A probabilistic polynomial-time calculus for analysis of cryptographic protocols (Preliminary report)

J. Mitchell*†‡ A. Ramanathan*
Stanford University

A. Scedrov*§¶
University of Pennsylvania

V. Teague*‖
Stanford University

Abstract

We describe properties of a process calculus that has been developed for the purpose of analyzing security protocols. The process calculus is a restricted form of π-calculus, with bounded replication and probabilistic polynomial-time expressions allowed in messages and boolean tests. In order to avoid problems expressing security in the presence of nondeterminism, messages are scheduled probabilistically instead of nondeterministically. We prove that evaluation may be completed in probabilistic polynomial time and develop properties of a form of asymptotic protocol equivalence that allows security to be specified using *observational equivalence,* a standard relation from programming language theory that involves quantifying over possible environments that might interact with the protocol. We also relate process equivalence to cryptographic concepts such as pseudo-random number generators and polynomial-time statistical tests.

1 Introduction

A variety of methods are used for analyzing and reasoning about security protocols. The main systematic or formal approaches include specialized logics such as BAN logic [BAN89, DMP01], special-purpose tools designed for cryptographic protocol analysis [KMM94], and theorem proving [Pau97b, Pau97a] and model-checking methods using several general purpose tools described in [Low96, Mea96, MMS97, Ros95, Sch96]. 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. In the common model, largely derived from [DY81] and suggestions found in [NS78] (see, *e.g.,* [CDL+99]), a protocol adversary is allowed to nondeterministically choose among possible actions. This is a convenient idealization, intended to give the adversary a chance to find an attack if there is one. In the presence of nondeterminism, however, the set of messages an adversary may use to interfere with a protocol must be restricted severely. Although the Dolev-Yao-style assumptions make protocol analysis tractable, they also

*Partially supported by DoD MURI "Semantic Consistency in Information Exchange," ONR Grant N00014-97-1-0505.

†Additional support from DARPA Contract N66001-00-C-8015.

‡Additional support from NSF CCR-9629754.

§Additional support by OSD/ONR CIP/SW URI "Software Quality and Infrastructure Protection for Diffuse Computing," ONR Grant N00014-01-1-0795.

¶Additional support from NSF Grants CCR-9800785 and CCR-0098096.

‖Partially supported by Stanford Graduate Fellowship.

make it possible to "verify" protocols that are in fact susceptible to simple attacks that lie outside the adversary model. Another limitation is that a deterministic or nondeterministic setting does not allow us to analyze probabilistic protocols.

In this paper we describe some general concepts in security protocol analysis, mention some of the competing approaches, and describe some technical properties of a process calculus that was proposed earlier in [LMMS98, LMMS99] as the basis for a form of protocol analysis that is formal, yet closer in foundations to the mathematical setting of modern cryptography. The framework relies on a language for defining communicating polynomial-time processes [MMS98]. The reason we restrict processes to probabilistic polynomial time is so that we can reason about the security of protocols by quantifying over all "adversarial" processes definable in the language. In effect, establishing a bound on the running time of an adversary allows us to relax the simplifying assumptions. Specifically, it is possible to consider adversaries that might send randomly chosen messages, or perform sophisticated (yet probabilistic polynomial-time) computation to derive an attack from messages it overhears on the network. An important aspect of our framework is that we can analyze probabilistic as well as deterministic encryption functions and protocols. Without a probabilistic framework, it would not be possible to analyze an encryption function such as ElGamal [ElG85], for which a single plaintext may have more than one ciphertext.

Some of the basic ideas of this work are outlined in [LMMS98], with the term language presented in [MMS98] and further example protocols considered in [LMMS99]. The closest technical precursor is the Abadi and Gordon spi-calculus [AG99, AG98] which uses observational equivalence and channel abstraction but does not involve probability or computational complexity bounds; subsequent related work is cited in [AF01], for example. Prior work on CSP and security protocols, e.g., [Ros95, Sch96], also uses process calculus and security specifications in the form of equivalence or related approximation orderings on processes.

Although our main long-term objective is to base protocol analysis on standard cryptographic assumptions, this framework may also shed new light on basic questions in cryptography. In particular, the characterization of "secure" encryption function, for use in protocols, does not appear to have been completely settled. While the definition of *semantic security* in [GM84] appears to have been accepted, there are stronger notions such as *non-malleability* [DDN91] that are more appropriate to protocol analysis. In a sense, the difference is that semantic security is natural for the single transmission of an encrypted message, while non-malleability accounts for vulnerabilities that may arise in more complex protocols. Our framework provides a setting for working backwards from security properties of a protocol to derive necessary properties of underlying encryption primitives. While we freely admit that much more needs to be done to produce a systematic analysis method, we believe that a foundational setting for protocol analysis that incorporates probability and complexity restrictions has much to offer in the future.

2 Preliminaries

2.1 Probabilistic Functions

Definition 2.1. We define a probabilistic function $\mathcal{F}$ on the sets X, Y as a function $\mathcal{F} : X \times Y \to [0, 1]$ that satisfies the following condition:

$$\forall x \in X : \sum_{y \in Y} \mathcal{F}(x, y) \leq 1 \tag{2.1}$$

For some $x \in X, y \in Y$, if $\mathcal{F}(x,y) = p$, we say that $\mathcal{F}$ takes on the value y at x with probability p or that $\mathcal{F}(x) = y$ with probability p.

2.2 Composition of Probabilistic Functions

Definition 2.2. We define the *composition* $\mathcal{F} : X \times Z \rightarrow [0,1]$ of two probabilistic functions $\mathcal{F}_1 : X \times Y \rightarrow [0,1]$ and $\mathcal{F}_2 : Y \times Z \rightarrow [0,1]$ as a probabilistic function that satisfies the following condition:

$$\forall x \in X. \forall z \in Z : \mathcal{F}(x,z) = \sum_{y \in Y} \mathcal{F}_1(x,y) \cdot \mathcal{F}_2(y,z) \tag{2.2}$$

We write the composition of $\mathcal{F}_1$ and $\mathcal{F}_2$ as $\mathcal{F}_2 \circ \mathcal{F}_1$.

Lemma 2.3. *Given two probabilistic functions $\mathcal{F}_1 : X \times Y \rightarrow [0,1]$ and $\mathcal{F}_2 : Y \times Z \rightarrow [0,1]$, the composition $\mathcal{F}_2 \circ \mathcal{F}_1 = \mathcal{F} : X \times Z \rightarrow [0,1]$ is a probabilistic function, i.e., it satisfies the condition $\forall x \in X : \sum_{z \in Z} \mathcal{F}(x,z) \leq 1$.*

Proof. For any fixed $x \in X$:

$$\begin{array}{rcll} \sum\limits_{z \in Z} \mathcal{F}(x,z) & = & \sum\limits_{z \in Z} \sum\limits_{y \in Y} \mathcal{F}_1(x,y) \cdot \mathcal{F}_2(y,z) & \text{by defn. 2.2} \\ & = & \sum\limits_{y \in Y} \mathcal{F}_1(x,y) \cdot \sum\limits_{z \in Z} \mathcal{F}_2(y,z) & \text{factoring} \\ & \leq & 1 & \text{by defn. 2.1} \end{array}$$

Hence, composition is a probabilistic function. □

2.3 Terminology

In what follows, P denotes a process, T denotes a term, $p(x)$ denotes one of an infinity of *bandwidth polynomials* such that $\forall a \in \mathbb{N} : p(a) \geq 0$. $c_{p(x)}$ denotes one of a countable infinity $\mathbf{C}_{q(x)}$ of *channel names* associated with the polynomial $p(x)$ in such a way that no channel name is associated with more than one polynomial.

There is a special *security parameter, n,* that can appear in terms (section 2.3.1), as the parameter of the replication operator (defn. 2.13), as the argument to the bandwidth polynomials, and in contexts (defn. 2.21). The bandwidth polynomial gives, for each value of n, the maximum number of bits that can be transmitted on that channel (explored in more detail in defn. 3.3). The two-place relation $\equiv$ stands for syntactic identity. We denote variables by the letters x, y and so on. Finally, the function $\gamma(x)$ is a polynomial such that $\forall a \in \mathbb{N} : \gamma(a) \geq 0$.

Definition 2.4. [Ata99] An *oracle Turing machine* is a Turing machine with an extra oracle tape and three extra states q_{query}, q_{yes} and q_{no}. When the machine enters state q_{query} control passes to the state q_{yes} if the contents on the oracle tape are in the oracle set; otherwise, control passes to the state q_{no}.

Given an oracle Turing machine M, we will write $M(\varphi, \vec{x})$ to specify the behavior of M on input $\vec{x}$ using oracle φ.

We only consider binary oracles. Another way of saying this is that the binary oracle φ determines a set X_φ since we can write X_φ as $\{x | \varphi(x)\}$. Thus, a query to φ is a binary result saying if $x \in X_\varphi$.

Definition 2.5. An oracle Turing machine M runs in *oracle polynomial time* if there exists a polynomial q such that for all oracles φ, $M(\varphi, \vec{x})$ halts in time $q(|\vec{x}|)$ where $\vec{x} = x_1, \ldots, x_m$ and $|\vec{x}| = |x_1|, \ldots, |x_m|$.

In order to query the oracle, M must write the number at which it wants to query the oracle on the oracle tape. If M runs in oracle polynomial time, then $M(\vec{x})$ queries the oracle set with at most $q(|\vec{x}|)$ bits.

Definition 2.6. Let M be a poly-time oracle Turing machine. We can view M as a *probabilistic poly-time Turing machine* if we randomly choose an oracle from the space of oracles that can be queried in the time bound of M. More precisely, let M be an oracle machine running in time bounded by the polynomial q. Since $M(\vec{x})$ can only query the oracle with at most $q(\vec{x})$ bits, we have a finite space of oracles that run in time bounded by q. Call this space $\mathbf{q}$. Then, we can view M as a probabilistic poly-time Turing machine where we say that $M(\vec{x}) = y$ with probability p if, choosing an oracle φ uniformly at random from $\mathbf{q}$, the probability that $M(\vec{x}, \varphi) = y$ is p.

Given a poly-time probabilistic Turing machine M, we will write $M(\vec{x})$ to specify the behavior of M on input $\vec{x}$ using a randomly chosen oracle.

Definition 2.7. A probabilistic poly-time Turing machine M *computes* a probabilistic function $\mathcal{F}$ if for all inputs $\vec{x}$ and for all outputs y we have that $\mathcal{F}(\vec{x}, y) = \mathrm{Prob}\big[M(\vec{x}) = y\big]$.

