \rightsquigarrow_{\mathsf{S}} (c, a)$, just when while evaluating P, the scheduler S selects the action $\langle c, a\rangle \cdot \langle \bar{c}, a\rangle$. The probability that $P \rightsquigarrow_{\mathsf{S}} (c, a)$ is given by

$$\omega(P, \langle c, a\rangle \cdot \langle \bar{c}, a\rangle, \textbf{Proc}, \mathsf{S}) + \sum_{\substack{R \in \textbf{Proc}/\sim \\ \alpha \in \textbf{PubAct} \setminus \{\langle c,a\rangle \cdot \langle \bar{c},a\rangle\}}} \omega(P, \alpha, \mathbf{R}, \mathsf{S}) \cdot \text{Prob}\left[\text{rep } R \rightsquigarrow_{\mathsf{S}} (c, a)\right].$$

An observable must be generated via a sequence of public actions and silent actions.

Lemma 30. $\forall \mathsf{S} \in \textbf{Sched}.\forall P \in \textbf{Proc}.\forall o \in \textbf{Obs}$: $\text{Prob}\,[P \rightsquigarrow_{\mathsf{S}} o] \leqslant 1$.

Proof. As usual we proceed by an induction on the maximum length of paths. The value $\text{Prob}\,[P \rightsquigarrow_{\mathsf{S}} o]$ is computed as

$$\omega(P, \langle c, a\rangle \cdot \langle \bar{c}, a\rangle, \textbf{Proc}, \mathsf{S}) + \sum_{\substack{R \in \textbf{Proc}/\sim \\ \alpha \in \Lambda_{actual} \setminus \{\langle c,a\rangle \cdot \langle \bar{c},a\rangle\}}} \omega(P, \alpha, \mathbf{R}, \mathsf{S}) \cdot \text{Prob}\left[\text{rep } R \rightsquigarrow_{\mathsf{S}} (c, a)\right].$$

By the inductive hypothesis we know that $\text{Prob}\left[\text{rep } \mathbf{R} \rightsquigarrow_{\mathsf{S}} (c, a)\right] \leqslant 1$. Furthermore,

$$\omega(P, \langle c, a\rangle \cdot \langle \bar{c}, a\rangle, \textbf{Proc}, \mathsf{S}) + \sum_{\substack{R \in \textbf{Proc}/\sim \\ \alpha \in \Lambda_{actual} \setminus \{\langle c,a\rangle \cdot \langle \bar{c},a\rangle\}}} \omega(P, \alpha, \mathbf{R}, \mathsf{S}) = \sum_{\alpha \in \textbf{PubAct}} \omega(P, \alpha, \textbf{Proc}, \mathsf{S}).$$

Applying Lemma 19 yields the desired result. $\square$

The reader will have noticed that a scheduler can select partial actions in addition to actual and silent actions. We define the set **PSched** of *perceptible schedulers* to be the set of all schedulers satisfying

$$\forall \alpha \in \Lambda_{in} \cup \Lambda_{out} : \text{Prob}\,[\mathsf{S}(A) = \alpha] = 0.$$

A perceptible scheduler is simply a scheduler that always picks actual actions or silent actions. When considering the equivalence of two expressions, we will restrict the choice of schedulers to just the perceptible ones since we are interested only in communication steps that involve communication between two entities. This allows us, for example, to talk of a process generating the observable o via a sequence of only actual and silent actions, in keeping with our intuitive idea of evaluation as a series of communications between two entities.

We will say that $\mathcal{P}$ and $\mathcal{Q}$, written as $\mathcal{P} \simeq \mathcal{Q}$, are *asymptotically close* just when

$$\begin{aligned}&\forall\, o \in \mathbf{Obs}.\forall \mathsf{S} \in \mathbf{PSched}.\forall p(\cdot) \in \mathbf{Poly}.\exists i_0 \in \mathbb{N}.\forall i > i_0 :\\ &\quad \left|\text{Prob}\left[[i/\eta]\mathcal{P} \rightsquigarrow_{\mathsf{S}} o\right] - \text{Prob}\left[[i/\eta]\mathcal{Q} \rightsquigarrow_{\mathsf{S}} o\right]\right| \leqslant \frac{1}{p(i)}.\end{aligned}$$

Definition 31 (*Observational equivalence*). Let $\mathcal{P}$ and $\mathcal{Q}$ be two expressions and let $\mathbf{Val}(\mathcal{P}, \mathcal{Q})$ be the set of all valuations of free variables of $\mathcal{P}$ and $\mathcal{Q}$. We will say that $\mathcal{P}$ and $\mathcal{Q}$ are *observationally equivalent*, written as $\mathcal{P} \cong \mathcal{Q}$, if

$$\forall \xi \in \mathbf{Val}(\mathcal{P}, \mathcal{Q}).\forall \mathcal{C}[\,\cdot\,] \in \mathbf{CExpr} : \xi(\mathcal{C}[\mathcal{P}]) \simeq \xi(\mathcal{C}[\mathcal{Q}]).$$

It is straightforward to check that $\cong$ is a congruence. Hence, we state the following lemma without proof.

Lemma 32. $\cong$ *is a congruence.*

The next theorem establishes probabilistic bisimulation as a proof technique for demonstrating observational equivalences.

Theorem 33. $\forall i \in \mathbb{N} : [i/\eta]P \sim [i/\eta]Q \Longrightarrow \mathcal{P} \cong \mathcal{Q}$.

Proof. If two processes P and Q are bisimilar then we know that if they are blocked, the number of ways they can take any action α is the same for the two processes. Whence it follows for blocked bisimilar processes that the scheduler behaves in the same way for both processes (since the two processes must be able to take the same actions). If they are unblocked then the scheduler is not invoked since a reduction step must be performed. However, the fact that P and Q are bisimilar means that the reduction step will take P and Q to bisimilar processes with the same probability. Hence, if P and Q are bisimilar, they are asymptotically close. Since $\sim$ is a congruence (Theorem 15), we have the desired result. □

We note that each valuation ξ can be expressed as a context expression $\mathcal{C}_\xi[\,\cdot\,]$. Let $\mathcal{P}$ have k free variables and let ξ substitute the value a_i for the variable x_i. Then, we can capture this valuation in the context

$$\nu(c_1).\cdots.\nu(c_k).(out(c_1, \mathrm{a}_1) \mid \cdots \mid out(c_k, \mathrm{a}_k)) \mid in(c_1, x_1).\cdots.in(c_k, x_k).[\,\cdot\,].$$

Via a series of private communications, each of the variables x_i $(1 \leqslant i \leqslant k)$ is replaced with the values a_i just as ξ demands. Thus, $\cong$ can be defined over all variable-open expressions without explicitly dealing with valuations. The quantification over all context expressions automatically takes care of the quantification over all valuations.

6. A proof 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 Fig. 3. We now proceed to sketch justifications for the proof rules. Rules (CON), (RFLX), (TRN), and (SYM) are formalisations of $\cong$'s congruence properties.

$$\frac{\mathcal{P} \cong \mathcal{Q},\ \mathcal{C}[\,\cdot\,] \in \textbf{CExpr}}{\mathcal{C}[\mathcal{P}] \cong \mathcal{C}[\mathcal{Q}]} \qquad \text{(CON)}$$

$$\mathcal{P} \cong \mathcal{P} \qquad \text{(RFLX)}$$

$$\frac{\mathcal{P} \cong \mathcal{Q},\ \mathcal{Q} \cong \mathcal{R}}{\mathcal{P} \cong \mathcal{R}} \qquad \text{(TRN)}$$

$$\frac{\mathcal{P} \cong \mathcal{Q}}{\mathcal{Q} \cong \mathcal{P}} \qquad \text{(SYM)}$$

$$\mathcal{P} \mid \mathcal{Q} \cong \mathcal{Q} \mid \mathcal{P} \qquad \text{(P-SYM)}$$

$$\emptyset \mid \mathcal{P} \cong \mathcal{P} \qquad \text{(ZERO)}$$

$$(\mathcal{P} \mid \mathcal{Q}) \mid \mathcal{R} \cong \mathcal{P} \mid (\mathcal{Q} \mid \mathcal{R}) \qquad \text{(P-ASC)}$$

$$\frac{\mathcal{P}_1 \cong \mathcal{P}_2,\ \mathcal{Q}_1 \cong \mathcal{Q}_2}{\mathcal{P}_1 \mid \mathcal{Q}_1 \cong \mathcal{P}_2 \mid \mathcal{Q}_2} \qquad \text{(P-TRN)}$$

$$\frac{c \notin \mathit{Channel}(\mathcal{C}[\emptyset])}{\nu(c).\mathcal{C}[\mathcal{P}] \cong \mathcal{C}[\nu(c).\mathcal{P}]} \qquad \text{(EXT)}$$

$$\frac{\mathcal{P} \text{ has no public channels}}{\mathcal{P} \cong \emptyset} \qquad \text{(ZER)}$$

