A probabilistic poly-time framework for protocol analysis

P. Lincoln — SRI International
J. Mitchell, M. Mitchell — Stanford University
A. Scedrov — University of Pennsylvania

April 3, 1998

Abstract

We develop a framework for analyzing security protocols in which protocol adversaries may be arbitrary probabilistic polynomial-time processes. In this framework, protocols are written in a restricted form of π-calculus and security may expressed as a form or *observational equivalence,* a standard relation from programming language theory that involves quantifying over possible environments that might interact with the protocol. Using an asymptotic notion of probabilistic equivalence, we relate observational equivalence to polynomial-time statistical tests and discuss some example protocols to illustrate the potential strengths of our approach.

1 Introduction

Protocols based on cryptographic primitives are commonly used to protect access to computer systems and to protect transactions over the internet. Two well-known examples are the Kerberos authentication scheme [KNT94, KN93], used to manage encrypted passwords on clusters of interconnected computers, and the Secure Sockets Layer [FKK96], used by internet browsers and servers to carry out secure internet transactions. Over the past decade or two, a variety of methods have developed for analyzing and reasoning about these and similar protocols. The recognized approaches include specialized logics such as BAN logic [BAN89], special-purpose tools designed for cryptographic protocol analysis [KMM94], and theorem proving [Pau97b, Pau97a] and model-checking methods using general purpose tools [Low96, Mea96, MMS97, Ros95, Sch96].

Although there are many differences among these approaches, most current approaches use the same basic model of adversary capabilities. In common models, largely derived from [DY81], 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. At the same time, the set of messages an adversary may use to interfere with a procotol is severely limited. Many approaches use an axiomatic characterization of “intruder knowledge” that only allows an adversary to send messages comprised of data they “know” as the result of overhearing previous messages. This prevents an adversary from sending a message with random data, for example, even though we believe that it would be possible to send randomly chosen messages in practice. The basic problem is that although these 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. Another limitation is that a deterministic or nondeterministic setting does not allow us to analyze probabilistic protocols. Some of the advantages of probabilistic encryption, for example, are enumerated in [GM84].

In this paper, we set the stage for a form of protocol analysis that is programming-language based, yet closer in foundations to the mathematical setting of modern cryptography. The framework is presented here using 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 lift other restrictions on the behavior of an adversary. 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.

Our framework may be viewed as an alternate form of *spi-calculus*, and we have taken a number of ideas from the pioneering work of Abadi and Gordon [AG97]. In particular, our use of a language framework to capture adversary behavior is based on spi-calculus, as well as our use of traditional observational equivalence (also called observational congruence) for analyzing security properties. Intuitively, two systems (such as two protocols) P and Q are observationally equivalent, written $P \simeq Q$, if any program $\mathcal{C}[P]$ containing P has the same observable behavior as the program $\mathcal{C}[Q]$ with Q replacing P. The reason observational equivalence is applicable to security analysis is that it involves quantifying over all possible additional processes (represented by the context $\mathcal{C}[\;]$) that might interact with P and Q, in precisely the same way that security properties involve quantifying over all possible adversaries. In the asymptotic approach we formulate in this paper, observational equivalence between probabilistic polynomial-time processes coincides with the traditional notion of indistinguishability by polynomial-time statistical tests [Lub96, Yao82], a standard way of characterizing cryptographically strong pseudo-random number generators.

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. This is illustrated in Section 4, where we consider a series of example protocols and relate their security to probabilistic games on the underlying number-theoretic operations. 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 A language for protocols and intruders

2.1 Protocol description

A protocol consists of a set of programs that communicate over some medium in order to achieve a certain task. In this paper, we are concerned with the security of cryptographic protocols, which are protocols that use some set of cryptographic operations. For simplicity, we will only consider protocols that require some fixed number of communications per instance of the protocol. For example, for each client-server session, we assume that there is some fixed number of client-server messages needed to execute the protocol. This is the case for most handshake protocols, key-exchange protocols and authentication protocols, such as Kerberos, the Secure Sockets Layer handshake protocol, and so on. While we do not see any difficulty in extending our basic methods to more general protocols that do not have a fixed bound set in advance, there are some minor technical complications that we avoid by making this simplifying assumption.

We will use a form of π-calculus (a general process calculus) for defining protocols. One reason for using a precise (and perhaps overly detailed language) is to make it possible to define protocols exactly. As will be illustrated by example, many protocols have been described using an imprecise notation that describes possible traces of the protocol, but does not define the way that protocol participants may respond to incorrect messages or other communication that may arise from the intervention of a malicious intruder. In contrast, process calculus descriptions specify the response to adversary actions precisely.

The second reason for defining a precise process computation and communication language is to characterize the possible behavior of a malicious intruder. Specifically, we assume that the protocol adversary may be any process or set of processes that are definable in the language. In the future, we hope to follow the direction established by the spi-calculus [AG97] and use proof methods for forms of observational congruence. However, in order to proceed in this direction, we need further understanding of probabilistic observational congruence and approximations such as probabilistic bisimulation. Since there has been little prior work on probabilistic process formalisms, one of our near-term goals is to better understand the forms of probabilistic reasoning that would be needed to carry out more accurate protocol analysis.

2.2 Protocol language

The protocol language consists of a set of *terms*, or functional expressions that do not perform any communication, and *processes*, which are defined using terms for internal computation and communication and composition primitives. The process part is a restriction of standard π-calculus. The terms we are most interested in are expressions for defining probabilistic polynomial-time computation.

2.3 Processes

For any set of terms, we can define a set of processes. Since we are interested in protocols with a fixed number of steps, we do not need arbitrary looping. We therefore use a bounded subset of asynchronous π-calculus, given by the following grammar:

$$
\begin{array}{lcll}
P & ::= & \mathcal{O} & \text{empty process (does nothing)} \\
 & & \overline{n}\langle M \rangle & \text{transmit value of } M \text{ on port } n \\
 & & n(x).\,P & \text{read value for } x \text{ on port } n \text{ and continue with process } P \\
 & & P|Q & \text{execute process } P \text{ in parallel with } Q \\
 & & \nu n.\,P & \text{execute process } P \text{ with port } n \text{ considered private to } P \\
 & & !_k P & \text{execute up to } k \text{ copies of process } P \\
 & & [M = N]P & \text{if } M = N \text{ then do } P \text{ (guarded command)} \\
 & & \mathsf{let}\ x = M\ \mathsf{in}\ P & \text{bind variable } x \text{ to the value of } M \text{ and execute } P
\end{array}
$$

2.4 Communication

Intuitively, the communication medium for this language is a buffered network that allows messages sent by any process to be received by any other process, in any order. Messages are essentially pairs consisting of a "port name" and a data value. The expression $\overline{n}\langle M \rangle$ places a pair $\langle n, M \rangle$ onto the network. The expression $n(x).\,P$ matches any pair $\langle n, m \rangle$ and continues process P with x bound to value m. When $n(x).\,P$ matches a pair $\langle n, M \rangle$, the pair $\langle n, M \rangle$ is removed from the network and is no longer available to be read by another process. Evaluation of $n(x).\,P$ does not proceed unless or until a pair $\langle n, m \rangle$ is available.