Definition 2.8. A probabilistic function $\mathcal{F}$ is *poly-time* if it is computed by a probabilistic poly-time Turing machine.

2.3.1 Terms

A functional term calculus based on [Hof99, Bel92] and a semantics for that calculus are studied in [MMS98]. Though we omit the details here, we do note that terms can contain the security parameter n as well as a function $rand$ that returns one random bit.

For each term T with variables $x_1, \ldots, x_k$, there is a probabilistic poly-time Turing machine M_T of $k+1$ inputs and a polynomial $q_T(v_1, \ldots, v_{k+1})$ such that the following two theorems hold:

Theorem 2.9. *[MMS98] Let T be a term with k variables and let M_T be the machine associated with T. Then $M_T(a_1, \ldots, a_k, n)$ halts in time at most $q_T(|a_1|, \ldots, |a_k|, |n|)$.*

Theorem 2.10. *[MMS98] For each probabilistic poly-time function f, there exists a term T such that M_T computes f.*

While there may be many assignments of Turing machines to terms that satisfy the above two theorems, for our purposes the exact nature of the assignment is irrelevant. In fact, we will simply consider the probabilistic poly-time Turing machine M_T to define the meaning of the term T.

Definition 2.11. We write $T \xrightarrow{r}_e a$ on inputs $x_1, \ldots, x_k$ if the probability that $M_T(x_1, \ldots, x_k, n) = a$ is r. We say that T *evaluates* to a with probability r.

Definition 2.12. We define two probabilities here: (1) the probability of two terms evaluating to the same number, and, (2) the probability of two terms evaluating to different numbers.

1. $\mathrm{Prob}_{=}\big[T_1, T_2\big] = \sum_a s_1 s_2$,
   where $a \in \mathbb{N}, T_1 \xrightarrow{s_1}_e a, T_2 \xrightarrow{s_2}_e a$, and
2. $\mathrm{Prob}_{\neq}\big[T_1, T_2\big] = \sum_{\langle a_1, a_2 \rangle} s_1 s_2$,
   where $a_1, a_2 \in \mathbb{N}, a_1 \neq a_2, T_1 \xrightarrow{s_1}_e a_1, T_2 \xrightarrow{s_2}_e a_2$.

2.3.2 Processes

Definition 2.13. The syntax of processes is given by the following grammar:

$$\begin{array}{lll} P & ::= & \mathbf{0} & \text{(termination)} \\ & & \nu_{c_{p(|n|)}}.(P) & \text{(private channel)} \\ & & c_{p(|n|)}(x).(P) & \text{(input)} \\ & & c_{p(|n|)}\langle T\rangle & \text{(output)} \\ & & [T = T].(P) & \text{(match)} \\ & & (P \mid P) & \text{(parallel composition)} \\ & & !_{\gamma(|n|)}.(P) & \text{(}\gamma(|n|)\text{-fold replication)} \end{array}$$

Every input or output on a private channel must appear within the scope of a ν-operator binding that channel. Otherwise the expression is considered unevaluable.

Another word about channel names. The polynomial $p(|n|)$ associated with the channel c is a *bandwidth parameter* on c—for further details see section 3.3. Also, as mentioned in section 2.3, no channel name is associated with more than one polynomial.

For simplicity, after fixing n, when we evaluate a process P, we replace all subexpressions of P of the form $!_{\gamma(|n|)}.R$ with $\gamma(|n|)$ copies of R in parallel. Furthermore, we define $!_{\gamma(|n|)}.R$ to be left associative. Finally, we assume that all channel names and variable names are α-renamed apart.

Finally, in what follows we will tend to omit parentheses if the parse tree of an expression is unambiguous.

Definition 2.14. We define an *input expression* as a process of the form $c_{p(|n|)}(x).P$ and an *output expression* is of the form $c_{p(|n|)}\langle T\rangle$.

Definition 2.15. We call a process expression with no free variables a *closed* process expression.

Definition 2.16. Let P be a process expression and let T be a term. If T is not in the scope of any input operator, we say that T is an *exposed term.* Similarly, let $[T_1 = T_2].R$ be a subexpression of P. We will say that $[T_1 = T_2].R$ is an *exposed match* if it does not appear in the scope of any input operator.

Definition 2.17. We define the indicator function $\mathrm{Eq?}_{\mathbf{0}}(P)$ as a unary function on processes that has the value 1 when its input is the $\mathbf{0}$ process and 0 otherwise:

$$\mathrm{Eq?}_{\mathbf{0}}(Q) = \begin{cases} 0 & \text{if } Q \not\equiv \mathbf{0}, \\ 1 & \text{if } Q \equiv \mathbf{0}. \end{cases}$$

Definition 2.18. *outerEval* is a probabilistic function on closed process expressions and closed process expressions with all exposed terms reduced to atoms and no exposed matches:

$$\begin{array}{ll} \mathit{outerEval}(\mathbf{0}, \mathbf{0}) & = 1 \\ \mathit{outerEval}(\nu_{c_{p(|n|)}}.(P), \nu_{c_{p(|n|)}}.(P')) & = s_1 \\ \mathit{outerEval}(c_{p(|n|)}(x).(Q), c_{p(|n|)}(x).(Q)) & = 1 \\ \mathit{outerEval}(c_{p(|n|)}\langle T\rangle, c_{p(|n|)}\langle a\rangle) & = s_2 \\ \mathit{outerEval}([T_1 = T_2].(Q_1), Q_2) & \\ \quad = s_3 \cdot \mathrm{Prob}_{=}[T_1, T_2] + \mathrm{Eq?}_{\mathbf{0}}(Q_2) \cdot \mathrm{Prob}_{\neq}[T_1, T_2] & \\ \mathit{outerEval}((P_1 \mid P_2), (P_1' \mid P_2')) & = s_4 \cdot s_5 \end{array}$$

$outerEval(P, Q) \qquad\qquad = 0$ otherwise.

where $outerEval(P, P') = s_1$,
$a \in \mathbb{N}, T \xrightarrow{s_2}_e a$,
$outerEval(Q_1, Q_2) = s_3$,
$outerEval(P_1, P_1') = s_4$,
$outerEval(P_2, P_2') = s_5$

If $outerEval(P, P') = s$ then we say that P *outer-evaluates* to P' with probability s, or that $P \xrightarrow{s}_o P'$.

The match case in the definition of $outerEval$ bears some discussion. Consider the match $[T_1 = T_2].P$. We wish to compute the probability of P outer-evaluating to P'. There are two distinct cases here:

1. $P' \not\equiv \mathbf{0}$, and,
2. $P' \equiv \mathbf{0}$.

In the first case, we will only outer-evaluate P if the match succeeds, *i.e.*, $T_1 = T_2$. In the second case, however, we can arrive at $\mathbf{0}$ either by passing the match and outer-evaluating P to $\mathbf{0}$ or by simply failing the match. Hence, the probability of outer-evaluating to P' is given by the sum of the product of the probability of passing the match and outer-evaluating P to P' and, in the case that we wish to outer-evaluate to a $\mathbf{0}$, failing the match—hence the use of the indicator function in the second term of the sum.

We must be sure, however, that we do not over count in this expression. From the definition of cases 1 and 2 we can see that P' can never satisfy both cases simultaneously—hence cases 1 and 2 are disjoint. We do not over count inside case 2 as the sets over which we define the two summations in case 2 (one from the $\text{Prob}_=[T_1, T_2]$ term and one from the $\text{Prob}_{\neq}[T_1, T_2]$ term) are disjoint.

Lemma 2.19. *Let P, Q and R be closed process expressions such that $P \xrightarrow{s_1>0}_o Q$ and $Q \xrightarrow{s_2>0}_o R$. Then, Q has all exposed terms reduced to atoms and no exposed matches, $Q \equiv R$, and $s_2 = 1$.*

Proof. By a straightforward induction on the structure of P. We give a sketch of the proof.

The bases, $P \equiv \mathbf{0}$ and $P \equiv c_{p(|n|)}\langle T\rangle$, are easy to see.

In the case that $P \equiv \nu_{c_{p(|n|)}}.(P')$ we notice that $Q \equiv \nu_{c_{p(|n|)}}.(Q')$ where $P' \xrightarrow{s_1}_o Q'$. By the inductive hypothesis, we have that $Q' \xrightarrow{1}_o Q'$ and the result follows. In the case that $P \equiv c_{p(|n|)}(x).P'$ we notice that $P \xrightarrow{1}_o Q$ and the result follows. In the case that $P \equiv [T_1 = T_2].P'$ we have two cases. If both T_1 and T_2 evaluate to the same number then $Q \equiv Q'$ where $P' \xrightarrow{t_1}_o Q'$. By the inductive hypothesis $Q' \xrightarrow{1}_o Q'$ and the result follows. In the case that T_1 and T_2 evaluate to different numbers we can see that $Q \equiv \mathbf{0}$ and the result follows easily. In the case that $P \equiv P_1 \mid P_2$ the inductive hypothesis gives us that $P_1 \xrightarrow{s_1'}_o Q_1 \xrightarrow{1}_o Q_1$ and $P_2 \xrightarrow{s_2'}_o Q_2 \xrightarrow{1}_o Q_2$ and the result follows. $\square$

So, in a composition of two $outerEvals$, only the first $outerEval$ does any work. Specifically, it converts all exposed terms to atoms and eliminates all exposed matches.

Corollary 2.20. *Let P and Q be processes and let $\rightarrow_{o,m}$ represent the m-fold composition of $\rightarrow_o$. If $P \xrightarrow{s>0}_o Q$, then $Q \xrightarrow{1}_{o,m} Q$.*

Proof. Directly from lemma 2.19. □

Definition 2.21. Let Γ be the set of expressions generated by the following grammar:

$$\begin{array}{rcl} C[\ ] & ::= & [\ ]_i \\ & & \nu_{c_{p(|n|)}}.(C[\ ]) \\ & & c_{p(|n|)}(x).(C[\ ]) \\ & & P \\ & & [T = T].(C[\ ]) \\ & & (C[\ ] \mid C[\ ]) \\ & & !_{\gamma(|n|)}.(C[\ ]) \end{array}$$

where $i \in \mathbb{N}$.

A *context* is a process expression that is a member of Γ whose "holes" (indicated by numbered empty square brackets $[\ ]_i$) are numbered uniquely.

Given a context $C[\ ]$ and processes $P_1, \ldots, P_m$, the notation $C[P_1, \ldots, P_m]$ means that we substitute the process P_i for the hole $[\ ]_i$ in $C[\ ]$.