$$\frac{\begin{array}{c}\mathcal{C}[\mathit{out}(c,\mathrm{T})] \text{ and } \mathcal{C}[\mathit{out}(c,\mathrm{U})] \text{ are scheduler-insensitive,}\\ [3pt] c \notin \mathit{Channel}(\mathcal{C}[\emptyset]),\ \mathit{PubChan}(\mathcal{C}[\mathit{out}(c,\mathrm{T})]) = \{c\},\\ [3pt] \mathcal{C}[\mathit{out}(c,\mathrm{T})] \cong \mathcal{C}[\mathit{out}(c,\mathrm{U})]\end{array}}{\exists \mathrm{T}_{\mathcal{C}[\,\cdot\,]}, \mathrm{U}_{\mathcal{C}[\,\cdot\,]} \in \textbf{Term}\colon \mathit{out}(c, \mathrm{T}_{\mathcal{C}[\,\cdot\,]}) \cong \mathit{out}(c, \mathrm{U}_{\mathcal{C}[\,\cdot\,]})} \qquad \text{(NU)}$$

$$\frac{\mathit{width}(c) = \mathit{width}(d),\ d \notin \mathit{Channel}(\mathcal{P}, \mathcal{Q})}{\nu(c).\mathcal{P} \cong \nu(d).\mathcal{P}^{[d/c]}} \qquad \text{(NREN)}$$

$$\frac{\begin{array}{c}\mathit{width}(c) = \mathit{width}(d),\ d \notin \mathit{Channel}(\mathcal{P}, \mathcal{Q}),\\ [3pt] c \in \textbf{Unbindable} \iff d \in \textbf{Unbindable},\ \mathcal{P} \cong \mathcal{Q}\end{array}}{\mathcal{P}^{[d/c]}, \cong \mathcal{Q}^{[d/c]}} \qquad \text{(PREN)}$$

$$\frac{f_{\mathrm{T}} \text{ and } f_{\mathrm{U}} \text{ are computationally indistinguishable}}{\mathit{out}(c,\mathrm{T}) \cong \mathit{out}(c,\mathrm{U})} \qquad \text{(EQ1)}$$

$$\frac{\forall i \in [1..k]\colon \mathit{out}(c,\mathrm{T_i}) \cong \mathit{out}(c,\mathrm{U_i})}{\mathit{out}(d, \mathrm{V}(\mathrm{T}_1,\ldots,\mathrm{T_k})) \cong \mathit{out}(d, \mathrm{V}(\mathrm{U}_1,\ldots,\mathrm{U_k}))} \qquad \text{(EQ2)}$$

$$\frac{\begin{array}{c}\forall a_1,\ldots,a_k\colon \mathit{out}(c_i, \mathrm{U_i}(\mathrm{a}_1,\ldots,\mathrm{a_k})) \cong \mathit{out}(c_i, \mathrm{V_i}(\mathrm{a}_1,\ldots,\mathrm{a_k})),\ i \in [1..m],\\ \mathit{FreeVar}(\mathcal{C}[\mathit{out}(c_1, \mathrm{U}_1(\mathrm{x}_1,\ldots,\mathrm{x_k})),\ldots,\mathit{out}(c_m, \mathrm{U_m}(\mathrm{x}_1,\ldots,\mathrm{x_k}))]) =\\ \mathit{FreeVar}(\mathcal{C}[\mathit{out}(c_1, \mathrm{V}_1(\mathrm{x}_1,\ldots,\mathrm{x_k})),\ldots,\mathit{out}(c_m, \mathrm{U_m}(\mathrm{x}_1,\ldots,\mathrm{x_k}))]) = \{x_i\}\end{array}}{\begin{array}{c}\mathit{in}(d,x_i).\mathcal{C}[\mathit{out}(c_1, \mathrm{U}_1(\mathrm{x}_1,\ldots,\mathrm{x_k})),\ldots,\mathit{out}(c_m, \mathrm{U_m}(\mathrm{x}_1,\ldots,\mathrm{x_k}))]\\ \cong \mathit{in}(d,x_i).\mathcal{C}[\mathit{out}(c_1, \mathrm{V}_1(\mathrm{x}_1,\ldots,\mathrm{x_k})),\ldots,\mathit{out}(c_m, \mathrm{V_m}(\mathrm{x}_1,\ldots,\mathrm{x_k}))]\end{array}} \qquad \text{(PUL)}$$

Fig. 3. A reasoning system for PPC.

$$\frac{\mathcal{P} \cong \mathcal{Q},\ \mathcal{C}[\,\cdot\,] \in \mathbf{CExpr}}{\mathcal{C}[\mathcal{P}] \cong \mathcal{C}[\mathcal{Q}]}, \tag{CON}$$

$$\mathcal{P} \cong \mathcal{P}, \tag{RFLX}$$

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

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

The four rules (P-SYM), (ZERO), (P-ASC), and (P-TRN) formalise various properties of the parallel composition operator. The three rules (P-SYM), (ZERO), and (P-ASC) are demonstrated via bisimulations. It is easy to see that the LTSs of the left-hand side expression and the right-hand side expression in each of these rules are isomorphic. Once we have the witnessing bisimulations, we apply Theorem 33. As a consequence of (P-ASC) we will write $\mathcal{P} \mid \mathcal{Q} \mid R$ for either $(\mathcal{P} \mid \mathcal{Q}) \mid R$ or $\mathcal{P} \mid (\mathcal{Q} \mid R)$. As for rule (P-TRN), we argue as follows. Using (CON) and the context $\mathcal{P}_1 \mid [\,\cdot\,]$, we have $\mathcal{Q}_1 \cong \mathcal{Q}_2$ implies $\mathcal{P}_1 \mid \mathcal{Q}_1 \cong \mathcal{P}_1 \mid \mathcal{Q}_2$. But, $\mathcal{P}_1 \cong \mathcal{P}_2$ implies $\mathcal{P}_1 \mid \mathcal{Q}_2 \cong \mathcal{P}_2 \mid \mathcal{Q}_2$ (using (CON) with the context $[\,\cdot\,] \mid \mathcal{Q}_2$). We use the transitivity of $\cong$ to complete the proof.

$$\mathcal{P} \mid \mathcal{Q} \cong \mathcal{Q} \mid \mathcal{P}, \tag{P-SYM}$$

$$\emptyset \mid \mathcal{P} \cong \mathcal{P}, \tag{ZERO}$$

$$(\mathcal{P} \mid \mathcal{Q}) \mid \mathcal{R} \cong \mathcal{P} \mid (\mathcal{Q} \mid \mathcal{R}), \tag{P-ASC}$$

$$\frac{\mathcal{P}_1 \cong \mathcal{P}_2,\ \mathcal{Q}_1 \cong \mathcal{Q}_2}{\mathcal{P}_1 \mid \mathcal{Q}_1 \cong \mathcal{P}_2 \mid \mathcal{Q}_2}. \tag{P-TRN}$$

The rules (EXT), (ZER), and (NU) allow us to manipulate private channels in various ways. In particular, they allow us to remove or add private channels to an expression under certain conditions. Rule (EXT) allows us to shrink or expand the scope of a private channel so long as it does not make formerly public channels private. The proof relies on the fact that if c is not a channel that appears in $\mathcal{C}[\,\cdot\,]$, then the LTS of $v(c).\mathcal{C}[\mathcal{P}]$ is isomorphic to the LTS of $\mathcal{C}[v(c).\mathcal{P}]$, i.e., $\{\langle v(c).\mathcal{D}[\mathcal{Q}], \mathcal{D}[v(c).\mathcal{Q}]\rangle \mid v(c).\mathcal{D}[\mathcal{Q}] \in \mathbb{L}(v(c).\mathcal{C}[\mathcal{P}])\}$ is a bisimulation. In particular, $v(c).\mathcal{C}[\mathcal{P}]$ takes an α-transition to $v(c).\mathcal{D}[\mathcal{Q}]$ iff $\mathcal{C}[v(c).\mathcal{P}]$ takes an α-transition to $\mathcal{D}[v(c).\mathcal{Q}]$ with the same probability. This follows since the absence of the channel c in $\mathcal{C}[\,\cdot\,]$ means that restricting the scope of the binding v to just $\mathcal{P}$ does not make formerly silent actions on the channel c public, nor does it make formerly impossible partial actions on the channel c possible. Since v does not affect probabilities of actions (except partial ones on the bound channel name) or priorities of actions, clearly, the two LTSs have to be isomorphic. The rule (ZER) follows from the fact that an expression that produces no observables is trivially observationally equivalent to the zero-expression $\emptyset$ since the cumulative mass of any α-path (with α a public action) to any set of processes is zero.

$$\frac{c \notin \mathit{Channel}(\mathcal{C}[\emptyset])}{v(c).\mathcal{C}[\mathcal{P}] \cong \mathcal{C}[v(c).\mathcal{P}]}, \tag{EXT}$$

$$\frac{\mathcal{P} \text{ has no public channels}}{\mathcal{P} \cong \emptyset}, \tag{ZER}$$

$$\frac{\begin{array}{c}\mathcal{C}[\mathit{out}(c, \mathrm{T})] \text{ and } \mathcal{C}[\mathit{out}(c, \mathrm{U})] \text{ are scheduler-insensitive,}\\ c \notin \mathit{Channel}(\mathcal{C}[\emptyset]),\ \mathit{PubChan}(\mathcal{C}[\mathit{out}(c, \mathrm{T})]) = \{c\},\\ \mathcal{C}[\mathit{out}(c, \mathrm{T})] \cong \mathcal{C}[\mathit{out}(c, \mathrm{U})]\end{array}}{\exists \mathrm{T}_{\mathcal{C}[\,\cdot\,]}, \mathrm{U}_{\mathcal{C}[\,\cdot\,]} \in \mathbf{Term}\colon \mathit{out}(c, \mathrm{T}_{\mathcal{C}[\,\cdot\,]}) \cong \mathit{out}(c, \mathrm{U}_{\mathcal{C}[\,\cdot\,]})}. \tag{NU}$$