Although we use port names to indicate the intended source and destination of a communication, any process can potentially read any message. (This is not strictly true since a port may be bound by the restriction operator, hiding communication. However, we only intend to use secret ports for specifications, not for modeling protocols, as illustrated in Section 4.) This communication model allows an adversary (or any process) to intercept a message between any two participants in a protocol. Once read, a pair is removed so that an adversary has the power to prevent the intended recipient from receiving a message. An adversary (or other process) may overhear a transaction without interfering simply by retransmitting every communication that it intercepts.

2.5 Example using symbolic cryptosystem

For readers not familiar with π-calculus, we give a brief example using a simple set of terms with "black-box" cryptography. Specifically, for this section only, let us use algebraic expressions over sorts $plain$, $cipher$ and key, representing plaintext, ciphertext and keys, and function symbols

$$
\mathit{encrypt}\colon \mathit{plain} \times \mathit{key} \to \mathit{cipher} \qquad \mathit{decrypt}\colon \mathit{cipher} \times \mathit{key} \to \mathit{plain}
$$

We will illustrate the use of process calculus by restating a simple protocol written in "the notation commonly found in the literature" where $A \to B$ indicates a message from A to B.

In the following protocol, A sends an encrypted message to B. After receiving a message back that contains the original plaintext, A sends another message to B.

$$
\begin{array}{lll}
A \to B\colon & \mathit{encrypt}(p_1, k_B) & (1) \\
B \to A\colon & \mathit{encrypt}(\mathit{conc}(p_1, p_2), k_A) & (2) \\
A \to B\colon & \mathit{encrypt}(p_3, k_B) & (3)
\end{array}
$$

We can imagine that p_1 is a simple message like "hello" and p_3 is something more critical, like a credit card number. Intuitively, after A receives a message back containing p_1, agent A may believe that it is communicating with B because only B can decrypt a message encoded with $B's$ key k_B.

This protocol can be written in π-calculus using the same cryptographic primitives. However, certain decisions must be made in the translation. Specifically, the notation above says what communication will occur when everything goes right, but does not say how the messages depend on each other or what might happen if other messages are received. Here is one interpretation of the protocol above. In this interpretation, B responds to A without examining the contents of the message from A to B. However, in step 3, A only responds to B if the message it receives is exactly the encryption of the concatenation of p_1 and p_2.

$$\begin{array}{ll} \overline{AB}\langle encrypt(p_1, k_B)\rangle & (1) \\ \mid \quad AB(x).\,\overline{BA}\langle encrypt(conc(decrypt(x, K_B), p_2), k_A)\rangle & (2) \\ \mid \quad BA(y).\,[decrypt(y, K_A) = conc(p_1, p_2)]\ \overline{AB}\langle encrypt(p_3, k_A)\rangle & (3) \end{array}$$

In words, the protocol is expressed as the parallel composition of three processes. Port AB is used to indicate a message from A to B and port BA for messages from B to A.

If an intruder runs in parallel with the protocol, then the intruder can intercept any message and replace it by another. For example, the following process could intercept the first message from A to B and replace it by another one using a different key:

$$AB(x).\,\overline{AB}\langle encrypt(p_1, k_C)\rangle$$

2.6 Probabilistic polynomial-time terms

For most of this paper, the syntax of terms is not overly significant. The main property we require of terms is that every probabilistic polynomial-time function can be expressed, and no function beyond this class can be defined. For concreteness, we summarize a specific language that has these properties. This is an applied lambda calculus, based on [Bel92, BC92, KC96, Hof97], that provides a specific syntax for probabilistic polynomial-time terms. The language is based on Hofmann's presentation of the Bellantoni-Cook language of polynomial-time functional expressions, with an added primitive to choose a random bit. Substantial technical effort is required to extend Hofmann's calculus with randomness and establish the basic complexity results. A brief overview appears in Appendix B; a full presentation will be published elsewhere.

Although the language presented in Appendix B includes variables of many types, we will only allow input variables of type $\Box\mathsf{N}$ (natural numbers that may be used as recursion variables) and output expression of type N. This is important for maintaining the complexity bounds.

2.7 Example

Our first example (continued in Section 4.1) is a simple protocol based on ElGamal Encryption [ElG85] and Diffie-Hellman Key Exchange [DH76], formulated in a way that gives us a series of steps to look at. The protocol assumes that a prime p and generator g of $\mathcal{Z}_p^*$ are given and publicly available.

Using the notation commonly found in the security literature, this protocol may be written

$$\begin{array}{lcl} A & \rightarrow & B \;:\; g^a \mod p \\ B & \rightarrow & A \;:\; g^b \mod p \\ A & \rightarrow & B \;:\; \mathsf{msg} * g^{ab} \mod p \end{array}$$

The main idea here is that by choosing a and receiving $g^b \mod p$, Alice can compute $g^{ab} \mod p$. Bob can similarly compute $g^{ab} \mod p$, allowing Alice and Bob to encrypt by multiplying by g^{ab} and decrypt by dividing by g^{ab}. It is generally believed that no eavesdropper can compute $g^{ab} \mod p$ by overhearing g^a and g^b. Since this protocol is susceptible to attack by an adversary who intercepts a message and replaces it, we will only consider adversaries who listen passively and try to determine if the message msg has been sent.

In π-calculus notation, the protocol may be written as follows. We use the convention that port $\overline{AB_i}$ is used for the ith message from A to B, and meta-notation for terms that could be written out in detail in our probabilistic polynomial-time language. To make explicit the assumption that p and g are public, the protocol transmits them on a public port.

let p be a random n-bit prime and g a generator of $\mathcal{Z}_p^*$
in $\overline{PUBLIC}\langle p\rangle \mid \overline{PUBLIC}\langle g\rangle$
$\mid$ let a be a random number in $[1, p-1]$
  in $\overline{AB_1}\langle g^a \mod p\rangle \mid BA(x).\overline{AB_2}\langle \mathsf{msg} * x^a \mod p\rangle$
$\mid$ let b be a random number in $[1, p-1]$
  in $AB_1(y).\overline{BA}\langle g^b \mod p\rangle$

An analysis appears in Section 4.1.

2.8 Parallelism, Nondeterminism and Complexity

For complexity reasons, we must give a nonstandard probabilistic semantics for to parallel composition. Specifically, our intention is to design a language of communicating processes so that an adversary expressed by a set of processes is restricted to probabilistic polynomial time. However, if we interpret parallel composition in the standard nondeterministic fashion, then a pair of processes may nondeterministically "guess" any secret information.

