# A probabilistic poly-time framework for protocol analysis

P. Lincoln*†
Computer Science Laboratory
SRI International

J. Mitchell*‡ M. Mitchell*§
Department of Computer Science
Stanford University

A. Scedrov*¶
Department of Mathematics
University of Pennsylvania

**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 form of process calculus where security may be expressed in terms of *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 of this 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 [15, 14], used to manage encrypted passwords, and the Secure Sockets Layer [12], used by internet browsers and servers to carry out secure internet transactions. Over the past decade or two, a variety of methods have been developed for analyzing and reasoning about such protocols. These approaches include specialized logics such as BAN logic [5], special-purpose tools designed for cryptographic protocol analysis [13], and theorem proving [26, 27] and model-checking methods using general purpose tools [16, 18, 23, 28, 29].

Although there are many differences among these approaches, most current approaches use the same basic model of adversary capabilities. This model, apparently derived from [10], treats cryptographic operations as "black-box" primitives. For example, encryption is generally considered a primitive operation, with plaintext and ciphertext treated as atomic data that cannot be decomposed into sequences of bits. In most uses of this model, as explained in [23, 26, 29], there are specific rules for how an adversary can learn new information. For example, if the decryption key is sent over the network "in the clear", it can be learned by the adversary. However, it is not possible for the adversary to learn the plaintext of an encrypted message unless the entire decryption key has already been learned. Generally, the adversary is treated as a nondeterministic process that may attempt any possible attack, and a protocol is considered secure if no possible interleaving of actions results in a security breach. The two basic assumptions of this model, perfect cryptography and nondeterministic adversary, provide an idealized setting in which protocol analysis becomes relatively tractable.

While there have been significant accomplishments using this model, the assumptions inherent in the standard model also make it possible to "verify" protocols that are in fact susceptible to attack. For example, the adversary is not allowed (by the model) to learn a decryption key by guessing it, since then some nondeterministic execution would allow a correct guess, and all protocols relying on encryption would be broken. However, in some real cases, adversaries can learn some bits of a key by statistical analysis, and can then exhaustively search the remaining (smaller) portion of the key space. Such an attack is simply not considered by the model described above, since it requires both knowledge of the particular encryption function involved and also the use of probabilistic methods.

Another way of understanding the limitations of common formal methods for protocol analysis is to consider the plight of someone implementing or installing a protocol. A protocol designer may design a protocol and prove that it is correct using the "black-box" cryptographic approach described above. However, an installed system must use a particular encryption function, or choice of encryption functions. Unfortunately, very few, if any, encryption functions satisfy all of the black-box assumptions. As a result, an implementation of a protocol may in fact be susceptible to attack, even though both the abstract protocol and the encryption function are individually correct.

Our goal is to establish an analysis framework that can be used to explore interactions between protocols and cryptographic primitives. In this paper, we set the stage for a form of protocol analysis that allows the analysis of these interactions as well as many other attacks not permitted in the standard model. Our framework uses a language for defining communicating probabilistic polynomial-time processes [22]. We restrict processes to probabilistic polynomial time so that we can say that a protocol is secure if there is

*Partially supported by DoD MURI "Semantic Consistency in Information Exchange," ONR Grant N00014-97-1-0505.
†Additional support from NSF CCR-9509931.
‡Additional support from NSF CCR-9629754.
§Additional support from Stanford University Fellowship.
¶Additional support from NSF Grant CCR-9800785.

no definable program which, when run in parallel with the protocol, causes a security breach. Establishing a bound on the running time of an adversary allows us to lift other restrictions on the behavior of an adversary. Specifically, an adversary may send randomly chosen messages, or perform sophisticated (yet probabilistic polynomial-time) computation to derive an attack from statistical analysis of messages overheard on the network. In addition, we treat messages as sequences of bits and allow specific encryption functions such as RSA or DES to be written in full as part of a protocol. An important feature 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 [11], for example, for which a single plaintext may have more than one ciphertext.