Rule (NU) states that if you have a scheduler-insensitive expression (i.e., an expression where only one action is possible at any stage in the evaluation of the expression thereby rendering the choice of scheduler superfluous) with only one output on a public channel, then the entire expression can be written as a single term placed in an output on the same channel. That is to say, we can fold all the private behaviour of the expression into the single public output by simulating this behaviour and generating the appropriate public message in the output term itself. That we can do this relies on the fact that every expression can be evaluated mechanically in polynomial time (Theorem 28) and that the choice of scheduler is irrelevant (which implies the existence of a single term that correctly simulates the evaluation of the expression regardless of the choice of scheduler). Essentially, this rule states the silent transitions are completely invisible. The proof is fairly straightforward. Consider the term $\mathrm{T}_{\mathcal{C}[\,\cdot\,],\mathrm{S}}$ that simulates the evaluation of $\mathcal{C}[out(c,\mathrm{T})]$ under the scheduler S and outputs a iff the observable (c, a) is generated. We know such a term exists since there exists a probabilistic poly-time Turing machine that evaluates the expression $\mathcal{C}[out(c,\mathrm{T})]$ under the scheduler S (see Theorem 28). Since $\mathcal{C}[out(c,\mathrm{T})]$ is scheduler-insensitive, the behaviour of the process does not depend on the scheduler. Thus, each $\mathrm{T}_{\mathcal{C}[\,\cdot\,],\mathrm{S}}$ (parameterised by the choice of scheduler) is in fact the same term (since the scheduler does not matter). So in fact we have a single term $\mathrm{T}_{\mathcal{C}[\,\cdot\,]}$ that simulates the evaluation of $\mathcal{C}[out(c,\mathrm{T})]$ (under any scheduler) and returns a iff the observable (c, a) is generated. Whence we have demonstrated the existence of the term $\mathrm{T}_{\mathcal{C}[\,\cdot\,]}$. Similarly, we construct $\mathrm{U}_{\mathcal{C}[\,\cdot\,]}$. To finish, we can easily verify that $out(c, \mathrm{T}_{\mathcal{C}[\,\cdot\,]}) \cong out(c, \mathrm{U}_{\mathcal{C}[\,\cdot\,]})$ via the obvious bisimulation.

$$\frac{width(c) = width(d),\ d \notin Channel(\mathcal{P}, \mathcal{Q})}{\nu(c).\mathcal{P} \cong \nu(d).\mathcal{P}^{[d/c]}}, \tag{NREN}$$

$$\frac{\begin{array}{c} width(c) = width(d),\ d \notin Channel(\mathcal{P}, \mathcal{Q}), \\ c \in \mathbf{Unbindable} \iff d \in \mathbf{Unbindable},\ \mathcal{P} \cong \mathcal{Q} \end{array}}{\mathcal{P}^{[d/c]} \cong \mathcal{Q}^{[d/c]}}. \tag{PREN}$$

The first of the two rules dealing with renaming channels, (NREN), states that one can rename a private channel to an unused private channel (as long as bandwidths are respected). In this rule, $\mathcal{P}^{[d/c]}$ denotes the expression obtained by replacing the bindable channel name c with the unused bindable channel name d (we define a similar notation for processes). It is easy to give a bisimulation between $\nu(c)P$ and $\nu(d)P^{[d/c]}$ (where P is a process in the expression $\mathcal{P}$) since the two LTSs are isomorphic and produce exactly the same observables. The second rule regarding renaming, (PREN), allows us to rename public channels to a name that is not currently in use by the expression. Naturally, we need to respect bandwidths, but, in addition, we must ensure that we do not change the priority of actions on a channel by replacing a low priority channel name with a high priority channel name or vice versa. 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 equivalence (all it does is map observables on the channel c to observables on the hitherto unused 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)—renaming the channels amounts to giving the isomorphism between the LTS of $\mathcal{P}$ (resp. $\mathcal{Q}$) and $\mathcal{P}^{[d/c]}$ (resp. $\mathcal{Q}^{[d/c]}$).

The final three proof rules—(EQ1), (EQ2), and (PUL)—allow us to manipulate computationally indistinguishable terms. The rule (EQ1) states that if the functions f_T and f_U are computationally indistinguishable then $out(c, \mathrm{T})$ and $out(c, \mathrm{U})$ are observationally indistinguishable. This rule is just the formalisation, as a proof rule, of Theorem 40. Since expression of the cryptographically important notion of computational indistinguishability is an important step in our program, rather than giving a sketch of the proof here, we simply state the result here and point the reader to the full proof given in Section 7.1. Essentially, the argument shows that if $out(c, \mathrm{T})$ can be distinguished from $out(C, \mathrm{U})$, then the context expression witnessing that distinction can be converted into a probabilistic poly-time algorithm that distinguishes between f_T and f_U by viewing the context as a function that takes as input the output of the term plugged into the hole and constructing a machine that runs the context on the input.

$$\frac{f_\mathrm{T} \text{ and } f_\mathrm{U} \text{ are computationally indistinguishable}}{out(c,\mathrm{T}) \cong out(c,\mathrm{U})}, \tag{EQ1}$$

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

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

The rule (EQ2) states that if there are k pairs of terms $\langle \mathrm{T_i}, \mathrm{U_i}\rangle$ that are pairwise observationally equivalent, then the two expressions $out(d, \mathrm{V}(\mathrm{T_1},\ldots,\mathrm{T_k}))$ and $out(d, \mathrm{V}(\mathrm{U_1},\ldots,\mathrm{U_k}))$ are observationally equivalent. The proof proceeds by contradiction. For each $1 \leqslant j \leqslant k$, we consider two expressions. The first, $\mathcal{H}_j$ is defined as $out(d, \mathrm{V}(\mathrm{T_1}, \ldots, \mathrm{T_{j-1}}, \mathrm{U_j}, \ldots, \mathrm{U_k}))$ and the second, $\mathcal{H}_{j+1}$, is defined as $out(d, \mathrm{V}(\mathrm{T_1}, \ldots, \mathrm{Tj}, \mathrm{U_{j+1}}, \ldots, \mathrm{U_k}))$. We note that $\mathcal{H}_j$ and $\mathcal{H}_{j+1}$ differ only in the jth argument to V. Assume that the context $\mathcal{C}[\,\cdot\,]$ witnesses that $\mathcal{H}_j \not\cong \mathcal{H}_{j+1}$. Then the context defined as $in(c, x).\mathcal{C}[out(d, \mathrm{V}(\mathrm{T_1}, \ldots, \mathrm{T_{j-1}}, \mathrm{x}, \mathrm{U_{j+1}}, \ldots, \mathrm{U_k}))]$ distinguishes between $out(c, \mathrm{T_j})$ and $out(c, \mathrm{U_j})$. This contradicts the hypothesis that each $out(c, \mathrm{T_j})$ is observationally equivalent to $out(c, \mathrm{U_j})$. 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. Finally, the rule (PUL) allows us to replace an argument to a term with a free variable and introduce a binding input under certain conditions. The proof sketch is similar to that employed for (EQ2). As in that case, we proceed by contradiction. For each $1 \leqslant j \leqslant m$, we consider two processes. The first, $\mathcal{H}_j$, is defined as

$$in(d, x_i).\mathcal{C}[out(c_1, \mathrm{V_1}(\mathrm{x_1}, \ldots, \mathrm{x_k})), \ldots, out(c_{j-1}, \mathrm{V_{j-1}}(\mathrm{x_1}, \ldots, \mathrm{x_k})),$$
$$out(c_j, \mathrm{U_j}(\mathrm{x_1}, \ldots, \mathrm{x_k})), \ldots, out(c_m, \mathrm{U_m}(\mathrm{x_1}, \ldots, \mathrm{x_k}))]]$$

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

$$in(d, x_i).\mathcal{C}[out(c_1, \mathrm{V_1}(\mathrm{x_1}, \ldots, \mathrm{x_k})), \ldots, out(c_j, \mathrm{V_j}(\mathrm{x_1}, \ldots, \mathrm{x_k})),$$
$$out(c_{j+1}, \mathrm{U_{j+1}}(\mathrm{x_1}, \ldots, \mathrm{x_k})), \ldots, out(c_m, \mathrm{U_m}(\mathrm{x_1}, \ldots, \mathrm{x_k}))].$$

Note that $\mathcal{H}_j$ and $\mathcal{H}_{j+1}$ only differ in the expression 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 $\mathrm{U_j}$ and $\mathrm{V_j}$ at those values can be distinguished with non-negligible advantage. But this means that $out(c_1, \mathrm{U_j}(\mathrm{a_1}, \ldots, \mathrm{a_k})) \not\cong out(c_1, \mathrm{V_j}(\mathrm{a_1}, \ldots, \mathrm{a_k}))$ 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. We could have presented an alternate version of (PUL) where the ith pair of terms $\langle \mathrm{U}_i, \mathrm{V}_i \rangle$ has k_i arguments rather than all the terms having the same number of arguments. A similar version of the reasoning just given would have worked since it does not depend on U_i and U_j $(i \neq j)$ having the same number of arguments, only on U_i and V_i having the same number of arguments. We chose the version given for its simplicity. Furthermore, the omission does not cost anything since, by introducing unused parameters, we can make any k-ary function an $(k + k')$-ary function. Then, using (EQ1) in conjunction with (PUL), we would be able to deploy the precisely the kind of reasoning made available by the omitted version of (PUL).