This issue may be illustrated by example. Let us assume that B has a private key K_b that is k bits long and consider the one-step protocol where A encrypts a message using this key and sends it to B.

$$A \;\rightarrow\; B \;:\; \{\mathsf{msg}\}_{K_b}$$

We assume that an evil adversary wishes to discover the message msg. If we allow the adversary to consist of 3 processes E_0, E_1 and E, scheduled nondeterministically, then this can be accomplished. Specifically, we let

$$\begin{array}{lcl} A & = & \overline{AB}\langle encrypt(K_b, \mathsf{msg})\rangle \\ E_0 & = & !_k\overline{E}\langle 0\rangle \\ E_1 & = & !_k\overline{E}\langle 1\rangle \\ E & = & E(b_0).\;\ldots E(b_{k-1}).\,AB(x).\,\overline{Public}\langle decrypt(concatenate(b_0,\ldots,b_{k-1}), \mathsf{msg})\rangle \end{array}$$

Adversary processes E_0 and E_1 each send k bits to E, all on the same port. Process E reads the message from A to B, concatenates the bits that arrive nondeterministically in some order, and decrypts the message. One possible execution of this set of processes allows the eavesdropper to correctly decrypt the message. Under traditional nondeterministic semantics of parallel composition, this means that such an eavesdropper can break any encryption mechanism.

Intuitively, the attack described above should not succeed with much more than probability $1/2^k$, the probability of guessing key K_b using random coins. Specifically, suppose that the key K_b is chosen at random from a space of order 2^k keys. If we run processes E_0, E_1, E on physical computers communicating over an ethernet, for example, then the probability that communication from E_0 and E_1 will accidentally arrive at E in an order producing exacly K_b cannot be any higher than the probability of randomly guessing K_b. Threfore, although nondeterminism is a useful modeling assumption in studying correctness of concurrent programs, it does not seem helpful for analyzing cryptographic protocols.

Since nondeterminism does not realistically model the probability of attack, we use a probabilistic form of parallel composition. This is described in more detail in Appendix A, which contains a full operational semantics. Although there are some slightly awkward aspects of probabilistic scheduling, it is important to keep in mind that *we will only be interested in distinguishing insecure protocols, where an adversary has a significant probability of success, from secure ones where the probability vanishes quickly, as function of a security parameter such as the length of encryption keys.* For the purpose at hand, minor differences in probability will not matter. The main property that we do require is that every process (including compositions of processes) can be simultated by a probabilistic polynomial-time algorithm.

3 Process Equivalence

Observational equivalence, also called observational congruence, is a standard notion in the study of programming languages. The main idea is that program parts, such as functions, processes or abstract data types, can be compared by embedding them in full programs that may produce observable actions or output. For each language $\mathcal{L}$, we assume there is some set of *program contexts*, each context $\mathcal{C}[\ ]$ consisting of a program with a "hole" (indicated by empty square brackets $[\ ]$) to insert a phrase of the language, and some set Obs of concrete *observable actions*, such as an integer outputs or final values. We assume that there is some semantic evaluation relation $\stackrel{eval}{\leadsto}$, with $M \stackrel{eval}{\leadsto} v$ meaning that evaluation of the program M produces the observable action v. (In a functional language, this would mean that v is a possible value of M, while in a concurrent setting this might mean that v is a possible output action.) Under these assumptions, we may associate an *experiment* on program phrases with each context $\mathcal{C}[\ ]$ and observable v: given phrase P, run the program $\mathcal{C}[P]$ obtained by placing P in the given context and see whether observable action v occurs.

In general, two program phrases (presumably of the same type) P and Q are observationally equivalent if they give exactly the same experimental results. More formally, we say program phrases P and Q are *observationally equivalent*, written $P \simeq Q$, if, for all program contexts $\mathcal{C}[\ ]$ and observables $v \in \mathcal{O}$, we have

$$\mathcal{C}[P] \stackrel{eval}{\leadsto} v \quad \text{iff} \quad \mathcal{C}[Q] \stackrel{eval}{\leadsto} v$$

In other words, $P \simeq Q$ if, for any program $\mathcal{C}[P]$ containing P, we can make exactly the same concrete observations about the behavior of $\mathcal{C}[P]$ as we can about the behavior of the program $\mathcal{C}[Q]$ obtained by replacing specific occurrences of P by Q. Note that for any given language, it may be possible to identify various sets of observables and, *a priori*, the relation $\simeq$ may depend on this choice. However, for most languages, the relation is not affected by variations in the set of observables.

This general definition of $\simeq$ is essentially standard for deterministic or nondeterministic functional, imperative or concurrent languages. Some additional considerations enter when we consider probabilistic languages. (What follows is essentially original, although inspired by standard notions in cryptography.) Intuitively, given program phrases P and Q, context $\mathcal{C}[\ ]$ and observable action v, it seems reasonable to compare the probability that $\mathcal{C}[P] \stackrel{eval}{\leadsto} v$ to the probability that $\mathcal{C}[Q] \stackrel{eval}{\leadsto} v$. However, since a probability distribution is an infinite entity, it is not clear how to "observe" a distribution. We might run $\mathcal{C}[P]$ some number of times, count how many times v occurs, and then repeat the series of experiments for $\mathcal{C}[Q]$. If the probabilities are very different, then we might be able to observe this difference (with high confidence) by a few runs of each program. However, if the probabilities are very close, then it might take many more runs of both programs to distinguish them.

We define computational indistinguishability within factor ϵ by

$$P \simeq_\epsilon Q \qquad \text{iff} \qquad \forall \mathcal{C}[\ ].\, \forall v \in \mathit{Obs}.\ |\mathrm{Prob}[\mathcal{C}[P] \stackrel{eval}{\leadsto} v] - \mathrm{Prob}[\mathcal{C}[Q] \stackrel{eval}{\leadsto} v]| \leq \epsilon$$

A difficulty with $\simeq_\epsilon$ is that it is not clear how to differentiate large ϵ from small ϵ. Specifically, we would like to draw a distinction between sets of processes that are "close" in behavior from those that are "far apart." Intuitively, the distinction should have something to do with running time, since it takes more trials to distinguish random variables that differ by a small amount than it does to distinguish random variables that differ by a large amount.

We can bring some concepts from asymptotic complexity theory to bear on the situation if the processes P and Q under consideration are actually families of processes indexed by natural numbers. This fits our intended application, since cryptographic primitives and security protocols are generally defined with some *security parameter* that may be increased if greater resistance to tampering is needed. A typical security parameter is the number of bits used in an encryption key. This is a parameter of any protocol that begins by generating encryption and decryption keys, since these are usually chosen to be of a specific length. If the keys length is increased, then greater security is generally provided.