In our framework, following the work of Abadi and Gordon [1], security properties of a protocol $P$ may be formulated by writing an idealized protocol $Q$ so that, intuitively, for any adversary $M$, the interactions between $M$ and $P$ have the same observable behavior as the interactions between $M$ and $Q$. Following [1], this intuitive description may be formalized by using observational equivalence (also called observational congruence), a standard notion from the study of programming languages. Namely, two processes (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 adversaries, represented by the environments, that might interact with the protocol participants. Our framework is a refinement of this approach in that in our asymptotic formulation, observational equivalence between probabilistic polynomial-time processes coincides with the traditional notion of indistinguishability by polynomial-time statistical tests [17, 30], a standard way of characterizing cryptographically strong pseudo-random number generators.

## 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 foresee any fundamental difficulty in extending our basic methods to more general protocols that do not have a fixed bound set in advance, there are some technical complications that we avoid by making this simplifying assumption.

We will use a form of $\pi$-calculus (a general process calculus) [21] for defining protocols. One reason for using a precise 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 [1] 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 sequential expressions that do not perform any communication, and *processes,* which can communicate with one another. The process portion of the language is a restriction of standard $\pi$-calculus. All computation done by a process is expressed using terms. Since our goal is to model probabilistic polynomial-time adversaries by quantifying over processes definable in our language, it is essential that all functions definable by terms lie in probabilistic polynomial time.

Although we use pseudo-code to write terms in this paper, we have developed an applied, simply-typed lambda calculus which exactly captures the probabilistic polynomial-time terms. Our language is described in [22].

### 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 $\pi$-calculus, given by the following grammar:

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

### 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 $\overline{n}\langle M \rangle$ sends a message $M$ on the port $n$. In other words, it 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, there are no delivery guarantees in this model. Any process containing a read expression for a given port can read any message sent by any other process on that port. In particular, an adversary can read any public network message sent by any protocol participant.

Some readers may wonder why reading a message has the side-effect of removing it from the network. One reason is that we wish to allow an attacker to intercept messages without forwarding them to other parties. This may occur in practice when an attacker floods the subnet of a receiver. In addition, we may express passive reads, which do not remove messages from the network, as a combination of destructive read and resend. To make this precise, let us write $n_{pasv}(x).P$ as an abbreviation for $n(x).(\overline{n}\langle x\rangle \mid P)$. It is not hard to see that this definable combination of actions is equivalent to the intuitive notion of a passive read. For example, consider the process $\overline{x}\langle a\rangle \mid n_{pasv}(x).P \mid Q$ containing an output and a passive read. If the passive read is scheduled first, one computation step of this process leads to $\overline{x}\langle a\rangle \mid P[a/x] \mid Q$ which is what one would expect from a passive read primitive. Further details on the operational semantics of the process language appear in Appendix A.

### 2.5 Example using symbolic cryptosystem

For readers not familiar with $\pi$-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

$$\begin{array}{l} encrypt\colon plain \times key \to cipher \\ decrypt\colon cipher \times key \to plain \end{array}$$

We illustrate the 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 $B$.

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

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 $p_1$, $A$ 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 $\pi$-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 $p_2$.

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

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

A fundamental idea that we have adopted from spi-calculus [1] is that an intruder may be modeled by a process context, which is a process expression containing a hole indicating a place that may be filled by another process. Intuitively, we think of the context as the environment in which the process in the hole is executed. To give a specific example, consider the context

$$\mathcal{C}[\,] \;=\; [\,] \mid AB(x).\,\overline{AB}\langle encrypt(p_1, k_C)\rangle$$

where the empty square brackets [ ] indicate the hole for an additional process. If we insert a process $P$ in this context, the resulting process $\mathcal{C}[P]$ will run $AB(x).\,\overline{AB}\langle encrypt(p_1, k_C)\rangle$ in parallel with $P$. It is easy to see that if we insert the protocol above in this context, then the context could intercept the first message from $A$ to $B$ and replace it by another one using a different key.

### 2.6 Example

Our first example (continued in Section 4.1) is a simple protocol based on ElGamal Encryption [11] and Diffie-Hellman Key Exchange [8], 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

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

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 $\mathsf{msg}$ has been sent.