Finally, we will indicate the number of holes is a context by placing an (often un-numbered) empty square bracket for each hole in the context. For example, $C[\ ]$ will be a one-hole context while $D[\ ][\ ][\ ]$ has three holes.

Definition 2.22. We will say that B is a *sub-process* of A if there exists a context $C[\ ]$ such that $C[B] \equiv A$. If $C[\ ] \not\equiv [\ ]$ we will say that B is a *strict* sub-process of A.

Definition 2.23. Let P and Q be process expressions and $c_{p(|n|)}$ be a channel. We say that P is *bound by* $c_{p(|n|)}$ *in* Q if $c_{p(|n|)}(x).D[P]$ is a sub-process of Q for some variable x and some context expression $D[\ ]$.

Definition 2.24. Let $C[\ ]$ be a context and P a process. Then, $C[\ ]$ is *minimal for P* if every free variable of P is bound in $C[P]$ and each channel to which P is bound in $C[P]$ binds a free variable of P.

Definition 2.25. The set of *schedulable processes* $S(P)$ of an outer-evaluated process P is defined inductively as:

$$\begin{array}{rcl} S(\mathbf{0}) & = & \emptyset \\ S(\nu_{c_{p(|n|)}}.(Q)) & = & S(Q) \\ S(c_{p(|n|)}(x).(Q)) & = & \{c_{p(|n|)}(x).(Q)\} \\ S(c_{p(|n|)}\langle T\rangle) & = & \{c_{p(|n|)}\langle T\rangle\} \\ S((Q_1 \mid Q_2)) & = & S(Q_1) \cup S(Q_2) \end{array}$$

Note that every process in $S(P)$ is either waiting for input or ready to output.

Definition 2.26. The set of *communication triples* $C(P)$ is defined as:

$$\begin{array}{rl} \{\langle P_1, P_2, Q_{P_1,P_2}[\ ]\rangle \mid & P_1 \equiv c_{p(|n|)}\langle a\rangle, P_2 \equiv c_{p(|n|)}(x).R, \\ & P_i \in S(P), \\ & P \equiv Q_{P_1,P_2}[P_1, P_2]\} \end{array}$$

Given a process P and a context $Q_{P_1,P_2}[\ ]$, P_1 and P_2 are uniquely determined.

Example 2.27. Consider the process expression:

$$P \equiv c_{p(|n|)}\langle 0\rangle \mid c_{p(|n|)}\langle 0\rangle \mid c_{p(|n|)}(x).d_{q(|n|)}\langle 1\rangle$$

The process expression $c_{p(|n|)}(x).d_{q(|n|)}\langle 1\rangle$ could receive its input from either one of the two output expressions. While both communications look the same, their outputs come from distinct locations in P. We distinguish between the two communications by associating the context $[\ ]_1 \mid c_{p(|n|)}\langle 0\rangle \mid [\ ]_2$ with the first communication (the one where the output 'came' from the first process in the parallel composition) and associating the context $c_{p(|n|)}\langle 0\rangle \mid [\ ]_1 \mid [\ ]_2$ with the second communication (the pair where the output 'came' from the second process in the parallel composition). So, $C(P)$ is the set:

$$\Big\{\langle c_{p(|n|)}\langle 0\rangle, c_{p(|n|)}(x).d_{q(|n|)}\langle 1\rangle, [\ ]_1 \mid c_{p(|n|)}\langle 0\rangle \mid [\ ]_2\rangle\ ,$$
$$\langle c_{p(|n|)}\langle 0\rangle, c_{p(|n|)}(x).d_{q(|n|)}\langle 1\rangle, c_{p(|n|)}\langle 0\rangle \mid [\ ]_1 \mid [\ ]_2\rangle\Big\}$$

Definition 2.28. The set of *eligible processes* $E(P)$ is defined as:

$$E(P) = \begin{cases} C(P) & \text{iff every channel in } C(P) \text{ is public,} \\ C(P)|_{\text{private channels}} & \text{otherwise.} \end{cases}$$

Note that restricting $E(P)$ to just private channels anytime $C(P)$ contains a private channel ensures that all private communications occurs prior to public communication. This allows us to "hide" private communications from a prying observer by making private communications privileged.

3 Evaluation

3.1 Scheduler Reduction

Definition 3.1. We write $[a/x]\,P$ to mean "substitute a for all free occurrences of x in P." We will define the *substitution operation* as follows:

$$\begin{array}{lcl}
[a/x]\,\mathbf{0} & = & \mathbf{0} \\
[a/x]\,\nu_{c_{p(|n|)}}.(Q) & = & \nu_{c_{p(|n|)}}.([a/x]\,Q) \\
[a/x]\,c_{p(|n|)}(y).(Q) & = & c_{p(|n|)}(y).([a/x]\,Q) \\
[a/x]\,c_{p(|n|)}(x).(Q) & = & c_{p(|n|)}(x).(Q) \\
[a/x]\,c_{p(|n|)}\langle T\rangle & = & c_{p(|n|)}\langle(\lambda x.T)\ a\rangle \\
[a/x]\,[T_1 = T_2].(Q) & = & [(\lambda x.T_1)\ a = (\lambda x.T_2)\ a].([a/x]\,Q) \\
[a/x]\,(P_1 \mid P_2) & = & ([a/x]\,P_1 \mid [a/x]\,P_2) \\
[a/x]!_{\gamma(|n|)}.(Q) & = & !_{\gamma(|n|)}.([a/x]\,Q)
\end{array}$$

We can generalize the substitution operation to contexts, *i.e.*, to substituting a for all free occurrences of x in $C[\ ]$.

Definition 3.2. We define a scheduler S to be a probabilistic function from sets of communication triples to communication triples such that for every set of communication triples E, if $S(E, e) > 0$ then $e \in E$. If $S(E, e) = r$ we will say that the scheduler S picks the communication triple e from the set E with probability r.

Definition 3.3. Let a be a natural number, n the value of the security parameter, and S a scheduler. We will say that P reduces to P' in *one communication step with respect to S with probability r* if and only if:

$$\begin{aligned} \exists e \equiv \langle P_1, P_2, Q_{P_1,P_2}[\ \ ]\rangle \in E(P) : & \ S(E(P), e) = r, \\ & P_1 \equiv c_{p(|n|)}\langle a \rangle, \\ & P_2 \equiv c_{p(|n|)}(x).P_2', \\ & P' \equiv Q_{P_1,P_2}\left[\mathbf{0}, [a'/x]P_2'\right], \\ & a' = a \bmod 2^{p(|n|)} - 1. \end{aligned}$$

We write this as $P \xrightarrow{r}_S P'$.

We reduce a modulo $2^{p(|n|)} - 1$ in order to ensure that at most only the $p(|n|)$ least significant bits of a are transmitted.

By stipulating that at most $p(|n|)$ bits of a message can be transmitted we ensure that we cannot create exponential-time processes out of polynomial-time process expressions—something that we can do if we allowed arbitrary length messages to be transmitted. Consider the process expression:

$$P \equiv !_{\gamma(|n|)}.c_{p(|n|)}(x).c_{p(|n|)}\langle x^2 \rangle \mid c_{p(|n|)}\langle 2 \rangle$$

Let the polynomial $\gamma(|x|)$ return x always. While more complex polynomials will make the results more dramatic, this choice of γ simplifies matters. It is clear that P will square its input (here initialized to 2) n times. We will call the output of this process P_n.

The table below shows outputs for several different values of n:

n	P_n	$\lvert P_n \rvert$
1	$4 = 2^2$	$2 = 2^1$
2	$16 = 2^4$	$4 = 2^2$
3	$256 = 2^8$	$8 = 2^3$
4	$65,536 = 2^{16}$	$16 = 2^4$
5	$4,294,967,296 = 2^{32}$	$32 = 2^5$
$\vdots$	$\vdots$	$\vdots$

Clearly, P outputs values of length exponential in n. Now if the output of P is used as the input to some poly-time process expression Q, then we will obtain an exponential-time process since Q must run on exponentially long values. However, if we truncate the messages transmitted then no message can ever get too long, *i.e.*, exponentially long.

3.2 Evaluation

3.2.1 Macro-Step Reduction

Intuitively, we define the *macro-step reduction* of P to P' based on the following three stage procedure:

1. We outer-evaluate P, yielding R.

2. The scheduler picks a process triple E from $E(R)$ and performs the selected communication. This results in the process expression R'.

3. Finally, we outer-evaluate R', yielding P'.

Thus, macro-step reduction is merely shorthand for a procedure whereby we select a communication to perform and evaluate the result until it becomes necessary for the next communication to be selected. The first outer-evaluation simply ensures that the process expression on which we perform the communication step satisfies the condition on $S(P)$ specified in defn. 2.25. Barring the first outer-evaluation, then, the evaluation of process expressions becomes an alternating series of process evaluations (the outer-evaluations) and communication steps. We will define macro-step reduction so that it is just a communication step followed by a process evaluation—with a preceding process evaluation for the reason mentioned above.

Definition 3.4. More precisely, *macro-step reduction with respect to scheduler S* is the probabilistic function defined by the composition:

$$\rightarrow_o \circ (\rightarrow_S \circ \rightarrow_o) \tag{3.1a}$$

The probability r of the macro-step reduction of P to P' is obtained directly from definition 2.2:

$$r = \sum_{Q,Q' \mid P \xrightarrow{a}_o Q \xrightarrow{b}_S Q' \xrightarrow{c}_o P'} abc \tag{3.1b}$$

We will write the macro-step reduction of P to P' with respect to the scheduler S with probability r as $P \xrightarrow{r}_{1,S} P'$.

3.2.2 m-Step Reduction

Definition 3.5. Let P and P' be two process expressions. Define the set $\mathrm{Paths}_{P,P'}$ as:

$$\{\{Q_1, \ldots, Q_{m-1}\} \mid P \xrightarrow{p_1}_{1,S} Q_1 \xrightarrow{p_2}_{1,S} \cdots \xrightarrow{p_{m-1}}_{1,S} Q_{m-1} \xrightarrow{p_m}_{1,S} P'\}$$

We call $\mathrm{Paths}_{P,P'}$ *the set of paths from P to P'*.