Let us assume that $P = \{P_n\}_{n \geq 0}$ and $Q = \{Q_n\}_{n \geq 0}$ are indexed families of processes. We can also consider contexts as similarly parameterized, so that a context family consists of a set $\mathcal{C}[\ ] = \{\mathcal{C}_n[\ ]\}_{n \geq 0}$ of contexts. In our setting, a context provides a set of processes to be run in parallel with the protocol being observed. We assume that the running times of P_n, Q_n and $\mathcal{C}_n[\ ]$ are bounded by polynomials in n, written in unary notation. Then for function f, we define *asymptotic equivalence within* f by

$$\begin{aligned} P \simeq_f Q \quad \text{if} \quad & \forall \mathcal{C}[\ ].\, \forall v \in \mathit{Obs}.\, \exists n_0.\, \forall n \geq n_0. \\ & |\mathrm{Prob}[\mathcal{C}_n[P_n] \stackrel{eval}{\leadsto} v] - \mathrm{Prob}[\mathcal{C}_n[Q_n] \stackrel{eval}{\leadsto} v]| \leq f(n) \end{aligned}$$

In words, P and Q are asymptotically equivalent within f if, for every computational experiment given by a context and an observable value, the difference between experimental observation of P_n and experimental observation of Q_n is bounded by $f(n)$, for all sufficiently large n.

Since we consider polynomial factors “small”, we define *observational equivalence of probabilistic processes* by

$$P \simeq Q \quad \text{if} \quad P \simeq_{1/p} Q \text{ for every polynomial } p.$$

Example: If $P = \{P_n\}_{n \geq 0}$ is a scheme for generating pseudorandom sequences of bits, and $Q = \{Q_n\}_{n \geq 0}$ consists of processes that generate truly random bits (e.g., by calls to our built-in random-bit primitive), then our definition of observational equivalence corresponds to a standard notion from the study of pseudorandomness and cryptography (see, e.g., [Lub96, Yao82]). Specifically, $P \simeq Q$ iff P and Q pass the same polynomial-time statistical tests.

It seems worth mentioning a technical difference between this probabilistic framework and the nondeterministic spi-calculus. In order to characterize hiding via encryption in the spi-calculus, a specialized form of process equivalence must be developed. In contrast, our model is not based on any special axioms that are designed to model “hiding” or encryption specifically. Instead, we use a general notion of observational equivalence that seems to be intrinsically plausible, independent of any interest in cryptography.

4 Specification of Security Properties

4.1 A variant of ElGamal Encryption

Protocol In section 2.7, we wrote a variant of ElGamal encryption as a three-step protocol and expressed this in probabilistic π-calculus. We can regard this protocol as a parameterized family of protocols by making the dependence on the lenght of the public prime (and therefore the key length) explicit:

protocol P_n:

let p be a random n-bit prime and g a generator of $\mathcal{Z}_p^*$
in $\overline{PUBLIC}\langle p\rangle \mid \overline{PUBLIC}\langle g\rangle$
$\mid$ let a be a random number in $[1, p-1]$
in $\overline{AB_1}\langle g^a \mod p\rangle \mid BA(x).\,\overline{AB_2}\langle \mathsf{msg} * x^a \mod p\rangle$
$\mid$ let b be a random number in $[1, p-1]$
in $AB_1(y).\,\overline{BA}\langle g^b \mod p\rangle$

Note that although the algorithm for generating a prime is probabilistic and may fail with small probability to produce a prime, our specification will also contain the same algorithm, compensating for this minor difficulty.

Specification This protocol for sending a message is easily defeated if an adversary intercepts g^b from Bob and sends g^c, for then Alice will send $\mathsf{msg} * g^{ac} \mod p$ and the adversary knowing g^a and c can easily decrypt. However, the protocol is secure against a non-interfering eavesdropper, under the assumption that discrete logarithm is hard. We will state precisely what we mean by “secure” and give a form of discrete logarithm recognition problem that is equivalent to decrypting the message from Alice to Bob.

Our specification is that if contexts are restricted so that messages may be read and replayed, but no new messages are sent by the adversary, then P_n is asymptotically observationally equivalent to the following protocol:

Protocol Q_n:

> let p be a random n-bit prime and g a generator of $\mathcal{Z}_p^*$
> in $\overline{PUBLIC}\langle p\rangle \mid \overline{PUBLIC}\langle g\rangle$
> $\mid$ let a be a random number in $[1, p-1]$ in $\overline{AB_1}\langle a\rangle$
> $\mid$ let c be a random number in $[1, p-1]$ in $BA(x).\overline{AB_2}\langle c\rangle$
> $\mid$ let b be a random number in $[1, p-1]$ in $AB_1(y).\overline{BA}\langle b\rangle$

In the specification Q_n, each secret message in P_n is replaced by a random number in the appropriate range. Intuitively, the equivalence $P \simeq Q$ means that to any other process observing P, the network traffic appears to be a series of encrypted random numbers. The reason we send encrypted random numbers instead of random numbers is that any variation in probability distribution induced by encryption should not be counted as a protocol flaw. In other words, our view of security here is that we wish to express that the content of the messages is hidden; this specification does not require the fact that encryption is used to be hidden.

Equivalent Game We cannot hope to prove $P \simeq Q$ without establishing significant new results about the difficulty of computing discrete logarithms. Instead, we will prove that any context that distinguished P and Q provides a method for asymptotically winning the following family of games, based on the decision form of Diffie-Hellman (see [NR97]):

Game G_n:

> Player A: Announces a pair of numbers $\langle g, p\rangle$ and displays two cards with triples of numbers $\langle a, b, c\rangle$ and $\langle a', b', c'\rangle$, one consisting of three random numbers $1 < a, b, c < p$ and the other random numbers of the form $\langle g^u, g^v, g^{uv}\rangle$ (mod p).
>
> Player B: Chooses one of the triples, winning the game if the triple has the form $\langle g^u, g^v, g^{uv}\rangle$.

Intuitively, the first two numbers of a random triple $\langle g^u, g^v, g^{uv}\rangle$, with all three exponentiations computed modulo p, are simply randomly chosen numbers in the interval $[2, p-1]$. Therefore, the game consists of trying to determine whether the third number of a triple bears the indicated relationship to the other two.

Protocol Correctness First note that since $[1, p-1]$ is a multiplicative group, multiplication by msg is simply a permutation, so $\mathsf{msg} * c$ will appear to an eavesdropper as a random chosen value in $[1, p-1]$. Thus, a game G'_n where a tuple $\langle a, b, \mathsf{msg} * c\rangle$ is generated instead of random $\langle a, b, c\rangle$ will be equivalent to G_n. Specifically, the distribution of pairs of cards where one pair is generated by choosing random a, b, c and presenting $\langle a, b, \mathsf{msg} * c\rangle$ and the other by choosing random $\langle g^u, g^v, g^{uv}\rangle$

will be exactly the same as G_n. Therefore, we show that a context family $\mathcal{C}[\ ]$ asymptotically distinguishing P from Q will asymptotically win game G' with the related probability.

Suppose $\mathcal{C}[\ ]$ is a family of pasive eavesdropping contexts and $v \in Obs$ an observable such that

$$\forall n_0.\, \exists n \geq n_0. \qquad |\mathrm{Prob}[\mathcal{C}_n[P_n] \overset{eval}{\leadsto} v] - \mathrm{Prob}[\mathcal{C}_n[Q_n] \overset{eval}{\leadsto} v]| > f(n)$$

and let n be any number where the difference in probabilities is greater than $f(n)$. Let $\langle u, v, w\rangle$ and $\langle u', v', w'\rangle$ be a pair of cards generated by Player A in game G'_n after announcing g and p.