In $\pi$-calculus notation, the protocol may be written as follows. We use the convention that port $\overline{AB}_i$ is used for the $i$th 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.

$$\begin{array}{l} \text{let } p \text{ be a random } n\text{-bit prime and} \\ \quad g \text{ a generator of } \mathcal{Z}_p^* \\ \text{in } \overline{PUBLIC}\langle p\rangle \mid \overline{PUBLIC}\langle g\rangle \\ \mid\ \ \text{let } a \text{ be a random number in } [1, p-1] \end{array}$$

$$\begin{array}{l} \quad\text{in } \overline{\scriptstyle AB1}\langle g^a \mod p\rangle \\ \qquad \mid {\scriptstyle BA}(x).\,\overline{\scriptstyle AB2}\langle \mathsf{msg} * x^a \mod p\rangle \\ \mid \ \text{let } b \text{ be a random number in } [1, p-1] \\ \quad \text{in } {\scriptstyle AB1}(y).\,\overline{\scriptstyle BA}\langle g^b \mod p\rangle \end{array}$$

An analysis appears in Section 4.1.

### 2.7 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 $\mathsf{msg}$. 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

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

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 exactly $K_b$ cannot be any higher than the probability of randomly guessing $K_b$. Therefore, 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.

## 3 Process Equivalence

Observational equivalence, also called observational congruence, is a standard notion in the study of programming languages. We explain the general concept briefly, as it arises in a variety of programming languages.

The main idea is that the important features of a part of a program, such as a function declaration, processes or abstract data type, are exactly those properties that can be observed by embedding them in full programs that may produce observable output. To formalize this in a specific programming language $\mathcal{L}$, we assume the language definitions gives rise to 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 integer or string outputs. We also assume that there is some semantic evaluation relation $\stackrel{eval}{\leadsto}$, with $M \stackrel{eval}{\leadsto} v$ meaning that evaluation or execution 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 phrase 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. The main idea underlying the concept of observational equivalence is that the properties of a program phrase that matter in program construction are precisely the properties that can be observed by experiment. Phrases that give the same experimental results can be considered equivalent.

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 some number of occurrences of $P$ by $Q$.

For the process language considered in this paper, we are interested in contexts that distinguish between processes. (We will not need to consider observational equivalence of terms.) Therefore, the contexts of interest are process expressions with a "hole", given by the following grammar

$$\begin{array}{lcl} \mathcal{C}[\,] & ::= & [\,] \;\mid\; n(x).\,\mathcal{C}[\,] \;\mid\; P|\mathcal{C}[\,] \;\mid\; \mathcal{C}[\,]|Q \;\mid \\ & & \nu n.\,\mathcal{C}[\,] \;\mid\; [M=N]\mathcal{C}[\,] \;\mid\; \mathsf{let}\ x = M\ \mathsf{in}\ \mathcal{C}[\,] \end{array}$$

A process observation will be a communication event on a port whose name is not bound by $\nu$. More specifically, we let *Obs* be the set of pairs $\langle n, m\rangle$, where $n$ is a port name and $m$ is an integer, and write $P \stackrel{eval}{\leadsto} \langle n, m\rangle$ if evaluation of process expression $P$ leads to a state (represented by a process expression) of the form $\ldots|\overline{n}\langle m\rangle$ in which the process is prepared to communicate integer $m$ on port $n$ and $n$ is not within the scope of a binding $\nu n.$. (This can be made more precise using the structural equivalence relation in the Appendix.) In more general terms, $P \stackrel{eval}{\leadsto} v$ in our language if process $P$ publicly outputs $v$.

The general definition of $\simeq$ above is essentially standard for deterministic or nondeterministic functional, imperative or concurrent languages. Some additional considerations enter when we consider probabilistic languages. Drawing from standard notions in cryptography, we propose the following adaptation of observational equivalence to the probabilistic polynomial-time process language at hand.

Intuitively, given program phrases $P$ and $Q$, context $\mathcal{C}[\,]$ and observable action $v$, it seems reasonable to compare the probability that $\mathcal{C}[P] \overset{eval}{\rightsquigarrow} v$ to the probability that $\mathcal{C}[Q] \overset{eval}{\rightsquigarrow} v$. 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.

As a first step toward developing a workable notion of observable equivalence, we define computational indistinguishability within factor $\epsilon$ by saying that $P \simeq_\epsilon Q$ if

$$\begin{array}{l} \forall \mathcal{C}[\,].\ \forall v \in Obs. \\ |\mathrm{Prob}[\mathcal{C}[P] \overset{eval}{\rightsquigarrow} v] - \mathrm{Prob}[\mathcal{C}[Q] \overset{eval}{\rightsquigarrow} v]| \leq \epsilon \end{array}$$

An immediate difficulty with $\simeq_\epsilon$ is that it is not a transitive relation. Moreover, it is not clear how to differentiate large $\epsilon$ from small $\epsilon$. 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 random variables that differ by a large amount.