Definition 3.6. The probability that P *reduces to P' in m steps with respect to the scheduler S* is given by:

$$\sum_{\substack{\{Q_1,\cdots,Q_{m-1}\} \in \mathrm{Paths}_{P,P'} \\ P \xrightarrow{p_1}_{1,S} Q_1 \xrightarrow{p_2}_{1,S} \cdots \xrightarrow{p_{m-1}}_{1,S} Q_{m-1} \xrightarrow{p_m}_{1,S} P'}} p_1 p_2 \cdots p_m$$

We write the m-step reduction of P to P' with probability r as $P \xrightarrow{r}_{m,S} P'$.

3.2.3 Evaluation

Definition 3.7. We say that P *evaluates to Q with respect to the scheduler S* if, using the scheduler S, P reduces to Q via some number of macro-steps. We write the evaluation of P to P' with respect to the scheduler S as $P \hookrightarrow_S P'$.

4 Polynomial-Time Bound on Evaluation

We are interested in the worst-case time of evaluation. Since the probability of a particular reduction has no bearing on the worst-case time of evaluation, we need not consider the probabilities of the various reductions in our analysis.

Our bound on evaluation will be derived as a certain combination of three bounds: a bound on the length of any evaluation sequence, a bound on any outer-evaluation step, and a bound on any communication step.

4.1 A Bound on the Length of any Evaluation Sequence

Definition 4.1. Let P be a process expression. We define $inputs(P)$, the *number of inputs in* P, inductively as follows:

$$\begin{array}{lcl}
inputs(\mathbf{0}) & = & 0 \\
inputs(\nu_{c_{p(|n|)}}.(Q)) & = & inputs(Q) \\
inputs(c_{p(|n|)}(x).(Q)) & = & 1 + inputs(Q) \\
inputs(c_{p(|n|)}\langle T\rangle) & = & 0 \\
inputs([T_1 = T_2].(Q)) & = & inputs(Q) \\
inputs((Q_1 \mid Q_2)) & = & inputs(Q_1) + inputs(Q_2) \\
inputs(!_{\gamma(|n|)}.(Q)) & = & \gamma(|n|) \cdot inputs(Q)
\end{array}$$

It is clear from the definition that $inputs(P)$ is a polynomial in $|n|$ that is positive for all input values.

Definition 4.2. Let P be a process expression. We define $outputs(P)$, the *number of outputs in* P, inductively as follows:

$$\begin{array}{lcl}
outputs(\mathbf{0}) & = & 0 \\
outputs(\nu_{c_{p(|n|)}}.(Q)) & = & outputs(Q) \\
outputs(c_{p(|n|)}(x).(Q)) & = & outputs(Q) \\
outputs(c_{p(|n|)}\langle T\rangle) & = & 1 \\
outputs([T_1 = T_2].(Q)) & = & outputs(Q) \\
outputs((Q_1 \mid Q_2)) & = & outputs(Q_1) + outputs(Q_2) \\
outputs(!_{\gamma(|n|)}.(Q)) & = & \gamma(|n|) \cdot outputs(Q)
\end{array}$$

It is clear from the definition that $outputs(P)$ is a polynomial in $|n|$ that is positive for all input values.

The following lemma is straightforward.

Lemma 4.3. *Let P be a closed process. Then, for all n and schedulers S, during any evaluation of P, at most $inputs(P)$ communication steps occur.*

Corollary 4.4. *Let P be a closed process. Then, for all n and schedulers S, during any evaluation of P, at most $inputs(P)$ macro-steps occur.*

4.2 A Bound on Outer-Evaluation Time

Definition 4.5. We have a natural number $\mathcal{L}_T$, the *length* of T, associated with each term T.

If T is a term, $\mathcal{L}_T$ is the product of a constant and the syntactic length of T, *i.e.*, $\mathcal{L}_T = c \cdot length(T)$ where c is a constant. More details about $\mathcal{L}_T$ can be found in the proof of lemma 4.10.

Definition 4.6. Let P be a process expression and n be the value for the security parameter. We define a polynomial $normlength(P)$, the *length of* P, inductively as follows:

$$
\begin{array}{lcl}
normlength(\mathbf{0}) & = & 1 \\
normlength(\nu_{c_{p(|n|)}}.(Q)) & = & c + normlength(Q) \\
normlength(c_{p(|n|)}(x).(Q)) & = & c + normlength(Q) \\
normlength(c_{p(|n|)}\langle T \rangle) & = & c + \mathcal{L}_T \\
normlength([T_1 = T_2].(Q)) & = & c + \mathcal{L}_{T_1} + \mathcal{L}_{T_2} + normlength(Q) \\
normlength((Q_1 \mid Q_2)) & = & c + normlength(Q_1) + normlength(Q_2) \\
normlength(!_{\gamma(|n|)}.(Q)) & = & \gamma(|n|) \cdot c \cdot normlength(Q)
\end{array}
$$

where c is a constant and $\mathcal{L}_T$ is the length of the term T.

Since the length of a term does not depend on n (*i.e.,* it is a constant for our purposes), it is clear from the definition that $normlength(P)$ is polynomial in $|n|$. Furthermore, $normlength(P)$ is positive for all n.

Definition 4.7. Let P be some process. Let $\mathbf{p}$ be the set of bandwidth polynomials that appear in P. Note that $\mathbf{p}$ is a finite (polynomial in $|n|$ big) set. For each $i \in \mathbb{N}$, let $\mathbf{a}$ be the set of the coefficients of the x_i terms in each polynomial in $\mathbf{p}$. Notice that $\mathbf{a}$ is finite since $\mathbf{p}$ is finite. For each $i \in \mathbb{N}$ we define the term a_i as the maximum term in $\mathbf{a}$. Since $\mathbf{a}$ is finite, the greatest coefficient is well-defined. We then define the polynomial $\chi(x)$ as $\sum_{i=0}^{\infty} a_i x^i$. Notice that $\forall b \in \mathbb{N}. \forall p(x) \in \mathbf{p} : \chi(b) \geq p(b)$.

Let P be a process and let $\chi(x)$ be defined as per defn. 4.7. We now specify a Turing machine that will pad out P. The Turing machine initially rewrites each variable x in P so that it takes up $\chi(|n|)$ spaces on the input tape. It does so by inserting $\chi(|n|) - 1$ blank symbols before the variable so that x becomes the string $\Delta \cdots \Delta\ x$. Since the length of P is a polynomial in the size of n there are at most a polynomial in the size of n variables each of which have a polynomial in the size of n number of blank symbols inserted before it. Thus, it is easy to see that padding out the variables of P can be done by the Turing machine in at most a polynomial in the size of n number of steps. What this padding does is to ensure that substituting a value for x can be done without increasing the length of a padded process P (since $\chi(|n|)$ exceeds the size of any bandwidth polynomial and since we only need to write down at most a bandwidth polynomial number of bits when substituting a value for a variable, we will always have plenty of space).

During the evaluation of P there can be at most $inputs(P)$ communication steps (lemma 4.3). Let T be an arbitrary term in P. Each communication step during the evaluation of P can hide T behind one level of λ-abstractions; at the end of evaluation, in the worst case, T will be hidden behind $inputs(P)$ λ-abstractions. The increase in the length of the term T caused by a single λ-abstraction is a constant—T becomes $(\lambda x.T)a$. The symbols added to T are 'λ', 'x', '.', '(', ')' and 'a'—recall that a is considered to be $\chi(|n|)$ bits long. Thus, if we add $inputs(P) \cdot c \cdot \chi(|n|)$ blank spaces before each term in P, we will have enough space around each term to ensure that we can account for the maximal increase in the size of terms incurred during evaluation due to the creation of λ-abstractions during communication steps. Again, this padding can be done in a polynomial in the size of n amount of time since there are at most a polynomial in the size of n number of terms and $inputs(P)$ is a polynomial in the size of n.

Hence, padding P can be done in a polynomial in the size of n amount of time by a Turing machine.

Definition 4.8. Let P be a process expression and n be the value for the security parameter. We define a polynomial $length(P)$, the *padded length of* P, inductively as follows:

$$\begin{array}{lcl} length(\mathbf{0}) & = & 1 \\ length(\nu_{c_{p(|n|)}}.(Q)) & = & c + length(Q) \\ length(c_{p(|n|)}(x).(Q)) & = & c + \chi(|n|) + length(Q) \\ length(c_{p(|n|)}\langle T\rangle) & = & c_1 + \mathcal{L}_T + inputs(P) \cdot c_2 \cdot \chi(|n|) \\ length([T_1 = T_2].(Q)) & = & c + (d_1 + \mathcal{L}_{T_1} + inputs(P) \cdot d_2 \cdot \chi(|n|)) \\ & & + (e_1 + \mathcal{L}_{T_2} + inputs(P) \cdot e_2 \cdot \chi(|n|)) + length(Q) \end{array}$$

$$\begin{array}{lcl} length((Q_1 \mid Q_2)) & = & c + length(Q_1) + length(Q_2) \\ length(!_{\gamma(|n|)}.(Q)) & = & \gamma(|n|) \cdot c \cdot length(Q) \end{array}$$

where c, c_1, c_2, d_1, d_2, e_1, e_2 are constants and $\mathcal{L}_T$ is the length of the term T.

Since the length of a term does not depend on n (*i.e.,* it is a constant for our purposes), it is clear from the definition that $length(P)$ is polynomial in $|n|$. Furthermore, $length(P)$ is positive for all n.

The rationale behind the construction of $length(P)$ should be clear given the discussion of padding on page 12. Essentially, for technical reasons, we assume that a variable has length $\chi(|n|)$ (see lemma 4.10 where $\chi(x)$ is a polynomial bigger everywhere than any bandwidth polynomial that appears in P. This allows us to substitute values for variables without pushing symbols around since the variables leave enough space for any conceivable value. Additionally, each term leaves enough blank space for the λ-abstractions created by communication steps (recall that at most $inputs(P)$ such communication steps can occur during evaluation).

In what follows, we shall mean by the phrase "length of P" the padded length of P and not the unpadded (normal) length of P unless otherwise explicitly stated.

Let us observe that the length does not increase as a result of macro-steps.

Lemma 4.9. *Let P be a closed process and R a process such that R is the result of performing some number of macro-steps on P. Then for all n, $length(R) \leq length(P)$.*

We now show that any outer-evaluation step may be carried out on a probabilistic polynomial-time Turing machine. We rely on the assignment of probabilistic polynomial-time Turing machines M_T to terms T discussed in section 2.3.1.