7. Cryptographic examples

In what follows we will denote an element x chosen uniformly at random from the set X by $x \in_{\mathrm{R}} X$. In Section 7.1 we will show that our notion of asymptotic observational equivalence and the standard notion of indistinguishability by poly-time statistical tests coincide. In Section 7.2 we apply the previous observation to define pseudorandom number generators in PPC. In Sections 7.3 and 7.4 we, respectively, define semantic security and the Decision Diffie–Hellman assumption (DDHA) in terms of equivalences between PPC expressions. Finally, in Section 7.5 we derive the equivalence between the DDHA and the semantic security of El Gamal encryption by making use of the formal proof system for PPC given in Section 6.

7.1. 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 the cryptographic literature [67,44,30,29,31,47].

Definition 34 (*Function ensemble*). A *function ensemble f* is an indexed family of functions $\{f_i : A_i \to B_i\}_{i \in \mathbb{N}}$. A function ensemble $f : A_i \to B_i$ is *uniform* if there exists a single Turing machine M that computes f for all values of i i.e., $\forall i \in \mathbb{N}.\forall a \in A_i : M(i, a) = f_i(a)$. A uniform function ensemble $f : A_i \to B_i$ is *poly-time* if there exists a polynomial q and a single Turing machine M such that $M(i, a)$ computes $f_i(a)$ in time at most $q(|i|, |a|)$. A uniform function ensemble $f : A_i \to B_i$ is *probabilistic poly-time* if f_i is a probabilistic poly-time function. A *poly-time statistical test* $\mathcal{A}$: $\{0, 1\}^{m(x)} \to \{0, 1\}$ 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 35 (*Computational indistinguishability*). Let $q(x)$ be a positive polynomial. A uniform probabilistic poly-time function ensemble $f : \emptyset \to \{0,1\}^{l(x)}$ is *computationally indistinguishable* from a uniform probabilistic poly-time function ensemble $g : \emptyset \to \{0,1\}^{l(x)}$ just when for all poly-time statistical tests $\mathcal{A}$ we have

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

Definition 36. Let $\mathcal{P}$ be a variable-closed expression with no public inputs and with exactly one public output on the channel c where $width(c) = q(x)$. We will say that the probabilistic poly-time function ensemble $f^{\mathcal{P}} : \emptyset \to \{0,1\}^{q(x)}$ is the *characteristic function for $\mathcal{P}$ with respect to the scheduler* S when we have that $\forall a \in \mathbb{N}\colon \mathrm{Prob}\left[f_i^{\mathcal{P}}() = a\right] = \mathrm{Prob}\left[[i/\eta]\mathcal{P} \rightsquigarrow_{\mathsf{S}} (c, a)\right]$. Let $f : \emptyset \to \{0,1\}^{q(x)}$ be a probabilistic poly-time function ensemble. Let T_f be a term such that M_{T_f} computes f. Then, we say that $out(c, \mathrm{T}_f)$ is the *characteristic expression for f.*

Let $f : \emptyset \to \{0,1\}^{q(x)}$ be a probabilistic poly-time function ensemble and let $\mathcal{P}_f \overset{\text{def}}{=} out(c, \mathrm{T}_f)$ be its characteristic expression. Then, it is easy to see that $\forall \mathsf{S} \in \mathbf{PSched}.\forall a \in \mathbb{N}\colon \mathrm{Prob}\left[f_i() = a\right] = \mathrm{Prob}\left[[i/\eta](\mathcal{P}_f \mid in(c, x)) \rightsquigarrow_{\mathsf{S}} (c, a)\right]$.

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 expression $\mathcal{C}[\,\cdot\,]$ that distinguishes between the characteristic expressions for f and g under any scheduler. This is shown in Lemma 37.
(2) If there exists a context expression $\mathcal{C}[\,\cdot\,]$ 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 39.

Lemma 37. *Let $\mathcal{A}\colon \{0,1\}^{m(x)} \to \{0,1\}$ be a poly-time statistical test. Let $\mathcal{P}$ be any variable-closed expression with no public inputs and exactly one public output such that $f^{\mathcal{P}} : \emptyset \to \{0,1\}^{m(x)}$ is its characteristic function. Then, we can construct a context $\mathcal{C}_{\mathcal{A}}[\,\cdot\,]$ 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 we can construct the context expression $\mathcal{C}_{\mathcal{A}}[\,\cdot\,] \equiv in(c, x).out(d, \mathrm{T}_{\mathcal{A}}(x)) \mid [\,]$ with $width(c) = m(x)$ and $width(d) = 1$.

It is easy to see that this context expression applies the test to the $m(\eta)$-bit output of some expression ‘plugged’ into the hole. By assumption, we have that $f^{\mathcal{P}}$ is the characteristic function for $\mathcal{P}$. Now, $\mathrm{T}_{\mathcal{A}}$ is the term that computes the probabilistic poly-time function $\mathcal{A}$. Hence, $\mathcal{C}_{\mathcal{A}}[\mathcal{P}]$ must produce the observables $(d, 0)$ and $(d, 1)$. The probability that $[i/\eta]\mathcal{C}_{\mathcal{A}}[\mathcal{P}]$ 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 $[i/\eta]\mathcal{C}_{\mathcal{A}}[\mathcal{P}]$ has only one possible communication (the one between $[i/\eta]\mathcal{P}$ and $[i/\eta]\mathcal{C}_{\mathcal{A}}[\,\cdot\,]$).

Similarly, the probability that $[i/\eta]\mathcal{C}_{\mathcal{A}}[\mathcal{P}]$ produces the observable $(d, 1)$ 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 a context so constructed is a *poly-time distinguishing context.*

Definition 38. Let $\mathcal{P}$ be a variable-closed expression and o an observable. We will say that $f : \emptyset \to \{0,1\}$ is an *indicator for $\mathcal{P}$ with respect to o under the scheduler* S when

$$\mathrm{Prob}\left[[i/\eta]\mathcal{P} \rightsquigarrow_{\mathsf{S}} o\right] = \mathrm{Prob}\left[f_i() = 1\right]$$

and

$$\text{Prob}\left[[i/\eta]\mathcal{P} \not\rightsquigarrow_{\mathsf{S}} o\right] = \text{Prob}\left[f_i() = 0\right].$$

Lemma 39. *Let $\mathcal{C}[\,\cdot\,]$ be a context and let o be an observable. Let $f : \emptyset \to \{0,1\}^{m(x)}$ be any function and $\mathcal{P}_f$ be its characteristic variable-closed expression. Then, we can specify a poly-time statistical test t from the triple $\langle \mathcal{C}[\,\cdot\,], o, \mathsf{S}\rangle$ 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 $\mathcal{C}[out(d, \mathrm{T}_f)]$ (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

$$\text{Prob}\left[f \circ t = 1\right] = \text{Prob}\left[[i/\eta]\mathcal{C}[out(d, \mathrm{T}_f)] \rightsquigarrow_{\mathsf{S}} o\right]$$

and that

$$\text{Prob}\left[f \circ t = 0\right] = \text{Prob}\left[[i/\eta]\mathcal{C}[out(d, \mathrm{T}_f)] \not\rightsquigarrow_{\mathsf{S}} o\right]$$

whence the desired result is obtained. □

Clearly, given a context and a variable-closed expression, each potential observable defines a poly-time statistical test.

Theorem 40. *Let $f : \emptyset \to \{0,1\}^{l(x)}$ be a uniform probabilistic poly-time function ensemble. Let $g : \emptyset \to \{0,1\}^{l(x)}$ 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 37, we can construct a poly-time distinguishing context $\mathcal{C}[\,\cdot\,]$ that, under any scheduler, distinguishes between $\mathcal{F}$ and $\mathcal{G}$ with the same probability that $\mathcal{A}$ distinguishes between *f* and g. So $\mathcal{C}[\,\cdot\,]$ 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, $\mathcal{F} \ncong \mathcal{G}$. Then we have that for some polynomial $p(x)$ and scheduler S there exists a context $\mathcal{D}[\,\cdot\,]$ that distinguishes between the two processes with probability greater than $1/p(i)$. Let the distinguishing observation be (d, a). We can then use Lemma 39 to construct a poly-time statistical test that distinguishes between the output of f and g with precisely the same probability that $\mathcal{D}[\,\cdot\,]$ distinguishes between $\mathcal{F}$ and $\mathcal{G}$ under the scheduler S, thereby creating a contradiction. □

7.2. Pseudorandom number generators

We can now prove that an algorithm taking short strings to long strings is pseudorandom if and only if the characteristic expression for the algorithm, when evaluated on a short random input, is observationally equivalent to the expression that returns a long random string.

Definition 41. A function ensemble $f : \emptyset \to \{0,1\}^{l(x)}$ is *random poly-time* if *f* 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 [67,44,30,29,31,47].

Definition 42 (*Pseudorandom number generator*). Let $q(x)$ be a positive polynomial. A *pseudorandom number generator* (PRNG) is a uniform poly-time function ensemble $f : \{0,1\}^{k(x)} \to \{0,1\}^{l(x)}$ such that for all poly-time statistical tests $\mathcal{A}$

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

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

Theorem 43. *Let $f'\colon \{0,1\}^{k(x)} \to \{0,1\}^{l(x)}$ $(\forall a \in \mathbb{N}\colon l(a) > k(a))$ 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 variable-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 variable-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 Theorem 40. 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. □

7.3. 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 [33,10]. 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 44 (*Diffie and Hellman [24], Goldreich [32], Goldwasser and Bellare [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 which 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 which 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 of semantic security is in a chosen-plaintext model of security since the adversary, being in possession of the public key, can encrypt a polynomial number of plaintexts that are chosen by the adversary, before it attempts to compute something about the plaintext associated with the given ciphertext.