Our objective is to construct a pair of protocols that can be distinguished by $\mathcal{C}_n[\ ]$ and use this to determine Player B's move in game G'_n. We can easily do this using a general template

Template $R(p, g, a, b, c)$:

$$\begin{array}{ll} & \overline{PUBLIC}\langle p\rangle \mid \overline{PUBLIC}\langle g\rangle \\ \mid & \overline{AB_1}\langle a\rangle \\ \mid & BA(x).\, \overline{AB_2}\langle c\rangle \\ \mid & AB_1(y).\, \overline{BA}\langle b\rangle \end{array}$$

The main idea is that $R(p, g, u, v, w)$ and $R(p, g, u', v', w')$ appear identical to instances of P_n and Q_n. Since the probabilities $\mathrm{Prob}[\mathcal{C}_n[P_n] \overset{eval}{\leadsto} v]$ and $\mathrm{Prob}[\mathcal{C}_n[Q_n] \overset{eval}{\leadsto} v]$ differ by $f(n)$, we can sample the probabilities $\mathrm{Prob}[\mathcal{C}_n[R(p, g, u, v, w)] \overset{eval}{\leadsto} v]$ and $\mathrm{Prob}[\mathcal{C}_n[R(p, g, u', v', w')] \overset{eval}{\leadsto} v]$ using a number of runs polynomial in $1/f(n)$ and determine which of the instances of template $R(p, g, a, b, c)$ most closely resembles P_n. This card can be chosen by Player B.

4.2 Part of Needham-Schroeder Private-Key Protocol

Protocol Let us consider the following authentication protocol, intended to ensure that Alice knows she is talking to Bob if they share a private key k. Let i be a random binary string of length n and let f be a numerical function computable in probabilistic polynomial time.

$$\begin{array}{lcl} A \rightarrow B & : & \{i\}_k \\ B \rightarrow A & : & \{f(i)\}_k \\ A \rightarrow B & : & \text{OK} \end{array}$$

The main idea is that Alice sends a nonce to Bob and Bob returns some function of that nonce. When Alice receives the message she expects, she concludes that Bob read her nonce since the encryption key is assumed to be shared only with Bob. This protocol may be expressed in our framework as follows, using key-generation, encryption and decryption functions computable in probabilistic polynomial time:

protocol P_n:

let k be a random n-bit key and i a random n-bit number
in $\overline{AB}\langle encrypt(k, i)\rangle$
$\quad\mid AB(x).\, \overline{BA}\langle encrypt(k,\ f(decrypt(k, x)))\rangle$
$\quad\mid BA(y).\, [y = encrypt(k,\ f(i))]\ \overline{AB}\langle\text{“OK”}\rangle$

Specification Our specification is a similar family of processes, sending encrypted random numbers in place of the "data" sent in the protocol. We specify that the protocol should not continue if a message is alterned by using a private channel in the specification.

protocol Q_n:

$$\begin{array}{l} \nu\, \mathit{PRIV}.\ \text{let } k \text{ be a random } n\text{-bit key and } i, j \text{ a random } n\text{-bit numbers} \\ \quad \text{in } \overline{AB}\langle encrypt(k,i)\rangle \\ \qquad \mid AB(x).\,(\overline{BA}\langle encrypt(k,j)\rangle \mid \overline{PRIV}\langle x\rangle) \\ \qquad \mid BA(y).\,PRIV(x).\,[x = encrypt(k,i)]\,[y = encrypt(k,j)]\ \overline{AB}\langle \text{“OK”}\rangle \end{array}$$

This specification illustrates a general technique we have found useful for authentication aspects of protocols. Although we do not expect this protocol (which models a key idea used in Kerberos, for example) to be implemented using private channels, we use a private channel in the specification as a way of expressing the ideal observable behavior of the protocol. In this specification, B forwards on the private channel the message x that B receives on the open channel. This allows A to check, in the third step, whether the messages have been tampered with by an intruder.

Correctness The observational equivalence of these two processes P and Q may be reduced to a certain game in the sense that if the processes P and Q may be distinguished by some probabilistic polynomial-time context, then there are probabilistic polynomial-time strategies for winning the associated game. This game may be regarded as a combination of subgames. The object of the first subgame is to distinguish between pairs $\langle\{rand\}_k, \{rand'\}_k\rangle$ and $\langle\{i\}_{k'}, \{f(i)\}_{k'}\rangle$, where i and f are as above, $rand$ and $rand'$ are independent random binary strings of length n, and k and k' are randomly chosen keys of length n. The object of the second game is to determine whether there exists a numerical function g such that the pairs $\langle\{i\}_k, \{j\}_k$ and $\langle\{i\}_{k'}, \{f(decrypt_{k'}(g(\{i\}_{k'})))\}_{k'}$ may be distinguished. Here i and f are as above, j is an independent random binary string of length n, and k and k' are randomly chosen keys of length n. For either game, "to distinguish between two pairs" means to find a polynomial $p(n)$ in one variable n and with non-negative integer coefficients such that the probability that the two pairs are different exceeds $1/p(n)$.

Suppose that processes P and Q are not observationally equivalent. That is, there exists a polynomial $p(n)$ in one variable n with non-negative integer coefficients and a set of contexts $\mathcal{C}[\ ] = \{\mathcal{C}_n[\ ]\}_{n\geq 0}$ such that for every n_0, there exists $n \geq n_0$ such that the probabilities that $\mathcal{C}_n[P_n]$ evaluates to "OK" and that $\mathcal{C}_n[Q_n]$ evaluates to "OK" differ by at least $1/p(n)$. If this context family consists only of passive eavesdropping contexts, then the context may be used to construct a winning strategy for the first game described above. Otherwise, a context step interfering with protocol communication results in a numerical function g as described above. Further details are omitted due to space limitations.

4.3 Needham-Schroeder Public Key

We have also analyzed both the originally flawed Needham-Schroeder public-key protocol to see how the flaw may be indentified in our framework and analyzed the corrected version of this protocol for comparison. Due to space limitations, we give only the informal definition of the flawed protocol, the result of translation into probabilistic π-calculus, and a specification protocol.