Lemma 4.10. *Let $T_1, \ldots, T_m$ be terms and let $M_{T_1}, \ldots, M_{T_m}$ be their associated probabilistic polynomial-time Turing machines. Let P be a closed process containing terms $(\lambda x_1 \cdots \lambda x_r.T_j)\ a_1 \cdots a_r$, where $1 \leq j \leq m$. From $M_{T_1}, \ldots, M_{T_m}$ one can construct a probabilistic Turing machine that outer-evaluates P in time polynomial in $|n|$.*

Proof. Let us make the following two assumptions. First, we assume that variables have length $\chi(|n|)$. This ensures that when we substitute a value (recall that the length of values are limited by the appropriate bandwidth polynomials due to the truncation in the communication step) for a variable, it is easy to replace the variable with the value, *i.e.,* we do not need to "push" forward the symbols following the variable in order to make room for the value we want to write down.

Second, we assume that when we evaluate a term, we only write down the $\chi(|n|)$ least significant bits of the resultant value. In principle we allow arbitrarily large values to be written down—it is only during communication that these values are truncated. However, this means that as terms evaluate the process expression we are working on might arbitrarily increase in size. Since the only time a value created by a term evaluation is used occurs when we communicate the value, truncating the value at creation does not change anything. It is true that values generated by term evaluations are used in the match—however we do not write down these values; we simply write down either $\mathbf{0}$ or the result of outer-evaluating the process expression bound to the match.

The result of these two assumptions is that each outer-evaluation either does not change the length of the process or decreases the length of the process (by replacing inputs and outputs with $\mathbf{0}$s and evaluating matches).

If we pad P prior to outer-evaluating it we can guarantee these assumptions. Furthermore the padding can be done, as we have seen, in a polynomial in size n amount of time.

We will now construct the desired probabilistic Turing machine, pTM. Essentially, the pTM must evaluate each exposed term and each exposed match in P (it is easy to determine if a term or match is exposed—every time an input operator, say $c_{p(|n|)}(x).P$, is encountered, we consider every subexpression of P as being not exposed). Since we know that P will only contain terms that are substitution instances of terms from the set $\{T_1, \ldots, T_m\}$, we can associate each term U_i from P with an algorithm $M_{T_{i'}}$ where $T_{i'}$ is the term of which U_i is a substitution instance. By theorem 2.9 we have that $M_{T_{i'}}$ computes, in polynomial time, the value to which $T_{i'}$ evaluates on some input. Since U_i is a substitution instance of $T_{i'}$ we have that $U_i \equiv (\lambda x_1 \cdots \lambda x_r.T_{i'})\ a_1 \cdots a_r$. Hence, we can compute the value of U_i by evaluating $M_{T_{i'}}$ at $a_1, \ldots, a_r$ and n.

So we define our pTM so that when it encounters a substitution instance of one of those m terms, it invokes the appropriate algorithm (in the case that it encountered $U_i \equiv (\lambda x_1 \cdots \lambda x_i.T_{i'})\ a_1 \ldots a_i$, it invokes $M_{T_{i'}}$ at $a_1, \ldots, a_1$ and n). Recall that each communication step creates λ-substitution instances of terms, *i.e.,* the substitution of values into terms is delayed until term-evaluation time. So, our pTM evaluates each exposed term it encounters by evaluating the associated algorithm at the values specified by the substitution instance. Evaluating a match simply involves evaluating the two terms and writing either the $\mathbf{0}$ or outer-evaluating the bound process.

How much does this cost in the sense of time complexity? The pTM "touches" each syntactic element of the process expression P. Whenever it hits a term $(\lambda x_1 \cdots \lambda x_i.T)\ a_1 \cdots a_i$, it evaluates T by evaluating the algorithm M_T at $a_1, \ldots, a_i$. This takes a polynomial $q_T(|a_1|, \ldots, |a_i|, |n|)$ time. So, the cost of outer-evaluating P on a pTM is a function of both the length of P and the costs of evaluating each exposed term that the pTM encounters. The length of P is given inductively in defn. 4.8 as $length(P)$. Assuming that there are terms $U_1, \ldots, U_s$ in P out of which the first i are exposed, the cost of outer-evaluating P is:

$$t_1(P) = c \cdot length(P) + \sum_{j=1}^{i} \left(\mathcal{L}_{U_j} + q_{T_j}(|a_1|, \ldots, |a_{\alpha(U_j)}|, |n|) \right) \tag{4.1a}$$

where $U_x \equiv (\lambda x_1 \cdots \lambda x_b.T_x)\ a_1 \cdots a_b$ and $\alpha(U_x) = b$. The $\mathcal{L}_{U_x}$ term accounts for the cost of actually determining the $\alpha(U_x)$ arguments to T_x. Finally, the term $c \cdot length(P)$ accounts for the cost of parsing a padded P to determine whether a term or an expression is exposed or not (note that the expansion in length caused by padding P is accounted for in the definition of $length(P)$).

We observe that any argument to T_x is at most $\chi(|n|)$ bits long. Hence $t_1(P) \leq t_2(P)$ where $t_2(P)$ is:

$$c \cdot length(P) + \sum_{j=1}^{i} \left(\mathcal{L}_{U_j} + q_{T_j}(\overbrace{\chi(|n|), \ldots, \chi(|n|)}^{\alpha(U_j) \text{ times}}, |n|) \right) \tag{4.1b}$$

But, q_{T_j} then becomes just a function of $\chi(|n|)$ and $|n|$. Since $\mathcal{L}_{U_j}$ does not depend on n, we can rewrite $t_2(P)$ as:

$$\tau(P) = c \cdot length(P) + \sum_{j=1}^{i} \left(q'_{U_j}(\chi(|n|), |n|) \right) \tag{4.1c}$$

where $q'_{U_j}(\chi(|n|), |n|) = \mathcal{L}_{U_j} + q_{T_j}(\overbrace{\chi(|n|), \ldots, \chi(|n|)}^{\alpha(U_j) \text{ times}}, |n|)$.

Thus, clearly $\tau(P)$ is a polynomial in $|n|$ (as $length(P)$ is polynomial in $|n|$) that bounds the amount of time the pTM takes to outer-evaluate P. □

¿From probabilistic Turing machines $M_{T_1}, \ldots, M_{T_m}$ we have constructed a probabilistic Turing machine that outer-evaluates P for all values of n. So, we will say that the pTM is *uniformly constructed* from the pTMs $M_{T_1}, \ldots, M_{T_m}$.

Definition 4.11. We define $\kappa(P)$, the cost of outer-evaluating a closed process expression P, as:

$$\kappa(P) = c \cdot length(P) + \sum_{j=1}^{s} \left(q'_{T_j}(\chi(|n|), |n|) \right)$$

where the process P has the terms $T_1, \ldots, T_s$ and q'_{T_j} is defined as for equation (4.1c) and c is a constant that accounts for the cost of parsing a padded P to determine exposed expressions and terms.

Whereas $\tau(P)$ is the time needed to outer-evaluate just the exposed terms and matches in P, $\kappa(P)$ is the cost of outer-evaluating every term and match in P. The following lemma is immediate.

Lemma 4.12. *Let P be a closed process. Then, $\tau(P) \leq \kappa(P)$.*

It may also be shown that macro-steps do not increase the cost of outer-evaluation.

Lemma 4.13. *Let P be a closed process expression and S a scheduler. Let R be a process such that R is obtained by performing a finite number of macro-steps on P. Then for all n and k, $\kappa(R) \leq \kappa(P)$.*

Thus we also obtain

Lemma 4.14. *Let P be a closed process. Let R be a process such that P evaluates to R in at most m macro-steps. Then $\tau(R) \leq \kappa(P)$.*

4.3 A Bound on the Communication Step

Lemma 4.15. *Let P be a closed process and π a communication triple. Then there exists a pTM such that for all P, π, n and k, the pTM performs the communication step indicated by π on P in time polynomial in $|n|$.*

Proof. We begin by padding P. This allows us to assume, as we did in the proof of lemma 4.10, that variables are $\chi(|n|)$ bits long. This means that we can easily substitute values for variables by simply overwriting the variable.

In order to perform the communication step, we make one pass to remove the input and output specified in π. Specifically, we rewrite the subexpression $c_{p(|n|)}(x).Q$ specified as the input in π with $\Delta\Delta\Delta\Delta\Delta Q$ (where Δ is a special character that the pTM ignores—this allows us to delete subexpressions of P without having to push symbols around) and we rewrite the subexpression $c_{p(|n|)}\langle a\rangle$ specified as the output in π with $\Delta\Delta\Delta\mathbf{0}$. So, the pTM performs a pass, prior to performing the substitution indicated by the communication step, to eliminate the input and output process expressions.

We then make another pass over the process replacing all instances of x with a. Since we assume that all channels are α-renamed apart and that both a and x are at most $\chi(|n|)$ bits long, we can do this substitution without any real effort except when we create a λ-substitution instance out of a term T. When this happens, we need to insert symbols. However, each communication step adds one level of λ-abstraction; so, when we write the process down we leave sufficient room around each term T, *i.e.,* we add some fixed amount of blank space after each term. Since the notation for a single λ-abstraction is constant sized and the value to apply to that λ-abstraction is at most $\chi(|n|)$ bits long, a fixed amount of space per term will suffice. We added this extra space to each term during the initial pass that padded out P. Essentially, we are adding a polynomial in $|n|$ amount of padding to the process—the length of the expression is a polynomial in $|n|$ and an upper-bound on the number of terms in the process.

Then, we can easily perform the needed rewriting necessitated by the communication step without moving symbols around. Thus, a single pass over the process should suffice to perform the communication step.

So, the time needed to perform the communication step is just $\sigma(P) = c_0 \cdot length(P)$ where c_0 is a constant that accounts for the multiple passes (one to eliminate the input and output expressions and another to perform the substitutions indicated by the communication step) and $length(P)$ accounts for the increased length caused by padding out P. Clearly, $\sigma(P)$ is a polynomial in $|n|$ since $length(P)$ is such a polynomial. ☐

Definition 4.16. Let P be a process expression. We notice that if there are $inputs(P)$ inputs and $outputs(P)$ outputs in P, then there are at most $inputs(P) \cdot outputs(P)$ communication steps possible. Let c_i be the constant factor that accounts for the multiples passes incurred during the evaluation of the ith communication step (meaning, the ith communication out of the set of $inputs(P) \cdot outputs(P)$ communication steps; not the communication taken at the ith evaluation step).

We define $\varphi(P)$, the cost of performing a substitution given by a communication triple π on the closed process expression P, as:

$$\varphi(P) = c_0 \cdot length(P) + c_1 \cdot length(P) + \cdots + c_k \cdot length(P)$$

where $k = inputs(P) \cdot outputs(P)$.