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 et al. [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 45. 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 k,

$$\left|\text{Prob}\left[A(1^k, e, F(1^k, e), c) = m | c \in E(e, m_0)\right] - \text{Prob}\left[A(1^k, e, F(1^k, e), c) = m | c \in E(e, m_1)\right]\right| \leqslant \frac{1}{q(k)},$$

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 so, we will assume an efficient tupling function i.e., a function $\langle x_1, \ldots, x_k\rangle$ 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. Normal 'diagonal' pairing or a scheme based on bit-interleaving will do nicely. In what follows, we will use the notation $in(c, \langle x_1, \ldots, x_k\rangle)$ 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.

Definition 46. Let $\langle G, E, D\rangle$ be an encryption scheme. Then $\langle G, E, D\rangle$ is an *observationally indistinguishable encryption scheme* iff

$$\begin{aligned}&\nu(internal).out(internal, \text{pkey}(G(1^\eta))) \mid in(internal, key).out(pub, \langle key, 1^\eta\rangle).\\&\quad in(msg, \langle m_0, m_1\rangle).out(challenge, \langle \text{key}, \langle \text{m}_0, \text{m}_1\rangle, \text{E}(\text{key}, \text{m}_0)\rangle)\end{aligned} \tag{$\mathcal{LSS}$}$$

is observationally indistinguishable from

$$\begin{aligned}&\nu(internal).out(internal, \text{pkey}(G(1^\eta))) \mid in(internal, key).out(pub, \langle key, 1^\eta\rangle).\\&\quad in(msg, \langle m_0, m_1\rangle).out(challenge, \langle \text{key}, \langle \text{m}_0, \text{m}_1\rangle, \text{E}(\text{key}, \text{m}_1)\rangle),\end{aligned} \tag{$\mathcal{RSS}$}$$

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

We note that all channel names are bindable (and, consequently high priority), while only *internal* has been made private. Making the channel names bindable allows us to make certain channels private, an ability that will be useful in proving the equivalence of the semantic security of El Gamal encryption and the Decision-Diffie–Hellman assumption. We remark that making all the channels high-priority does not affect the order in which communication steps of either $\mathcal{LSS}$ or $\mathcal{RSS}$ occur since at each stage there is only one possible communicationstep that can happen. An examination of the expression $\mathcal{LSS}$ 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[5] and the public key. The message-generation algorithm F can adaptively choose the message-pair based on the public key and security parameter.
(4) Publishes the encryption of the first message, along with the message pair and the encryption key. 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.

[5] 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.

An examination of the expression $\mathcal{RSS}$ shows that it is identical to expression $\mathcal{LSS}$ except that $\mathcal{RSS}$ chooses to encrypt the second message. It should be 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 formalise this intuition.

Theorem 47. *Let $\langle G, E, D\rangle$ be an encryption scheme. Then, $\langle G, E, D\rangle$ is semantically secure iff $(\mathcal{LSS}) \cong (\mathcal{RSS})$.*

Proof. Let us assume that $\langle G, E, D\rangle$ is not indistinguishably secure and then show that $\mathcal{LSS} \cong \mathcal{RSS}$ 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 probability, A can distinguish between encryptions of messages chosen by F, i.e., for some positive polynomial q we have

$$\left|\text{Prob}\left[A(1^k, e, F(1^k, e), c) = m \mid c \in E(e, m_0)\right] - \text{Prob}\left[A(1^k, e, F(1^k, e), c') = m \mid c' \in E(e, m_1)\right]\right| > \frac{1}{q(k)}$$

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

$$\begin{array}{l} [\cdot] \mid in(pub, \langle key, sec\rangle). \\ \quad out(msg, \text{F(sec, key)}).in(challenge, \langle pubkey, msgpair, cipher\rangle). \\ \qquad out(response, \text{A(sec, pubkey, msgpair, cipher)}) \\ \qquad\quad \mid in(response, x).\oslash \end{array}$$

Let us consider $\mathcal{A}[\mathcal{LSS}]$ (resp. $\mathcal{A}[\mathcal{RSS}]$). The first thing to note is that in any evaluation path of $\mathcal{A}[\mathcal{LSS}]$ (resp. $\mathcal{A}[\mathcal{RSS}]$), 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 $\mathcal{A}[\cdot]$ supplies a pair of messages chosen by F (based on the security parameter and public key) to $\mathcal{LSS}$ (resp. $\mathcal{RSS}$) 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 $\mathcal{A}[\cdot]$ applies A to the tuple consisting of the security parameter, message-pair, encryption key, and ciphertext. This yields a guess as to the message that was encrypted. Since $\mathcal{A}[\cdot]$ 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 $\mathcal{A}[\cdot]$ can distinguish between $\mathcal{LSS}$ and $\mathcal{RSS}$ with non-negligible probability (i.e., with probability greater than $1/q(\eta)$ which is the probability with which A distinguishes between encryptions of messages given by F). Thus, $\mathcal{LSS} \ncong \mathcal{RSS}$ since we have constructed a distinguishing context $\mathcal{A}[\cdot]$ out of A and F.

Let us now tackle the reverse direction. We assume that $\mathcal{LSS} \ncong \mathcal{RSS}$ and show that $\langle G, E, D\rangle$ is not indistinguishably secure. Since $\mathcal{LSS} \ncong \mathcal{RSS}$, we have a context expression $\mathcal{A}[\cdot]$ that, under some perceptible scheduler S, distinguishes between $\mathcal{LSS}$ and $\mathcal{RSS}$ on the basis of some observable o. Furthermore, we assume $\mathcal{A}[\cdot]$ must provide messages to $\mathcal{LSS}$ (resp. $\mathcal{RSS}$) in order for the protocol to run (since $\mathcal{LSS}$ and $\mathcal{RSS}$ differ only in the challenge step, any distinguishing observable can only be generated after the challenge step has occurred). In particular, no context expression can distinguish between two expressions if the context expression does not interact with the two expressions. Thus there must exist a probabilistic poly-time algorithm F such that the probability that $\mathcal{A}[\cdot]$, using security parameter k and public key e, provides the message pair π to $\mathcal{LSS}$ (resp. $\mathcal{RSS}$) 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 $\mathcal{P}\langle e, \pi, c\rangle$ and then evaluating $\mathcal{A}[\mathcal{P}\langle e, \pi, c\rangle]$ under the scheduler S and with security parameter η 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 enc_i. The expression $\mathcal{P}\langle e, \pi, enc_i\rangle$ is defined as

$$\begin{array}{l} \nu(internal).out(internal, pkey(G(1^\eta))) \mid in(internal, key).out(pub, \langle key, 1^\eta\rangle). \\ \quad in(msg, \langle m_0, m_1\rangle).out(challenge, \langle e, \pi, enc_i\rangle) \end{array}$$

Clearly, the function A will successfully distinguish between encryptions of messages in π with the same probability that, using the scheduler S, $\mathcal{A}[\cdot]$, distinguishes between $\mathcal{P}\langle e, \pi, enc_0\rangle$ and $\mathcal{P}\langle e, \pi, enc_1\rangle$. Hence $\langle G, E, D\rangle$ is not indistinguishably secure. Since $\langle G, E, D\rangle$ is indistinguishably secure iff $\mathcal{LSS} \cong \mathcal{RSS}$, we conclude that $\langle G, E, D\rangle$ is semantically secure iff $\mathcal{LSS} \cong \mathcal{RSS}$. $\square$

7.4. The decision Diffie–Hellman assumption

We start by defining the Decision Diffie–Hellman assumption [24]. Our presentation is drawn from Boneh [12] and Tsiounis and Yung [64]. Goldreich [32] offers helpful discussions, as does Cramer and Shoup [20].

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 48. *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 polynomial $q(x)$ such that $IG(\eta) = \langle p, g\rangle$ implies that $|\langle p, g\rangle| = \Omega(q(\eta))$.

The probability is taken over the probability that the instance generator $IG(1^\eta)$ returns $\langle p, g\rangle$ given η, over the random choice of a, b, c in $[1..\mathrm{ord}\, G_p]$, and over the 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 $r(x)$ such that $|\langle p, g\rangle| = O(r(\eta))$.

Example 49. 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 parameterised by the choice of prime p.
(2) Let $N = pq$ where $p, q, (p-1)/2, (q-1)/2$ 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 parameterised 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 parameterised 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 parameterised 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 that 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 $[1..\mathrm{ord}\, G_p]$, g^a is the a-fold application of the group operation, and $=$ is group identity.

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

1. $\langle p, g\rangle = IG(1^\eta)$,
2. return $\langle p, g, g^a, g^b, g^c\rangle$,

where η 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 50. The group family $\mathbb{G}$ is *observationally DDHA-secure* if

$$out(challenge, \langle \mathrm{p, g, g^a, g^b, g^{ab}}\rangle | \mathrm{a, b} \in_{\mathrm{R}} [1..\mathrm{ord\, G_p}]) \cong out(challenge, \langle \mathrm{p, g, g^a, g^b, g^c}\rangle | \mathrm{a, b, c} \in_{\mathrm{R}} [1..\mathrm{ord\, G_p}]),$$

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

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

Proof. A special case of Theorem 40. □

7.5. The semantic security of El Gamal encryption

We now proceed to give an example of the use of PPC to establish properties of protocols. In particular, we will show that the semantic security of El Gamal encryption is equivalent to the Decision Diffie–Hellman assumption. We start by describing the El Gamal encryption scheme [28].