Flawed protocol

$$
\begin{array}{l}
A \rightarrow B \colon \{N_A, K_A\}_{K_B} \\
B \rightarrow A \colon \{N_A, N_B\}_{K_A} \\
A \rightarrow B \colon \{N_B\}_{K_B}
\end{array}
$$

This is the the essential portion of the original Needham-Schroeder Public Key protocol. As discussed above, this notation leaves many details unspecified, such as what happens if A recieves messages that are not correctly encrypted. So we formalize this protocol in our lanuage

Let $K_A, K_A^{-1}, K_B, K_B^{-1}, \ldots, N_A, N_B$

$$
\overline{\mathrm{Key}}.K_A \quad | \quad \overline{\mathrm{Key}}.K_B \quad |
$$

$$
\mathrm{Key}(K)[K \neq K_A].\left(\begin{array}{l}
\overline{\mathrm{Key}}.K \\
| \quad \overline{AB}\langle\{N_A, K_A\}_K\rangle \\
| \quad AB(Z).\langle x, y\rangle = dec(z, K_B^{-1})\overline{BA}\langle\{x, N_B\}_y\rangle \\
| \quad BA(C).\langle d, f\rangle = dec(c, K_A^{-1})\,[d = N_A]\,(\overline{AB_2}\langle\{f\}_K\rangle \quad | \quad \overline{\mathrm{out}}.AOK) \\
| \quad AB_2(g).h = dec(g, K_B^{-1})\,[h = N_B]\,\overline{\mathrm{out}}.BOK
\end{array}\right)
$$

Specification This implementation is intended to provide assurance of the authenticity of the participants. We specify this by providing a specification which is generated from the above implementation by replacing the important decryption steps with message sends on a new truly private channel $priv$. This private channel is used to send the ”real” information. Since no interesting information is sent on a public channel, no context could possibly capture private information. Thus if some protocol implementation is observationally equivalent to a specification which ’obviously’ provides privacy, we can infer that the implementation provides privacy. Further, one can specify that the information sent on the public channel also be sent on the private channel, and then checked by the recipient. If any adversary were to modify a message in any way, such a specification would detect the change immediately. Again, if one could show a protocol equivalent to a specification which ’obviously’ provides authenticity in this way, then one could infer that the protocol has this desirable property.

In the present case we use the somewhat artificial report at the end of the protocol (and the specification below) of the participants asserting on a public channel (out) the msssages AOK and BOK. These are the outputs which a context can cause to occur through the protocol or specification.

Let $K_A, K_A^{-1}, K_B, K_B^{-1}, \ldots, N_A, N_B, r_1, r_2, r_3, r_4, r_5, r_6$

$$
\overline{\mathrm{Key}}.K_A \quad | \quad \overline{\mathrm{Key}}.K_B \quad |
$$

$$
\mathrm{Key}(K)[K \neq K_A].\left(\begin{array}{l}
\overline{\mathrm{Key}}.K \\
| \quad \overline{AB}\langle\{N_A, K_A\}_K\rangle \\
| \quad AB(Z).\langle x, y\rangle = dec(z, K_B^{-1})\overline{priv_1}(z)|\overline{BA}\langle\{r_3, r_4\}_y\rangle \\
| \quad BA(C).\langle d, f\rangle = dec(c, K_A^{-1})priv_1(z).\,[d = r_3, z = \{N_A, r_2\}_{K_B}]\,(\overline{AB_2}\langle\{r_6\}_K\rangle \quad | \quad \overline{\mathrm{out}}.AC \\
| \quad AB_2(g).h = dec(g, K_B^{-1})\,[h = N_B]\,priv_2(c).[c = \{r_3, r_5\}_y]\overline{\mathrm{out}}.BOK
\end{array}\right.
$$

This specification Q may be somewhat difficult to read, but it has the same structure as the implementation P. Again, the question of observational equivalence of P and Q may be reduced to a game in the sense that if P and Q may be distinguished by polynomial time statistical tests, then there are polynomially computable strategies for winning the game. In this case, P and Q are *not* observationally equivalent, as the classic man-in-the-middle attack on Needham-Schroeder demonstrates.

5 Differences from spi-calculus

While we have drawn much of our inspiration from the spi-calculus, there are some important differences in objective and mathematical setting. We summarize these so that the reader may understand the main points of comparison. This short section is *not* a criticism of spi-calculus, which seems a valuable framework for analyzing protocols at a specific level of abstraction.

The most significant difference is that we have developed a probabilistic framework with complexity restrictions. We also use asynchronous communication, which we regard as a relatively minor technical difference. In our approach, we have made limited use of π-calculus restriction operator. Specifically, we use private port names to provide idealized "secret" communication in specifications, but not as a primitive for creating new data values such as nonces. Our asymptotic notion of process equivalence is, in some sense, less "contrived" than equivalence in spi-calculus, since there is no need to make special provision for secrecy or observability of encrypted data.

Finally, it seems safe to say that it is much more difficult to establsih correctness properties of non-trivial protocols in our framework. This is in part because the framework is a more significant departure from established process calculus formalisms.

6 Conclusion and Future Directions

The language presented in this paper allows us to define probabilistic polynomial-time processes, communicating over a network that gives an adversarial process access to communication between other processes. Many of the language design decisions are motivated by interests in security properties, as illustrated by a series of examples throughout the paper. In particular, the probabilistic semantics of parallel composition is chosen to avoid unrealistic attacks on complexity-based encryption schemes.

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

The steps taken in this paper form the beginning of a larger program that we hope to carry out over the next few years. In part following the program established in the study of spi-calculus [AG97], we hope to develop methods for reasoning about observational equivalence (or some approximation to observational equivalence such as probabilistic trace equivalence or bisimulation) and use these methods to establish security properties of various protocols. We expect some interesting foundational questions to arise in the formulation of security properties such as authentication and secrecy. It may also be possible to develop model-checking procedues along the lines of these already explored for probabilistic temporal logics [HJ89, Han94, dA97, HK97].

Acknowledgements: Thanks to Martin Abadi, Stephen Bellantoni, Dan Boneh, Cynthia Dwork, Stefano Guerrini, Martin Hofmann, Sampath Kannan, Bruce Kapron, Moni Naor and Jim Royer for helpful discussions and advice on relevant literature.

References

[AG97] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. In *Proc. Concur '97*, pages 59–73. Springer LNCS 1243, 1997.

[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.

[BC92] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

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

[BgB90] G. Berry and g. Boudol. The chemical abstract machine. In *Proc. 17th ACM Symp. Principles of Programming Languages*, pages 81–94, 1990.

[Cob64] A. Cobham. The intrinsic computational difficulty of functions. In *Proc. Int'l Cong. Logic Methodology and Philosophy of Science*, pages 24–30. North-Holland, 1964.

[dA97] L. de Alfaro. Temporal logics for the specification of performance and reliability. In *STACS'97*, volume 1200 of *LNCS*, pages 165–176. Springer-Verlag, February 1997.

[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.

[de 97] L. de Alfaro. *Formal verification of probabilistic systems*. PhD thesis, Stanford University Dept. of Computer Science, 1997.

[Der] C. Derman. *Finite-state Markov decision processes*. Academic Press', year=.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[DP96] R. Davies and F. Pfenning. A modal analysis of staged computation. In *23rd Annual ACM Symposium on Principles of Programming Languages (POPL'96)*, pages 258–270, 1996.

[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.

[FKK96] A. Freier, P. Karlton, and P. Kocher. The SSL protocol version 3.0. `draft-ietf-tls-ssl-version3-00.txt`, November 18 1996.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

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

[Han94] H. Hansson. *Time and Probabilities in Formal Design of Distributed Systems*. Real-Time Safety Critical Systems. Elsevier, 1994.

[HJ89] H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In *Proc. of Real Time Systems Symposium*, pages 102–111. IEEE, 1989.