$\sigma(P)$ is the time needed to perform the substitution specified by π. However, $\varphi(P)$ is meant to bound the cost of any substitution that might possibly occur in the course of evaluating P. A process has $inputs(P)$ inputs and $outputs(P)$ outputs. Since an arbitrary substitution consists of one input expression and one output expression, there are at most $inputs(P) \cdot outputs(P)$ possible communications that can occur. At any communication step (by lemma 4.4, there are at most $inputs(P)$ such steps), only one potential communication occurs. Hence φ is an upper bound on the time needed to perform any communication that might occur in the evaluation of P.

It is clear that

Lemma 4.17. *Let P be a closed process. Then, $\sigma(P) \leq \varphi(P)$.*

It follows from lemmas 4.9 and 4.17 that

Lemma 4.18. *Let P be a closed process expression and S a scheduler. Let R be a process such that R is obtained by a finite number of macro-steps from P. Then for all n, $\sigma(R) \leq \varphi(P)$.*

4.4 A Bound on Evaluation Time

We now use the bounds obtained in the previous three subsections to obtain a polynomial bound on total evaluation time.

Definition 4.19. Let P be a process expression. We define an *evaluation context for P* as a context $C[\ ]$ of the form:

$$c_{1p_1(|n|)}(x_1)\dots c_{ip_i(|n|)}(x_i).[\ ] \mid c_{1p_1(|n|)}\langle a_1\rangle \mid \cdots \mid c_{tp_t(|n|)}\langle a_t\rangle$$

where $t = i + inputs(P)$, there are no free variables in $C[P]$, $a_j \in \mathbb{N}$, and each $x_1, \cdots, x_i$ either appears free in P or does not appear at all in P.

We will sometimes specify that the evaluation context $C[\ ]$ is an (i,t)-evaluation context, meaning that it has i inputs and t outputs.

If P is a process and $C[\ ]$ is an evaluation context for P, then $C[P]$ is a closed process.

Definition 4.20. Let $P \equiv P_1 \mid \cdots \mid P_m$ be a process such that P is the result of outer-evaluating some process expression Q. If, for each P_i such that $P_i \equiv c_{p(|n|)}(x).P_i'$ there does not exist a P_j with $j \neq i$ such that $P_j \equiv c_{p(|n|)}\langle a\rangle$, we say that P is a *normal form.*

It should be clear from defn. 4.20 that if a process P is a normal form, then P cannot be further evaluated as $E(P) = \emptyset$.

Theorem 4.21. *Let P be a process expression. Then, there exists a polynomial $q(x)$ with all positive coefficients such that for all n, all poly-time schedulers S and minimal evaluation contexts $C[\ ]$ for P, the process $C[P]$ evaluates to a normal form on a probabilistic Turing machine in time at most $q(|n|)$.*

The idea behind this theorem is that given a process P (that is possibly not closed) and an evaluation context $C[\ ]$ that closes P and provides P with all the necessary inputs, $C[P]$ evaluates, on a probabilistic Turing machine, to a normal form in at most polynomial time.

Proof. By lemma 4.3, we have that at most $inputs(C[P])$ (recall that this is a polynomial in $|n|$—see defn. 4.1) communication steps (and hence macro-steps) occur during the evaluation. Thus, all we need to do is determine the time it takes to perform a macro-step.

We note that the second outer-evaluation of a macro-step composes with the first outer-evaluation of the next macro-step. Since outer-evaluation is idempotent (lemma 2.19), neither the probability distribution nor the resultant process changes as a consequence of the double outer-evaluation. All that happens is that we take a little extra time.

The time to outer-evaluate any process that can be obtained by a reduction from $C[P]$ on a Turing machine is given by lemma 4.14 as a polynomial in $|n|$—call this $q_o(|n|)$. Recall that outer-evaluation is performed by making a pass over the process expression and replacing each exposed term that has no free variables with a number (truncated so that its size doesn't exceed the band-width of the channel). In the case of matches, we do not write down the value of the two terms being compared. Instead we outer-evaluate the bound process or write down $\mathbf{0}$. Since the length of $C[P]$ is polynomial in $|n|$ outer-evaluation takes time polynomial in $|n|$.

The time to perform any communication step that might arise in the evaluation of $C[P]$ is given by lemma 4.18 as a polynomial, q_c, in $|n|$. Recall that a communication step that substitutes a for x is performed by making a pass over the process expression and replacing each instance of the variable x with a. Since the length of $C[P]$ is polynomial in $|n|$ communication takes time polynomial in $|n|$.

However, there is an additional cost incurred in actually selecting the communication step. It requires a single pass over the process expression to build $S(C[P])$ and then time proportional in $|S(C[P])|$ to build $E(C[P])$. But there are at most $inputs(C[P])$ input expressions and $outputs(C[P])$ output expressions in $C[P]$. Consequently $|S(C[P])|$ is at most $inputs(C[P]) \cdot outputs(C[P])$ which is a polynomial in $|n|$. Consequently, generating $E(C[P])$ from $C[P]$ takes polynomial in $|n|$ time.

Furthermore, the scheduler is defined to be poly-time, *i.e.,* given the eligible process set $E(C[P])$, the scheduler takes a polynomial in $|E(C[P])| = |S(C[P])|^2$ amount of time to choose a communication step. But this is just a polynomial in $|n|$. Hence, in order to generate $E(C[P])$ and make a choice we require a polynomial, q_d, in $|n|$ amount of time. Since macro-steps do not increase the length of a process expression, q_d is an upper-bound on the amount of time required to schedule a communication when evaluating R where R is a process expression obtained from $C[P]$ via some number of macro-steps.

Hence the time needed for evaluation q is given by:

$$inputs(C[P]) \cdot \Big(q_o(|n|) + q_d(|n|) + q_c(|n|) + q_o(|n|)\Big) \tag{4.2}$$

which is just a polynomial in $|n|$. □

5 Equivalence

Definition 5.1. We define an *observation* to be a test on a particular public channel for a particular natural number. More precisely, we will define Obs to be the set of all possible observations, *i.e.,* the set of pairs $\langle i, c_{p(|n|)}\rangle$ where $i \in [0..2^{p(|n|)} - 1]$ is a natural number and $c_{p(|n|)}$ is a public channel. If, during an evaluation of process expression P, the scheduler selects the communication triple $\langle \overline{c_{p(|n|)}}\langle i\rangle, c_{p(|n|)}(x).P', Q_{\overline{c_{p(|n|)}}\langle i\rangle, c_{p(|n|)}(x).P'}\rangle$ we will say that the observable $\langle i, c_{p(|n|)}\rangle \in Obs$ was observed and write $P \rightsquigarrow \langle i, c_{p(|n|)}\rangle$.

Definition 5.2. Let Δ be the set of expressions generated by the following sub-grammar of the one which produces contexts (see defn. 2.21):

$$\begin{array}{rcl} C[\ ] & ::= & [\ ]_i \\ & & c_{p(|n|)}(x).(C[\ ]) \\ & & P \\ & & [T = T].(C[\ ]) \\ & & (C[\ ] \mid C[\ ]) \\ & & !_{\gamma(|n|)}.(C[\ ]) \end{array}$$

where $i \in \mathbb{N}$.

An *adversarial context* is a process expression that is a member of Δ—Δ is just the subset of the set of contexts (Γ) where a hole doesn't appear in the scope of a ν-operator. As for a standard context, the "holes" are numbered uniquely.

Definition 5.3. A process P may contain the security parameter n (section 2.3). We will write P_m to signify that the parameter n is assigned the natural number m. A *process family* $\mathcal{P}$ is the set $\langle P_i | \, i \in \mathbb{N} \rangle$. Since contexts may contain the process parameter n, we can define the *context family* $\mathcal{C}[\ ]$ and the *adversarial context family* $\mathcal{C}[\ ]$ analogously.

We evaluate a process family by picking a value for n and then evaluating the indicated member of the process family. In this manner we can extend all the concepts regarding process expressions and contexts to process and context families.

Definition 5.4. We define $\mathbf{P}$ to be the set of all process families, $\mathbf{C}$ to be the set of all context families and $\mathbf{A}$ to be the set of all adversarial context families.

5.1 Observational Equivalence

Definition 5.5. Let $\mathcal{P}$ be a process family. We will say that the channel name $c_{q(|n|)}$ *appears in* $\mathcal{P}$ just when there are inputs and outputs on $c_{q(|n|)}$ that are subexpressions of $\mathcal{P}$.

Definition 5.6. Let $\mathcal{P}$ and $\mathcal{Q}$ be two process families. Let $\mathbf{q}$ be the set of polynomials $q(x)$ such that $\forall y : q(y) > 0$. We will say that $\mathcal{P}$ and $\mathcal{Q}$ are *observationally equivalent* if:

$$\forall q(x) \in \mathbf{q}. \forall \mathcal{C}[\ ] \in \mathbf{A}. \forall o \in Obs. \exists n_o. \forall n > n_o :$$
$$\Big| \operatorname{Prob}\big[C_n[P_n] \rightsquigarrow o\big] - \operatorname{Prob}\big[C_n[Q_n] \rightsquigarrow o\big] \Big| \leq \frac{1}{q(n)}$$

If this is so, we will write that $\mathcal{P} \cong \mathcal{Q}$.

It is our goal to define adversaries to a protocol as adversarial contexts. Then, we will be able to prove security properties by stating observational equivalences.

The idea here is that if two process families are the same, then no adversarial context (*i.e.*, attacker) can distinguish between the two with significant probability once the security parameter gets large enough—the probability of one process family producing an observable is indistinguishable (to within an arbitrary polynomial factor) from the probability of the other process family producing that same observable.

However, if we forced observational equivalence to hold under all contexts, then we are essentially saying that adversaries can attack a protocol by changing the way a protocol executes (by specifying the scope of a private channel and, thereby, being able to read values transmitted on private channels). However, such attacks do not seem appropriate given our model of adversary capabilities. Hence, we discount such contexts from our definition of observational equivalence.

We can only guarantee such a property in the case that n is large enough as most any security system can be defeated by brute-force searches if the search-space is small enough. So, even though a context (*i.e.,* a brute force search over, say, keys) may distinguish two process families (that is break one protocol and not the other), once we up the security parameter sufficiently, that context will get defeated.

Proposition 5.7. $\cong$ *is an equivalence relation.*

Because the definition of observational equivalence involves all adversarial contexts (definition 5.6), it is plain that

Property 5.8 (**SUBST-EQ**)**.** $\mathcal{P} \cong \mathcal{Q} \iff \forall \mathcal{C}[\ ] \in \mathbf{A} : \mathcal{C}[\mathcal{P}] \cong \mathcal{C}[\mathcal{Q}]$.