Definition 52. Let $\cdot$ denote the group operation and $=$ denote group equality. An *El Gamal 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 $\langle g, p\rangle \in IG(1^k)$, $a \in_R [1..\text{ord}\, G_p]$.
(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 $b \in_R [1..\text{ord}\, G_p]$).
(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 $k = g^b$, $c' = m \cdot g^{ab} \bmod p$, and $d = a$ for some a, b, m. Then

$$\frac{c'}{k^d} = \frac{m \cdot g^{ab}}{g^{ba}} = \frac{m \cdot g^{ab}}{g^{ab}} = m.$$

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

$$\begin{aligned}&\nu(internal).out(internal, pkey(G(1^\eta))) \,|\, in(internal, \langle p, g, g^a\rangle).\\ &\quad out(pub, \langle\langle p, g, g^a\rangle, 1^\eta\rangle).in(msg, \langle m_0, m_1\rangle).\\ &\quad out(challenge, \langle\langle p, g, g^a\rangle, \langle m_0, m_1\rangle, \langle g^b, m_0 \cdot g^{ab}\rangle\rangle \,|\, b \in_R [1..\text{ord}\, G_p]) \qquad (\mathcal{LEG})\end{aligned}$$

and

$$\begin{aligned}&\nu(internal).out(internal, pkey(G(1^\eta))) \,|\, in(internal, \langle p, g, g^a\rangle).\\ &\quad out(pub, \langle\langle p, g, g^a\rangle, 1^\eta\rangle).in(msg, \langle m_0, m_1\rangle).\\ &\quad out(challenge, \langle\langle p, g, g^a\rangle, \langle m_0, m_1\rangle, \langle g^b, m_1 \cdot g^{ab}\rangle\rangle \,|\, b \in_R [1..\text{ord}\, G_p]) \qquad (\mathcal{REG})\end{aligned}$$

from the assertion that

$$\begin{aligned}&out(challenge, \langle \mathrm{p, g, g^a, g^b, g^{ab}}\rangle \,|\, \mathrm{a, b} \in_R [1..\text{ord}\, \mathrm{G_p}])\\ &\quad \cong out(challenge, \langle \mathrm{p, g, g^a, g^b, g^c}\rangle \,|\, \mathrm{a, b, c} \in_R [1..\text{ord}\, \mathrm{G_p}]). \qquad (\mathcal{DDHA})\end{aligned}$$

(From Section 7.4 we know that this second assertion is an expression of the DDHA.)

Main Theorem 53. *If the Decision Diffie–Hellman assumption holds for a group family $\mathbb{G}$, then El Gamal 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 Fig.* 3, *the assertion in PPC that El Gamal encryption using* $\mathbb{G}$ *is semantically secure* (*i.e.*, *the equivalence* $\mathcal{LEG} \cong \mathcal{REG}$) *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 $\mathcal{LEG}$ and $\mathcal{REG}$ 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 El Gamal'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 El Gamal'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 El Gamal encryption. This mathematical argument will be formalised 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 indistinguishable 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 $[1..\text{ord}\, G_p]$). We will use $\asymp$ to denote computational indistinguishability. We recall to the reader's attention that computational indistinguishability is transitive. So we have that

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

Since the two tuples are computationally indistinguishable, it follows that

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

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 $[1..\text{ord}\, G_p]$, it follows that $g^{r+ab} = g^{c'}$ for randomly chosen $c' \in [1..\text{ord}\, G_p]$. Thus we get that for all $m_0, m_1 \in [1..\text{ord}\, G_p]$

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

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

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

whence the transitivity of $\asymp$ yields

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

for all message m_0, m_1. This claim is (almost) the assertion that El Gamal 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

$$\begin{aligned} & \mathit{out}(\mathit{challenge}, \langle \mathrm{p}, \mathrm{g}, \mathrm{g^a}, \mathrm{g^b}, \mathrm{g^{ab}}\rangle \mid \mathrm{a}, \mathrm{b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]) \\ & \quad \cong \mathit{out}(\mathit{challenge}, \langle \mathrm{p}, \mathrm{g}, \mathrm{g^a}, \mathrm{g^b}, \mathrm{g^c}\rangle \mid \mathrm{a}, \mathrm{b}, \mathrm{c} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]) \end{aligned}$$

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

$$\begin{aligned} & \mathit{out}(\mathit{challenge}, \langle \mathrm{p}, \mathrm{g}, \mathrm{g^a}, \mathrm{m_0}, \mathrm{g^b}, \mathrm{m_0} \cdot \mathrm{g^{ab}}\rangle \mid \mathrm{a}, \mathrm{b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]) \\ & \quad \cong \mathit{out}(\mathit{challenge}, \langle \mathrm{p}, \mathrm{g}, \mathrm{g^a}, \mathrm{m_0}, \mathrm{g^b}, \mathrm{m_0} \cdot \mathrm{g^c}\rangle \mid \mathrm{a}, \mathrm{b}, \mathrm{c} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]). \end{aligned}$$

Furthermore, since $m_0 = g^d$ 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 $[1..\text{ord}\, G_p]$) both induce the same distribution on elements of the group. Thus

$$\begin{aligned} & \forall\, m_0 \in G_p \colon \mathit{out}(\mathit{challenge}, \langle \mathrm{p}, \mathrm{g}, \mathrm{g^a}, \mathrm{m_0}, \mathrm{g^b}, \mathrm{m_0} \cdot \mathrm{g^c}\rangle \mid \mathrm{a}, \mathrm{b}, \mathrm{c} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]) \\ & \quad \cong \mathit{out}(\mathit{challenge}, \langle \mathrm{p}, \mathrm{g}, \mathrm{g^a}, \mathrm{m_0}, \mathrm{g^b}, \mathrm{g^c}\rangle \mid \mathrm{a}, \mathrm{b}, \mathrm{c} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]). \end{aligned}$$

Then, (TRN) (transitivity) gives us

$$\forall m_0 \in G_p\colon out(challenge, \langle \mathrm{p, g, g^a, m_0, g^b, m_0 \cdot g^{ab}} \rangle \mid \mathrm{a, b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]) \cong out(challenge, \langle \mathrm{p, g, g^a, m_0, g^b, g^c} \rangle \mid \mathrm{a, b, c} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]).$$

Another application of (EQ2) gives us

$$\forall m_0, m_1 \in G_p\colon out(challenge, \langle \mathrm{p, g, g^a, m_0, m_1, g^b, m_0 \cdot g^{ab}} \rangle \mid \mathrm{a, b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]) \cong out(challenge, \langle \mathrm{p, g, g^a, m_0, m_1, g^b, g^c} \rangle \mid \mathrm{a, b, c} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]). \quad (\dagger)$$

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

$$\forall m_0, m_1 \in G_p\colon out(challenge, \langle \mathrm{p, g, g^a, m_0, m_1, g^b, m_1 \cdot g^{ab}} \rangle \mid \mathrm{a, b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]) \cong out(challenge, \langle \mathrm{p, g, g^a, m_0, m_1, g^b, g^c} \rangle \mid \mathrm{a, b, c} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]). \quad (\ddagger)$$

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

$$\forall m_0, m_1 \in G_p\colon out(challenge, \langle \mathrm{p, g, g^a, m_0, m_1, g^b, m_0 \cdot g^{ab}} \rangle \mid \mathrm{a, b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]) \cong out(challenge, \langle \mathrm{p, g, g^a, m_0, m_1, g^b, m_1 \cdot g^{ab}} \rangle \mid \mathrm{a, b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]). \quad (✠)$$

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 formalised 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 El Gamal encryption. In particular, we have an observational equivalence that states the tuple consisting of the elements of an El Gamal 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 El Gamal 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 El Gamal (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 (PREN) to ensure that the channel-names in the derived expression match the channel-names used in our statement of semantic security (see Theorem 47).