[HK97] M. Huth and M. Kwiatkowska. Quantitative analysis and model checking. In *LICS'97*, pages 111–122, 1997.

[Hof97] M. Hofmann. A mixed/modal lambda calculus with applications to Bellantoni-Cook safe recursion. Manuscript; see `http://www.mathematik.th-darmstadt.de/~mh/`, 1997.

[KC96] B.M. Kapron and S.A. Cook. A new characterization of type-2 feasibility. *SIAM J. Computing*, 25(1):117–132, 1996.

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

[KN93] J.T. Kohl and B.C. Neuman. The Kerberos network authentication service (version 5). Internet Request For Comment RFC-1510, September 1993.

[KNT94] J.T. Kohl, B.C. Neuman, and T.Y. Ts'o. *The evolution of the Kerberos authentication service*, pages 78–94. IEEE Computer Society Press, 1994.

[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.

[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.

[Mil92] R. Milner. Functions as processes. *Math. Structures in Computer Science*, 2(2):119–141, 1992.

[Mit96] J.C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[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] M. Mitchell, J. Mitchell, and A. Scedrov. A logical characterization of probabilistic polynomial time. 1998.

[Mog89] E. Moggi. Computational lambda calculus and monads. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 14–23, 1989.

[NR97] M. Naor and O. Reingold. Efficient cryptographic primitives based on the decisional Diffie-Hellman assumption. In *Proc. 38th IEEE Symp. on Foundations of Computer Science*, 1997.

[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.

[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.

[TMN90] M. Tatebayashi, N. Matsuzaki, and D.B. Newman. Key distribution protocol for digital mobile communication systems. In *Proc. CRYPTO '89*, pages 324–333, 1990.

[WLPD98] Philip Wickline, Peter Lee, Frank Pfenning, and Rowan Davies. Modal types as staging specifications for run-time code generation. *ACM Surveys: Special Issue on Partial Evaluation*, page (To appear), 1998.

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

A Operational Semantics

We use an operational semantics similar in outline to the spi-calculus semantics given in [AG97]. This is a variant of Milner's reaction relation approach [Mil92], inspired by the Chemical Abstract Machine of Berry and Boudol [BgB90]. One difference is that we only allow values (the results of evaluating terms) to be communicated. Another is the use of probabilistic schedules instead of nondeterminism.

The operational semantics is defined using three relations on processes. The first is the evaluation relation $\Downarrow$ on closed terms, which we will take as given by the language of terms. The second is a structural congruence relation $\equiv$ on processes and the third a reduction relation $\rightarrow$ on processes. Computation proceeds by probabilistic $\rightarrow$ reduction steps on $\equiv$-equivalence classes of processes, with $\Downarrow$ used to define reduction.

Since the terms we are most interested in are evaluated probabilistically, we assume a probabilistic evaluation relation on terms, with $M \Downarrow_r v$ indicating that if we choose to evaluate M, then with probability r, the result will be value v. We may also write $Prob[M \Downarrow v] = r$ to express that $M \Downarrow_r v$. Since evaluation of probabilistic polynomial-time terms is guaranteed to terminate, we know that for any term M, there is a set V of values so that $\sum_{v \in V} Prob[M \Downarrow v] = 1$.

The structural equivalence relation formalizes the intuitive fact that a process can be written in a variety of syntactic forms. The inference rule

$$\text{(React Struct)} \quad \frac{P' \equiv P \quad P \rightarrow Q \quad Q \equiv Q'}{P' \equiv Q'}$$

indicates that structurally equivalent processes have precisely the same reductions. Structural equivalence is defined by the following axioms and inference rules.

$$\begin{array}{lrcl} \text{(Struct Refl)} & P & \equiv & P \\ \text{(Struct Nil)} & P|\mathcal{O} & \equiv & P \end{array}$$

$$
\begin{array}{llcll}
\text{(Struct } !_k\text{)} & !_k P & \equiv & \overbrace{P|P|\ldots|P}^{k} & \\
\text{(Struct Comm)} & P|Q & \equiv & Q|P & \\
\text{(Struct Assoc)} & P|(Q|R) & \equiv & (P|Q)|R & \\
\text{(Struct Switch)} & \nu m.\, \nu n.\, P & \equiv & \nu n.\, \nu m.\, P & \\
\text{(Struct Extrusion)} & \nu n.\, (P|Q) & \equiv & P|\nu n.\, Q & \text{provided } n \notin fn(P)
\end{array}
$$

$$
\text{(Struct Symm)} \quad \frac{P \equiv Q}{Q \equiv P} \qquad \text{(Struct Trans)} \quad \frac{P \equiv Q \quad Q \equiv R}{P \equiv R}
$$

$$
\text{(Struct Par)} \quad \frac{P \equiv P'}{P|Q \equiv P'|Q} \qquad \text{(Struct Res)} \quad \frac{P \equiv P'}{\nu n.\, P \equiv \nu n.\, P'}
$$

The first form of reduction is communication between processes, with the remaining involving "internal" reduction without communication. While communication is determinisitic, once input and output are chosen, an internal step may have an associated probability distribution, induced by probabilistic evaluation of terms.

$$
\begin{array}{llcll}
\text{(React Inter)} & \overline{n}\langle v\rangle|n(x).\, P & \rightarrow & [v/x]P & \\
\text{(Red Let)} & \mathsf{let}\ x = M\ \mathsf{in}\ P & \rightarrow_r & [v/x]P & \text{provided } M \Downarrow_r v \\
\text{(Red Output)} & \overline{n}\langle M\rangle & \rightarrow_r & \overline{n}\langle v\rangle & \text{provided } M \Downarrow_r v \\
\text{(Red Test)} & [M = N]P & \rightarrow_{rs} & P & \text{provided } M \Downarrow_r v \text{ and } N \Downarrow_s v
\end{array}
$$

Reduction may occur inside a parallel composition or restriction, as indicated by the final inference rules.

$$
\text{(React Par)} \quad \frac{P \rightarrow P'}{P|Q \rightarrow P'|Q} \qquad \text{(React Res)} \quad \frac{P \rightarrow P'}{\nu n.\, P \rightarrow \nu n.\, P'}
$$

This concludes the definition of reduction, except for probabilistic considerations.

Probability distribution In the axioms and inference rules above, we have defined an operational semantics that is both probabilistic and nondeterministic. As suggested by researchers in other contexts (e.g., Markov decision processes [de 97, Der]), nondeterminism and probability can be combined by introducing additional machinery to associate probabilities with nondeterministic choices.

One natural way to determine probabilities is through the notion of *policy*, used in [de 97, Der]. The main idea is that a policy (or scheduler) associates a probability with each action, conditional on the sequence of preceding actions. While our interest in security protocols suggests that we should allow the protocol adversary to choose any scheduling policy that is computable in probabilistic polynomial time, we will consider a more restricted setting in this paper. One complication with adversary-chosen policies arises in connection with restriction: if a process representing one participant in a protocol uses local communication for some purpose, the adversary should not be able to use this to determine the probability of one of its own actions. In order to avoid this issue, and generally simplify our semantics, we therefore adopt a fixed policy with uniform distribution: if process P can be written in k structurally equivalent forms, $P_1, \ldots, P_k$, each with a corresponding distinct reduction $P_i \rightarrow_r Q_i$, then we let the probability $P \rightarrow Q_i$ be r/k.