We shall refer to the property SUBST-EQ as the rule of *substitutive equivalence.*

A series of properties follow immediately from SUBST-EQ.

Property 5.9 (**REPL-EQ**)**.** *If* $\mathcal{P} \cong \mathcal{Q}$ *then* $!_{r(|n|)}.\mathcal{P} \cong\ !_{r(|n|)}.\mathcal{Q}$ *where* r *is a polynomial such that* $\forall a \in \mathbb{N} : r(a) \geq 0$.

Property 5.10 (**OUTPUTS-EQ**)**.** *If* $\mathcal{P} \cong \mathcal{Q}$ *then:*

$$c_{1p(|n|)}\langle m_1 \rangle \mid \cdots \mid c_{ip(|n|)}\langle m_i \rangle \mid \mathcal{P} \cong c_{1p(|n|)}\langle m_1 \rangle \mid \cdots \mid c_{ip(|n|)}\langle m_i \rangle \mid \mathcal{Q}$$

Property 5.11 (**MATCH-EQ**)**.** *If* $\mathcal{P} \cong \mathcal{Q}$ *then* $[T_1 = T_2].\mathcal{P} \cong [T_1 = T_2].\mathcal{Q}$.

Property 5.12 (**REST-SUBST-EQ**)**.** *Let* $\mathbf{A} \subseteq \{\mathcal{R} | \mathcal{R}$ *is a process family*$\}$ *be a set such that* $\mathbf{0} \in \mathbf{A}$. *Let* $\mathcal{P}$ *and* $\mathcal{Q}$ *be process families. Then,* $\forall \mathcal{A} \in \mathbf{A}. (\mathcal{A} \mid \mathcal{P} \cong \mathcal{A} \mid \mathcal{Q}) \iff \mathcal{P} \cong \mathcal{Q}$.

Property 5.13 (**PARALLEL-COMP**)**.** *If* $\mathcal{P}_1 \cong \mathcal{P}_2$ *and* $\mathcal{Q}_1 \cong \mathcal{Q}_2$*, then* $\mathcal{P}_1 \mid \mathcal{Q}_1 \cong \mathcal{P}_2 \mid \mathcal{Q}_2$.

Another basic property of observational equivalence involves the relationship among the probabilistic polynomial-time computable functions, the terms of our calculus, and the polynomial-time oracle Turing machines, described in section 2.3.1.

Property 5.14 (**EQ-DISTRIB**)**.** *Let* f *and* g *be probabilistic polynomial-time computable functions such that* $range(f) = range(g) = X \subseteq \mathbb{N}$ *and both* f *and* g *induce the same distribution on* X. *Let* T_f *be the term such that* M_{T_f} *computes* f *and* T_g *be the term such that* M_{T_g} *computes* g. *Then there exists a polynomial* q *such that* $c_{q(|n|)}\langle T_f \rangle \cong c_{q(|n|)}\langle T_g \rangle$.

6 Cryptographic examples

Let us show that our asymptotic notion of observational equivalence between probabilistic polynomial-time processes coincides with the traditional notion of *indistinguishability by polynomial-time statistical tests*, a standard way of characterizing cryptographically strong pseudorandom number generators [Yao82, Gol00, Gol99, Lub96, Gen00].

6.0.1 Pseudorandom Number Generators

We begin by recalling several standard notions from cryptographic literature [Yao82, Gol00, Gol99, Lub96, Gen00].

Definition 6.1 (function ensemble). A *function ensemble* f is an indexed family of functions $\{f_n : A_n \to B_n\}_{n\in\mathbb{N}}$.

The reader might wish to review defns. 2.4, 2.5, 2.6, 2.7 and 2.8 for details before proceeding.

Definition 6.2. A function ensemble $f : A_n \to B_n$ is *uniform* if there exists a single Turing machine M that computes f for all values of n, *i.e.,* $M(n, x) = f_n(x)$.

Definition 6.3. A uniform function ensemble $f : A_n \to B_n$ is *poly-time* if there exists a polynomial q and a single Turing machine M such that $M(n, x)$ computes $f_n(x)$ in time at most $q(|n|, |x|)$.

Definition 6.4. A uniform function ensemble $f : A_n \times B_n \to [0, 1]$ is *probabilistic poly-time* if there exists a single probabilistic poly-time Turing machine M such that $M(n)$ computes f_n.

Definition 6.5. A *poly-time statistical test* $\mathcal{A} : \{0,1\}^{m(|n|)} \times \{0,1\} \to [0,1]$ is a $\{0,1\}$-valued probabilistic poly-time function ensemble.

Definition 6.6 (pseudorandom number generator). Let $q(x)$ be a positive polynomial. A *pseudorandom number generator* (PRNG) is a uniform polynomial time function ensemble $f : \{0,1\}^{k(|n|)} \to \{0,1\}^{l(|n|)}$ such that for all poly-time statistical tests $\mathcal{A}$:

$$\forall q(x).\exists n_o.\forall n > n_o : \big|\ \mathrm{Prob}_{s\in_R\{0,1\}^{k(|n|)}}\big[\mathcal{A}(f_n(s)) = \text{“1”}\big] - \mathrm{Prob}_{r\in_R\{0,1\}^{l(|n|)}}\big[\mathcal{A}(r) = \text{“1”}\big]\big| \le \frac{1}{q(n)}$$

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

The reader might wish to review the definition of a probabilistic function (defn. 2.1) and composition of probabilistic functions (defn. 2.2) before proceeding.

Definition 6.7. Let $f : \{\epsilon\} \times \{0,1\}^{q(|n|)} \to [0,1]$ be a probabilistic poly-time function ensemble. [1] Let T_f be a term such that M_{T_f} computes f. Then, we say that $c_{q(|n|)}\langle T_f\rangle$ is the *characteristic process family for f with respect to channel $c_{q(|n|)}$.*

Let $f : \{\epsilon\} \times \{0,1\}^{q(|n|)} \to [0,1]$ be a probabilistic poly-time function ensemble and let $\mathcal{P}_f \equiv c_{q(|n|)}\langle T_f\rangle$ be its characteristic process family with respect to channel $c_{q(|n|)}$. Then it is easy to see that $\forall i \in \mathbb{N} : \mathrm{Prob}\big[f(\epsilon) = i\big] = \mathrm{Prob}\big[((c_{q(|n|)}(x).\mathbf{0}) \mid \mathcal{P}_f) \rightsquigarrow \langle i, c_{q(|n|)}\rangle\big]$. This is the probability that $\mathcal{P}_f$ is willing to communicate i on channel $c_{q(|n|)}$.

Lemma 6.8. *Let $\mathcal{A} : \{0,1\}^{m(|n|)} \times \{0,1\} \to [0,1]$ be a poly-time statistical test. Let $f : \{\epsilon\} \times \{0,1\}^{m(|n|)} \to [0,1]$ be a probabilistic poly-time function ensemble and let $\mathcal{P}_f$ be its characteristic process family with respect to channel $c_{m(|n|)}$. Then one can construct an adversarial context family $\mathcal{C}_{\mathcal{A}}[\ \ ]$ using a new channel d_1, such that for any fixed value for n, it is the case that $\forall i \in \mathbb{N}$: $Prob\big[(\mathcal{A} \circ f)(\epsilon) = i\big] = Prob\big[\mathcal{C}_{\mathcal{A}}[\mathcal{P}_f] \rightsquigarrow \langle i, d_1\rangle\big]$.*

[1] ϵ is a dummy symbol. We wish to define a probabilistic function ensemble that takes no input and produces a probabilistic output. However $\{\} \times A = \{\}$ where A is any set. Hence, we have function take as input a single dummy value ϵ.

Proof. If $\mathcal{A}$ is a poly-time statistical test then using theorem 2.10 we can construct the context family $\mathcal{C}_{\mathcal{A}}[\ \ ] \equiv ((d_1(x).\mathbf{0}) \mid (c_{m(|n|)}(x).d_1\langle T_{\mathcal{A}}\rangle \mid [\ \ ])$. Note that $\mathcal{C}_{\mathcal{A}}[\ \ ]$ is an adversarial context family.

By assumption, $\mathcal{P}_f$ is the characteristic process family for f with respect to channel $c_{m(|n|)}$. Furthermore, $T_{\mathcal{A}}$ is produced from $\mathcal{A}$ using theorem 2.10. Hence, the observables $\langle 0, d_1\rangle$ or $\langle 1, d_1\rangle$ are observed during evaluation of $\mathcal{C}_{\mathcal{A}}[\mathcal{P}_f]$. The probability that $\langle 0, d_1\rangle$ is observed must be the same as the probability that $(\mathcal{A} \circ f)(\epsilon) = 0$. Similarly, the probability that $\langle 1, d_1\rangle$ is observed must be the same as the probability that $(\mathcal{A} \circ f)(\epsilon) = 1$. □

We will say that an adversarial context family so constructed is a *poly-time attacker*.

Definition 6.9. Let $\mathcal{P}$ be a process family and o an observable. We will say that $f : \{\epsilon\} \times \{0,1\} \to [0,1]$ is an *indicator for $\mathcal{P}$ with respect to o* when $\mathrm{Prob}\big[\mathcal{P} \rightsquigarrow o\big] = \mathrm{Prob}\big[f(\epsilon) = 1\big]$ and $\mathrm{Prob}\big[\mathcal{P} \not\rightsquigarrow o\big] = \mathrm{Prob}\big[f(\epsilon) = 0\big]$.

Lemma 6.10. *Let $\mathcal{C}[\ \ ]$ be an adversarial context family and let o be an observable. Let $f : \{\epsilon\} \times \{0,1\}^{m(|n|)} \to [0,1]$ be any probabilistic poly-time function ensemble and let $\mathcal{P}_f$ be its characteristic process family. Then one can specify a poly-time statistical test t from the pair $\langle \mathcal{C}[\ \ ], o\rangle$ such that $t \circ f$ is an indicator for $\mathcal{C}[\mathcal{P}_f]$ with respect to the observable o.*

Proof. We construct t as follows. We compute $t \circ f$ by evaluating $\mathcal{C}[d_{q(|n|)}\langle T_f\rangle]$ (where M_{T_f} computes f) and returning 1 if the observable o was generated and 0 otherwise. It is easy to see that $\mathrm{Prob}\big[(t \circ f)(\epsilon) = 1\big] = \mathrm{Prob}\big[\mathcal{C}[d_{q(|n|)}\langle T_f\rangle] \rightsquigarrow o\big]$ and that $\mathrm{Prob}\big[(t \circ f)(\epsilon) = 0\big] = \mathrm{Prob}\big[\mathcal{C}[d_{q(|n|)}\langle T_f\rangle] \not\rightsquigarrow o\big]$. □

Clearly, given an adversarial context family and a process family, each potential observable defines a poly-time statistical test.