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

$$\forall m_0, m_1 \in G_p\colon out(challenge, \langle \langle \mathrm{p, g, g^a} \rangle, \langle \mathrm{m_0, m_1} \rangle, \langle \mathrm{g^b, m_0 \cdot g^{ab}} \rangle \rangle \mid \mathrm{a, b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]) \cong out(challenge, \langle \langle \mathrm{p, g, g^a} \rangle, \langle \mathrm{m_0, m_1} \rangle, \langle \mathrm{g^b, m_1 \cdot g^{ab}} \rangle \rangle \mid \mathrm{a, b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}])$$

Using proof rule (PUL) and (PREN) we can obtain the message-pair from a context

$$in(msg, \langle m_0, m_1 \rangle).\ out(challenge, \langle \langle \mathrm{p, g, g^a} \rangle, \langle \mathrm{m_0, m_1} \rangle, \langle \mathrm{g^b, m_0 \cdot g^{ab}} \rangle \rangle \mid \mathrm{a, b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]) \cong in(msg, \langle m_0, m_1 \rangle).\ out(challenge, \langle \langle \mathrm{p, g, g^a} \rangle, \langle \mathrm{m_0, m_1} \rangle, \langle \mathrm{g^b, m_1 \cdot g^{ab}} \rangle \rangle \mid \mathrm{a, b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]).$$

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

1. $\langle p, g\rangle = IG(1^\eta)$,
2. return $\langle p, g, g^a\rangle$

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

$$\begin{aligned}&out(pub, \langle\langle p, g, g^a\rangle, 1^\eta\rangle).in(msg, \langle m_0, m_1\rangle).\\ &\qquad out(challenge, \langle\langle \mathrm{p, g, g^a}\rangle, \langle \mathrm{m_0, m_1}\rangle, \langle \mathrm{g^b, m_0 \cdot g^{ab}}\rangle\rangle | \mathrm{a, b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}])\\ &\cong\\ &out(pub, \langle\langle p, g, g^a\rangle, 1^\eta\rangle).in(msg, \langle m_0, m_1\rangle).\\ &\qquad out(challenge, \langle\langle \mathrm{p, g, g^a}\rangle, \langle \mathrm{m_0, m_1}\rangle, \langle \mathrm{g^b, m_1 \cdot g^{ab}}\rangle\rangle | \mathrm{a, b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]).\end{aligned}$$

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 p, g, a by making use of the key generator—which will generate the pair consisting of $\langle p, g, g^a\rangle$ (the public key) and 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.

$$\begin{aligned}&\nu(internal).out(internal, pkey(G(1^\eta))) \mid in(internal, \langle p, g, g^a\rangle).\\ &\qquad out(pub, \langle\langle p, g, g^a\rangle, 1^\eta\rangle).in(msg, \langle m_0, m_1\rangle).\\ &\qquad out(challenge, \langle\langle \mathrm{p, g, g^a}\rangle, \langle \mathrm{m_0, m_1}\rangle, \langle \mathrm{g^b, m_0 \cdot g^{ab}}\rangle\rangle \mid \mathrm{b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}])\\ &\cong\\ &\nu(internal).out(internal, pkey(G(1^\eta))) \mid in(internal, \langle p, g, g^a\rangle).\\ &\qquad out(pub, \langle\langle p, g, g^a\rangle, 1^\eta\rangle).in(msg, \langle m_0, m_1\rangle).\\ &\qquad out(challenge, \langle\langle \mathrm{p, g, g^a}\rangle, \langle \mathrm{m_0, m_1}\rangle, \langle \mathrm{g^b, m_1 \cdot g^{ab}}\rangle\rangle \mid \mathrm{b} \in_{\mathrm{R}} [1..\mathrm{ord}\, \mathrm{G_p}]).\end{aligned}$$

But this equivalence is precisely the statement that El Gamal encryption is semantically secure (see Theorem 47). Hence, given the DDHA we can show that El Gamal encryption is semantically secure. □

We draw the reader's attention to the fact that the proof, in PPC, of the semantic security of El Gamal encryption from the Decision Diffie–Hellman assumption 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 challenge to the semantic security of El Gamal encryption. 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, coupled with a few transformations whose soundness is 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 equation $m \cdot g^{ab}$ with g^c). This proof could be rendered entirely mechanical by axiomatising in PPC important mathematical properties about groups that are relevant for this proof and using these rules directly rather than invoking (EQ2). For example, we could formalise the mathematical fact that $\langle g^a\rangle \asymp \langle g^a \cdot m\rangle$ (where g is a group generator, $a \in_{\mathrm{R}} \mathbb{G}_p$, m is some constant in the group, and g^a and $\cdot$ are group operations) as the axiom

$$\forall m \in \mathbb{G}_p\colon out(c, g^a \mid a \in_{\mathrm{R}} \mathbb{G}_p) \cong out(c, g^a \cdot m \mid a \in_{\mathrm{R}} \mathbb{G}_p). \qquad \text{(GRP)}$$

Taken with the structural rules of Fig. 3 this would allow us to derive El Gamal'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 El Gamal encryption's semantic security and reversed the proof's structural transformations to obtain the DDHA—in fact, we will do something analogous to this in proving the next theorem. 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 54. *If El Gamal 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* Fig. 3, *the assertion in PPC that the DDHA (i.e., the equivalence* $\mathcal{DDHA}$*) holds for* $\mathbb{G}$ *from the assertion in PPC that El Gamal encryption using* $\mathbb{G}$ *is semantically secure (i.e., the equivalence* $\mathcal{LEG} \cong \mathcal{REG}$*).*

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

$$\begin{aligned}&\nu(internal).out(internal,\ pkey(G(1^\eta)))\ |\ in(internal,\ \langle p, g, g^a\rangle).\\ &\qquad out(pub,\ \langle\langle p, g, g^a\rangle,\ 1^\eta\rangle).in(msg,\ \langle m_0, m_1\rangle).\\ &\qquad\quad out(challenge,\ \langle\langle \mathrm{p, g, g^a}\rangle,\ \langle \mathrm{m_0, m_1}\rangle,\ \langle \mathrm{g^b},\ \mathrm{m_0}\cdot \mathrm{g^{ab}}\rangle\rangle\ |\ \mathrm{b} \in_{\mathrm{R}} [1..\mathrm{ord\,G_p}]) \qquad (\mathcal{LEG})\end{aligned}$$

is observationally equivalent to

$$\begin{aligned}&\nu(internal).out(internal,\ pkey(G(1^\eta)))\ |\ in(internal,\ \langle p, g, g^a\rangle).\\ &\qquad out(pub,\ \langle\langle p, g, g^a\rangle,\ 1^\eta\rangle).in(msg,\ \langle m_0, m_1\rangle).\\ &\qquad\quad out(challenge,\ \langle\langle \mathrm{p, g, g^a}\rangle,\ \langle \mathrm{m_0, m_1}\rangle,\ \langle \mathrm{g^b},\ \mathrm{m_1}\cdot \mathrm{g^{ab}}\rangle\rangle | \mathrm{b} \in_{\mathrm{R}} [1..\mathrm{ord\,G_p}]). \qquad (\mathcal{REG})\end{aligned}$$

But then, using the rule (CON) we obtain that

$$\begin{aligned}&\nu(pub).\nu(msg).\nu(challenge).\mathcal{LEG}\ |\ in(pub,\ \langle p, g, g^a\rangle).\\ &\qquad out(msg,\ \langle 1, \mathrm{g^r}\rangle | \mathrm{r} \in_{\mathrm{R}} [1..\mathrm{ord\,G_p}]).in(challenge,\ \langle\langle p, g, g^a\rangle,\ \langle m_0, m_1\rangle,\ \langle g^b, g^c\rangle\rangle).\\ &\qquad\qquad out(ddh,\ \langle \mathrm{p, g, g^a, g^b, g^c}\rangle)\\ &\qquad\qquad\cong\\ &\qquad \nu(pub).\nu(msg).\nu(challenge).\mathcal{REG}\ |\ in(pub,\ \langle p, g, g^a\rangle).\\ &\qquad out(msg,\ \langle 1, \mathrm{g^r}\rangle | \mathrm{r} \in_{\mathrm{R}} [1..\mathrm{ord\,G_p}]).in(challenge,\ \langle\langle p, g, g^a\rangle,\ \langle m_0, m_1\rangle,\ \langle g^b, g^c\rangle\rangle).\\ &\qquad\qquad out(ddh,\ \langle \mathrm{p, g, g^a, g^b, g^c}\rangle) \qquad (\dagger)\end{aligned}$$

It is here that we make use of the fact that all the channel names of $\mathcal{LEG}$ and $\mathcal{REG}$ are bindable. By making them private, we create expressions with only one public action. We then use the fact that the only observables the left-hand side and right-hand side expressions in (†) can generate are on the output on the channel named ddh and look like DDHA-challenge tuples. 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 El Gamal instance generator and a, b are chosen at random from $[1..\mathrm{ord}\, G_p]$. 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 $[1..\mathrm{ord}\, G_p]$. 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. As a consequence, we can use proof rule (NU) to 'fold-in' the private portions into the outputs on the channel ddh and construct an expression that only outputs a DDHA-challenge tuple. That is to say, we can obtain the equivalence:

$$\begin{aligned}&out(d,\ \langle p, g, g^a, g^b, g^{ab}\rangle\ |\ \langle p, g\rangle \in IG(1^\eta), a, b \in [1..\mathrm{ord}\, G_p])\\ &\quad\cong out(d,\ \langle p, g, g^a, g^b, g^c\rangle\ |\ \langle p, g\rangle \in IG(1^\eta), a, b, c \in [1..\mathrm{ord}\, G_p]),\end{aligned}$$

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.

8. 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 $in(c, x).out(d, \mathrm{x} + 2)$, 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 formalise 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 has 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 behaviour, 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 [49]. 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, our bisimilarity relation is only one of an assortment of tools that we hope to use to demonstrate equivalences. While bisimilarity is useful for showing that the distributions induced on observables are identical (i.e., that two expressions are information-theoretically indistinguishable), there are many cryptographically interesting constructs that are only computationally indistinguishable. It stands to reason that we will need a proof technique capable of showing such equivalences directly.

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 El Gamal 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 wish to 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 their helpful discussions and advice on relevant literature. We are also indebted to the anonymous referees for their careful and thorough reading of drafts of this monograph.

References

[1] M. Abadi, C. Fournet, Mobile values, new names, and secure communication, in: 28th ACM Symp. Principles of Programming Languages, 2001, pp. 104–115.

[2] M. Abadi, A.D. Gordon, A bisimulation method for cryptographic protocol, in: Proc. ESOP 98, Lecture Notes in Computer Science, Springer, Berlin, 1998.