Although space considerations prevent us from developing this idea in full, we remark that if we rename communication ports so that the protocol adversary intercepts every communication, then the protocol adversary may effectively control the probability of each action. Therefore, the more general setting of adversary-chosen scheduling policies is definable within our more restricted language based on a uniform scheduling policy.

We can consider a process P as a function from network contents (atoms $\overline{PORT}\langle n \rangle$) to network contents. Specifically, we start the process with some set of pairs in the network buffer and execute the processes until all have terminated (or are stuck waiting for input). The set of pairs remaining in the network form the "output" of the program. We therefore measure running time as a function of the lengths of all the natural numbers in the initial network buffer.

Theorem A.1. *The running time of process P, measured as a function of the size of the initial network contents, is bounded by a polynomial whose degree may be determined from the height of the execution tree of P and the polynomial bounds on all the terms that occur in P.*

B Probabilistic poly-time terms

We use an applied lambda calculus, based on [Bel92, BC92, KC96, Hof97], to define a class of probabilistic sequential processes. The language is based on Hofmann's presentation of the Bellantoni-Cook language of polynomial-time functional expressions, with an added primitive to choose a random bit. Substantial technical effort is required to extend Hofmann's calculus with randomness and establish the basic complexity results. This section contains a brief summary of the system. Additional details and proofs of complexity results are given in a separate paper to be submitted for publication shortly.

In brief, the origin of this language lies with Cobham's early characterization of polynomial-time computable functions using a bounded form of primitive recursion [Cob64]. That characterization has been reformulated in various ways (e.g., [Bel92, BC92, KC96]), leading to a notation in which a side condition involving a polynomial bound is replaced by a distinction between two kinds of function arguments, *normal* and *safe*. Intuitively, a *safe* function argument has only restricted uses, limiting the ability to define primitive recursive functions of primitive recursive arguments. In [Bel92, BC92], a syntactic characterization of polynomial-time is given using functions with these two classes of arguments. In recent work, Hofmann has developed a variant of the Bellantoni-Cook language where the two classes of arguments are given two different types [Hof97]. The distinction between the two kinds of types is related to distinctions from linear logic [Gir87] and has been used for other purposes [DP96, WLPD98].

We use the type system of [Hof97] to distinguish between ordinary natural numbers and "safe" natural numbers that have only restricicted use. The types are given by the grammar

$$\begin{array}{rcll} \tau & ::= & \mathsf{N} & \text{(restricted natural numbers)} \\ & | & \tau \to \tau & \text{(function type)} \\ & | & \Box\tau \to \tau & \text{(functions from unrestricted inputs)} \end{array}$$

Intuitively, an expression of type N denotes a "safe" natural number that may not be used to produce arbitrarily large values, while an expression of type $\Box\mathsf{N}$ denotes a natural number that may be used arbitrarily, as the recursion variable in a primitive recursive definition or otherwise. One contribution of [Hof97] is the limited way that the modality $\Box$ may occur in types. This avoids the expression forms associated with ! in linear logic and, more generally associated with any modal type operator associated with any monad [Mog89]. A second innovation we adopt from [Hof97] is a form of subtyping, with $A \to B <: \Box A \to B$, further avoiding explicit conversions between types. Together, these innovations allow a useful form of type inference [Hof97].

The *expression*s in the language are given by the following grammar, where v may be any variable and A any type:

$$
\begin{array}{rcll}
e & ::= & v & \text{(variable)} \\
& | & n & \text{(numeral)} \\
& | & \mathsf{S}_0 \mid \mathsf{S}_1 & \text{(doubling functions)} \\
& | & (e_1\ e_2) & \text{(application)} \\
& | & \mathsf{fun}(v{:}\,A)\ e & \text{(abstraction)} \\
& | & \mathsf{case}_A\ e_1\ \mathsf{zero}\ e_2\ \mathsf{even}\ e_3\ \mathsf{odd}\ e_4 & \text{(case distinction)} \\
& | & \mathsf{saferec} & \text{(safe recursion))} \\
& | & \mathsf{rand} & \text{(random bit)}
\end{array}
$$

Variables, lambda abstraction and application are standard from typed lambda calculus (see, *e.g.*, [Mit96]), with the modification that $\mathsf{fun}(v{:}\,A)\ e$ may have types $A \to B$, $\Box A \to B$, $A \multimap B$, or $\Box A \multimap B$, according to the type inference algorithm given in [Hof97]. Functions S_0 and S_1 double a number or double and add 1, and case_A has three branches, according to whether the first argument is zero, positive odd, or positive even. The restricted primitive recursion operator $\mathsf{saferec}$ is described below. The function rand returns a random bit.

The type system is an extension of standard typed lambda calculus, with subtyping as described above and restrictions on computation achieved by careful distinction between N and $\Box\mathsf{N}$ in the typing of basic operations. The type $\tau(c)$ of a constant c is defined by:

$$
\begin{array}{rcl}
\tau(n) & = & \mathsf{N}, \text{ when } n \text{ is an integer constant} \\
\tau(\mathsf{S}_0) & = & \mathsf{N} \to \mathsf{N} \\
\tau(\mathsf{S}_1) & = & \mathsf{N} \to \mathsf{N} \\
\tau(\mathsf{case}_A) & = & \mathsf{N} \to A \to A \to A \\
\tau(\mathsf{saferec}) & = & \Box\mathsf{N} \to \mathsf{N} \to (\Box\mathsf{N} \to \mathsf{N} \to \mathsf{N}) \to \mathsf{N} \\
\tau(\mathsf{rand}) & = & \mathsf{N}
\end{array}
$$

Intuitively, we would expect $n{:}\,\Box\mathsf{N}$ for numeral n, since an explicit numeral has a fixed value, and therefore cannot implicitly define a fast-growing function of any input. However, $\Box\mathsf{N}$ itself is not a type. Instead, the typing rules of [Hof97] are formulated so that it is possible to apply a function of type $\Box\mathsf{N} \to \mathsf{N}$ (or $\Box\mathsf{N} \multimap \mathsf{N}$) to a numeral, since a numeral does not have any non-modal free variables.

Theorem B.1. *Any closed* $e{:}\,\Box\mathsf{N} \to \mathsf{N}$ *defines a function* $f{:}\,\mathcal{N} \times (\mathcal{N} \to \{0,1\}) \to \mathcal{N}$*. Specifically, there exists a standard Oracle Turing Machine* T *such that for any natural number* $x \in \mathcal{N}$ *and function* $S{:}\,\mathcal{N} \to \{0,1\}$ *representing a sequence of random bits, the function value* $f(x,S)$ *may be computed by* M *in time polynomial in* $|x|$.