We can now prove that an algorithm taking short strings to long strings is pseudorandom if and only if the process given by the algorithm, when evaluated on a short random input, is observationally equivalent to the process that returns a long random seed.

Definition 6.11. A function ensemble $f : \{\epsilon\} \times \{0,1\}^{l(|n|)} \to [0,1]$ is *random poly-time* if f with respect to n is a poly-time function such that $\forall x \in \{0,1\}^{l(|n|)} : f(\epsilon, x) = \frac{1}{2^{l(|n|)}}$.

Theorem 6.12. *Let $f' : \{0,1\}^{k(|n|)} \to \{0,1\}^{l(|n|)}$ ($\forall x \in \mathbb{N}.l(x) > k(x)$) be a uniform poly-time function ensemble. Let $r : \{\epsilon\} \times \{0,1\}^{l(|n|)} \to [0,1]$ and $s : \{\epsilon\} \times \{0,1\}^{k(|n|)} \to [0,1]$ be uniform poly-time random function ensembles. Define f as $f' \circ s$. Let $\mathcal{F}$ (resp. $\mathcal{R}$) be the characteristic process family for f (resp. r).*

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

Let M_{T_r} compute r and let M_{T_f} compute f. Then, $\mathcal{F}$ is a process family that, essentially, transmits a random seed generated by s to f' (a candidate PRNG—note that the type of $f' \circ s$ is $\{\epsilon\} \times \{0,1\}^{l(|n|)} \to [0,1]$) via function composition, and then transmits the value output by T_f on a public channel. In contrast, $\mathcal{R}$ is a process family that simply transmits the value output by T_r (a function that returns truly random values of the same length as those generated by T_f) on a public channel.

Proof. Assume that $\mathcal{F} \cong \mathcal{R}$ and that, by way of producing a contradiction, f' is not a PRNG. Then, we have that there exists a poly-time statistical test $\mathcal{A}$ that distinguishes between the output of f'

given a random input value (a seed) and the output of a truly random source. That is to say:

$$\exists q(x).\forall n_o.\exists n > n_o : \Big|\ \mathrm{Prob}_{x \in_R \{0,1\}^{k(|n|)}}\big[\mathcal{A}(f'(x)) = \text{“1”}\big] - \mathrm{Prob}_{y \in_R \{0,1\}^{l(|n|)}}\big[\mathcal{A}(y) = \text{“1”}\big]\Big| > \frac{1}{q(n)}$$

But we have that $\forall j \in \mathbb{N} : \mathrm{Prob}_{x \in_R \{0,1\}^{k(|n|)}}\big[f'(x) = j\big] = \mathrm{Prob}\big[(f' \circ s)(\epsilon) = j\big]$ and $\forall j \in \mathbb{N}$: $\mathrm{Prob}_{y \in_R \{0,1\}^{l(|n|)}}\big[y = j\big] = \mathrm{Prob}\big[r(\epsilon) = j\big]$. Hence, by lemma 6.8, we can construct a poly-time attacker $\mathcal{C}[\ \ ]$ that distinguishes between $\mathcal{F}$ and $\mathcal{R}$ with the same probability that $\mathcal{A}$ distinguishes between f and r. So $\mathcal{C}[\ \ ]$ will distinguish between $\mathcal{F}$ and $\mathcal{R}$ with probability greater than $\frac{1}{q(n)}$ for some polynomial q (as $\mathcal{A}$ distinguishes between f and r with probability greater than $\frac{1}{q(n)}$), thus producing a contradiction.

Now, assume that f' is a PRNG and that, by way of producing a contradiction, $\mathcal{F} \not\cong \mathcal{R}$. Then we have that for some polynomial p there exists an adversarial context family $\mathcal{D}[\ \ ]$ that distinguishes between the two processes with probability greater than $\frac{1}{p(n)}$ for some polynomial p. Let the distinguishing observation be $\langle i, d_{q(|n|)}\rangle$. We notice that $\mathrm{Prob}\big[\mathcal{D}[\mathcal{F}] \rightsquigarrow \langle j, c_{l(|n|)}\rangle\big] = \mathrm{Prob}\big[f(\epsilon) = j\big]$ where $j \in \mathbb{N}$ and that $\mathrm{Prob}\big[\mathcal{D}[\mathcal{R}] \rightsquigarrow \langle j, c_{l(|n|)}\rangle\big] = \mathrm{Prob}\big[r(\epsilon) = j\big]$. We can then use lemma 6.10 to construct a poly-time statistical test that distinguishes between the output of f and a truly random source with precisely the same probability that $\mathcal{D}[\ \ ]$ distinguishes between $\mathcal{F}$ and $\mathcal{R}$, thereby creating a contradiction. $\square$

Acknowledgements: Thanks to M. Abadi, D. Boneh, R. Canetti, C. Dwork, R. van Glabbeek, A. Jeffrey, S. Kannan, B. Kapron, P. Lincoln, R. Milner, M. Mitchell, M. Naor, P. Panangaden and P. Selinger for helpful discussions and advice on relevant literature.

References

[AF01] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *28th ACM Symposium on Principles of Programming Languages*, pages 104–115, 2001.

[AG98] M. Abadi and A. Gordon. A bisimulation method for cryptographic protocol. In *Proc. ESOP'98, Springer Lecture Notes in Computer Science*, 1998.

[AG99] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 143:1–70, 1999. Expanded version available as SRC Research Report 149 (January 1998).

[AJ01] M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4-th International Symposium on Theoretical Aspects of Computer Software (TACS2001)*, Tohoku University, Sendai, Japan, 2001. Springer LNCS.

[AR00] M. Abadi and P. Rogaway. Reconciling two views of cryptography (The computational soundness of formal encryption). In *IFIP International Conference on Theoretical Computer Science*, Sendai, Japan, 2000. Full paper to appear in *J. of Cryptology.*

[Ata99] M.J. Atallah, editor. *Algorithms and Theory of Computation Handbook*, pages 19–28. CRC Press LLC, 1999.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society, Series A*, 426(1871):233–271, 1989. Also appeared as SRC Research Report 39 and, in a shortened form, in ACM Transactions on Computer Systems 8, 1 (February 1990), 18-36.

[Bel92] S. Bellantoni. *Predicative Recursion and Computational Complexity.* PhD thesis, University of Toronto, 1992.

[Can00] R. Canetti. A unified framework for analyzing security of protocols. Cryptology ePrint Archive: Report 2000/067; see http://eprint.iacr.org/2000/067/, 2000.

[CDL+99] I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *12-th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.

[DDN91] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography (extended abstract). In *Proc. 23rd Annual ACM Symposium on the Theory of Computing*, pages 542–552, 1991.

[DMP01] N.A. Durgin, J.C. Mitchell, and D. Pavlovic. A compositional logic for protocol correctness. In *IEEE Computer Security Foundations Workshop*, 2001.

[DY81] D. Dolev and A. Yao. On the security of public-key protocols. In *Proc. 22nd Annual IEEE Symp. Foundations of Computer Science*, pages 350–357, 1981.

[ElG85] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31:469–472, 1985.

[Gen00] R. Gennaro. An improved pseudo-random generator based on discrete log. In *Proc. CRYPTO 2000*, pages 469–481. Springer LNCS 1880, 2000. Revised version available on www.research.ibm.com/people/r/rosario/.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Computer and System Sciences*, 28:281–308, 1984.

[Gol99] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Springer Verlag, 1999.

[Gol00] O. Goldreich. The Foundations of Cryptography — A Book in Preparation. Available on www.wisdom.weizmann.ac.il/~oded/foc-book.html, 2000.

[Hof99] M. Hofmann. Type systems for polynomial-time computation. Habilitation thesis, Darmstadt, 1999; see www.dcs.ed.ac.uk/home/mxh/papers.html, 1999.

[KMM94] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *J. Cryptology*, 7(2):79–130, 1994.

[LMMS98] P.D. Lincoln, M. Mitchell, J.C. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In M.K. Reiter, editor, *Proc. 5-th ACM Conference on Computer and Communications Security*, pages 112–121, San Francisco, California, 1998. ACM Press.

[LMMS99] P.D. Lincoln, J.C. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security protocols. In J.M. Wing and J. Woodcock and J. Davies, editor, *Formal Methods World Congress, Vol. I*, pages 776–793, Toulouse, France, 1999. Springer LNCS 1708.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In *2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*. Springer-Verlag, 1996.

[LS91] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[Lub96] M. Luby. *Pseudorandomness and Cryptographic Applications.* Princeton Computer Science Notes, Princeton University Press, 1996.

[Mea96] C. Meadows. Analyzing the Needham-Schroeder public-key protocol: a comparison of two approaches. In *Proc. European Symposium On Research In Computer Security.* Springer Verlag, 1996.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[MMS97] J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murφ. In *Proc. IEEE Symp. Security and Privacy*, pages 141–151, 1997.

[MMS98] J.C. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39-th Annual IEEE Symposium on Foundations of Computer Science*, pages 725–733, Palo Alto, California, 1998. IEEE Computer Society Press.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–9, 1978.

[Pau97a] L.C. Paulson. Mechanized proofs for a recursive authentication protocol. In *10th IEEE Computer Security Foundations Workshop*, pages 84–95, 1997.

[Pau97b] L.C. Paulson. Proving properties of security protocols by induction. In *10th IEEE Computer Security Foundations Workshop*, pages 70–83, 1997.

[PW00] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *7-th ACM Conference on Computer and Communications Security, Athens, November 2000*, pages 245–254. ACM Press, 2000. Preliminary version: IBM Research Report RZ 3234 (# 93280) 06/12/00, IBM Research Division, Zürich, June 2000.

[Ros95] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *CSFW VIII*, page 98. IEEE Computer Soc Press, 1995.

[Sch96] S. Schneider. Security properties and CSP. In *IEEE Symp. Security and Privacy*, 1996.

[vGSS95] R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.

[Yao82] A. Yao. Theory and applications of trapdoor functions. In *IEEE Foundations of Computer Science*, pages 80–91, 1982.