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

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

[7] M. Backes, B. Pfitzmann, M. Waidner, Reactively secure signature schemes, in: Proc. Sixth Information Security Conf., Lecture Notes in Computer Science, Vol. 2851, Springer, Berlin, 2003, pp. 84–95.

[8] M. Backes, B. Pfitzmann, M. Waidner, A general composition theorem for secure reactive systems, in: Proc. First Theory of Cryptography Conference, Lecture Notes in Computer Science, Vol. 2951, Springer, Berlin, 2004.

[9] S. Bellantoni, Predicative recursion and computational complexity, Ph.D. Thesis, University of Toronto, 1992.

[10] M. Bellare, A. Desai, D. Pointcheval, P. Rogaway, Relations among notions of security for public-key encryption schemes, in: H. Krawczyk (Ed.), Proc. CRYPTO 1998, Santa Barbara, CA, Lecture Notes in Computer Science, Vol. 1462, Springer, Berlin, 1998, pp. 26–45.

[11] M. Bernardo, R. Gorrieri, A tutorial on EMPA: a theory of concurrent processes with nondeterminism, priorities, probabilities and time, Theoret. Comput. Sci. 202 (1998) 1–54.

[12] D. Boneh, The decision Diffie–Hellman problem, Proc. Third Algorithmic Number Theory Symp., Vol. 1423, 1998, pp. 48–63.

[13] M. Burrows, M. Abadi, R. Needham, A logic of authentication, Proc. Roy. Soc. Ser. A 426 (1871) (1989) 233–271 (also appeared as SRC Research Report 39 and, in a shortened form, in ACM Trans. Comput. Systems 8(1) (1990) 18–36).

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

[15] R. Canetti, Universally composable security: a new paradigm for cryptographic protocols, in: Proc. 42nd IEEE Symp. Foundations of Computer Science, IEEE, 2001(Full version available at ⟨http://eprint.iacr.org/2000/067/⟩).

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

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

[18] R. Canetti, T. Rabin, Universal composition with joint state, in: D. Boneh (Ed.), Proc. CRYPTO 2003, Santa Barbara, CA, Lecture Notes in Computer Science, Vol. 2729, Springer, Berlin, 2003, pp. 265–281.

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

[20] R. Cramer, V. Shoup, A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack, Lecture Notes in Computer Science, Vol. 1462, Springer, Berlin, 1998, pp. 13–25.

[21] L. de Alfaro, Temporal logics for the specification of performance and reliability, in: STACS ’97, Lecture Notes in Computer Science, Vol. 1200, Springer, Berlin, 1997, pp. 165–176.

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

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

[24] W. Diffie, M.E. Hellman, New directions in cryptography, IEEE Trans. Inform. Theory 22 (1976) 644–654.

[25] D. Dolev, C. Dwork, M. Naor, Non-malleable cryptography (extended abstract), in: Proc. 23rd Annu. ACM Symp. Theory of Computing, 1991, pp. 542–552.

[26] D. Dolev, A.C.-C. Yao, On the security of public-key protocols, in: Proc. 22nd Annu. IEEE Symp. Foundations of Computer Science, 1981, pp. 350–357.

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

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

[29] R. Gennaro, An improved pseudo-random generator based on discrete log, in: Proc. CRYPTO 2000, Santa Barbara, CA, Lecture Notes in Computer Science, Vol. 1880, Springer, Berlin, 2000, pp. 469–481 (Revised version available at ⟨http://www.research.ibm.com/people/r/rosario/⟩).

[30] O. Goldreich, Modern Cryptography, Probabilistic Proofs and Pseudorandomness, Springer, Berlin, 1999.

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

[32] O. Goldreich, The Foundations of Cryptography, Vol. 2, Cambridge University Press, Cambridge, May 2004.

[33] S. Goldwasser, M. Bellare, 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] S. Goldwasser, S. Micali, Probabilistic encryption, J. Comput. System Sci. 28 (2) (1984) 270–299 Previous version in STOC 1982.

[35] H. Hansson, Time and probabilities in formal design of distributed systems, Real-Time Safety Critical Systems, Elsevier, Amsterdam, 1994.

[36] H. Hansson, B. Jonsson, A framework for reasoning about time and reliability, in: Proc. Real Time Systems Symp., IEEE, 1989, pp. 102–111.

[37] M. Hofmann, Type systems for Polynomial-Time Computation, Habilitation Thesis, Darmstadt; see ⟨http://www.dcs.ed.ac.uk/home/mxh/papers.html⟩, 1999.

[38] M. Huth, M.Z. Kwiatkowska, Quantitative analysis and model checking, in: Lecture Notes in Computer Science, Springer, Berlin, 1997, pp. 111–122.

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

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

[41] P.D. Lincoln, J.C. Mitchell, M. Mitchell, A. Scedrov, A probabilistic poly-time framework for protocol analysis, in: M.K. Reiter (Ed.), Proc. Fifth ACM Conf. Computer and Communications Security, San Francisco, CA, ACM Press, New York, 1998, pp. 112–121.

[42] P.D. Lincoln, J.C. Mitchell, M. Mitchell, A. Scedrov, Probabilistic polynomial-time equivalence and security protocols, in: J.M. Wing, J. Woodcock, J. Davies (Eds.), Formal Methods World Congress, Vol. I, Toulouse, France, Lecture Notes in Computer Science, Vol. 1708, Springer, Berlin, 1999, pp. 776–793.

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

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

[45] P. Mateus, J.C. Mitchell, A. Scedrov, Composition of cryptographic protocols in a probabilistic polynomial-time process calculus, in: R.M. Amadio, D. Lugiez (Eds.), 14th Internat. Conf. Concurrency Theory, Marseille, France, Lecture Notes in Computer Science, Vol. 2761, Springer, Berlin, 2003, pp. 327–349.

[46] C. Meadows, Analyzing the Needham–Schroeder public-key protocol: a comparison of two approaches, in: Proc. European Symp. Research in Computer Security, Springer, Berlin, 1996, pp. 351–364.

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

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

[49] R. Milner, Communication and Concurrency, International Series in Computer Science, Prentice-Hall, Englewoods Cliffs, NJ, 1989.

[50] J.C. Mitchell, M. Mitchell, A. Scedrov, A linguistic characterization of bounded oracle computation and probabilistic polynomial time, in: Proc. 39th Annu. IEEE Symp. Foundations of Computer Science, Palo Alto, CA, IEEE, 1998, pp. 725–733.

[51] J.C. Mitchell, M. Mitchell, U. Stern, Automated analysis of cryptographic protocols using Murφ, in: Proc. IEEE Symp. Security and Privacy, 1997, pp. 141–151.

[52] J.C. Mitchell, A. Ramanathan, A. Scedrov, V. Teague, A probabilistic polynomial-time calculus for the analysis of cryptographic protocols (preliminary report), in: S. Brookes, M. Mislove (Eds.), 17th Annu. Conf. Mathematical Foundations of Programming Semantics, Arhus, Denmark, Vol. 45, May 2001 (electronic notes in Theoretical Computer Science).

[53] R. Needham, M. Schroeder, Using encryption for authentication in large networks of computers, Commun. ACM 21 (12) (1978) 993–999.

[54] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, Reading, MA, 1994.

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

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

[57] B. Pfitzmann, M. Waidner, Composition and integrity preservation of secure reactive systems, in: Seventh ACM Conf. Computer and Communications Security, Athens, ACM, New York, November 2000, pp. 245–254, (preliminary version: IBM Research Report RZ 3234 (# 93280) 06/12/00, IBM Research Division, Zürich, June 2000).

[58] B. Pfitzmann, M. Waidner, A model for asynchronous reactive systems and its application to secure message transmission, in: IEEE Symp. Security and Privacy, Washington, 2001, pp. 184–200.

[59] A. Ramanathan, J.C. Mitchell, A. Scedrov, V. Teague, Probabilistic bisimulation and equivalence for security analysis of network protocols, in: I. Walukiewicz (Ed.), Foundations of Software Science and Computation Structures, Seventh Internat. Conf., FOSSACS 2004, Barcelona, Spain, Lecture Notes in Computer Science, Vol. 2987, Springer, Berlin, 2004, pp. 468–483 (summarizes results of [60]).

[60] A. Ramanathan, J.C. Mitchell, A. Scedrov, V. Teague, Probabilistic Bisimulation and Equivalence for Security Analysis of Network Protocols, Technical Report, 2004 (see ⟨http://www-cs-students.stanford.edu/~ajith/⟩).

[61] A.W. Roscoe, Modeling and verifying key-exchange protocols using CSP and FDR, in: CSFW 8, IEEE Computer Society Press, 1995, pp. 98–99.

[62] P.Y.A. Ryan, S.A. Schneider, An attack on a recursive authentication protocol—A cautionary tale, Inform. Process. Lett. 65 (1) (1998) 7–10.

[63] S. Schneider, Security properties and CSP, in: IEEE Symp. Security and Privacy, Oakland, CA, 1996.

[64] Y. Tsiounis, M. Yung, On the security of El Gamal-based encryption, Lecture Notes in Computer Science, Vol. 1431, Springer, Berlin, 1998, pp. 117–134.

[65] R.J. van Glabbeek, S.A. Smolka, B. Steffen, Reactive, generative, and stratified models of probabilistic processes, Internat. J. Inform. and Comput. 121 (1) (1995).

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