We can bring 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 required. For any protocol that begins by generating encryption and decryption keys, the security parameter is typically the number of bits used in the keys. If the key 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 $n$. Then for function $f$, we define *asymptotic equivalence within* $f$ by writing $P \simeq_f Q$ if

$$\begin{array}{l} \forall \mathcal{C}[\,].\ \forall v \in Obs.\ \exists n_0.\ \forall n \geq n_0. \\ |\mathrm{Prob}[\mathcal{C}_n[P_n] \overset{eval}{\rightsquigarrow} v] - \mathrm{Prob}[\mathcal{C}_n[Q_n] \overset{eval}{\rightsquigarrow} v]| \leq f(n) \end{array}$$

In words, $P$ and $Q$ are asymptotically equivalent within $f$ if, for every computational experiment given by a context family 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

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

It is easy to check that this (finally) is an equivalence relation. Moreover, we believe that this formal definition reasonably models the ability to distinguish two processes by feasible intervention and observation. The examples given in Section 4 provide some evidence for this thesis, and we hope that future work will further confirm our belief (or allow us to usefully refine the concept).

*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., [17, 30]). Specifically, $P \simeq Q$ iff $P$ and $Q$ pass the same polynomial-time statistical tests.

## 4 Specification of Security Properties

### 4.1 A variant of ElGamal Encryption

**Protocol** In section 2.6, we formulated a variant of ElGamal encryption as a three-step protocol and expressed this in probabilistic $\pi$-calculus. We can regard this protocol as a parameterized family of protocols by making the dependence on the length of the public prime (and therefore the key length) explicit:

Protocol $P_n$:

$$\begin{array}{ll} & \text{let } p \text{ be a random } n\text{-bit prime and} \\ & \quad g \text{ a generator of } \mathcal{Z}_p^* \\ \text{in} & \overline{PUBLIC}\langle p\rangle \mid \overline{PUBLIC}\langle g\rangle \\ \mid & \text{let } a \text{ be a random number in } [2, p-1] \\ & \quad \text{in } \overline{AB_1}\langle g^a \mod p\rangle \\ & \quad \mid BA(x).\,\overline{AB_2}\langle \mathsf{msg} * x^a \mod p\rangle \\ \mid & \text{let } b \text{ be a random number in } [2, p-1] \\ & \quad \text{in } AB_1(y).\,\overline{BA}\langle g^b \mod p\rangle \end{array}$$

Note that although the algorithm for generating a prime $p$ and generator $g$ may be probabilistic and may fail with small probability (see, e.g., [19]), our specification will also contain the same algorithm, compensating for this minor difficulty.

This protocol is easily defeated if an adversary intercepts $g^b$ from Bob and sends $g^c$ instead. In this case, Alice will send $\mathsf{msg} * g^{ac} \mod p$. Since the adversary will know $g^a$ and $c$, the plaintext can discovered. 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.

**Specification** In general, we specify the intended security properties of a protocol $P$ by writing an idealized protocol $Q$ so that $P \simeq Q$ will imply the intended properties. If $P$ is intended to send some data securely, then $Q$ could send random numbers (noise) instead and use a private port to communicate the same information if needed. In this case, $P \simeq Q$ will imply that no adversary can uncover the secret, since the adversary would have no chance to uncover the secret from any run of $Q$.

For the simple protocol $P$ above, we wish to specify a weak security property, namely, that $\mathsf{msg}$ is transmitted secretly from Alice to Bob in the presence of any passive adversary. This requires a restriction of our general approach. However, since the proof of correctness is simplified by the restriction on adversaries, this seems like an appropriate first example.

Our specification is written using an idealized protocol that clearly cannot reveal $\mathsf{msg}$ to an adversary:

Protocol $Q_n$:

$$
\begin{array}{l}
\text{let } p \text{ be a random } n\text{-bit prime and} \\
\quad g \text{ a generator of } \mathcal{Z}_p^* \\
\text{in } \overline{\mathit{PUBLIC}}\langle p\rangle \mid \overline{\mathit{PUBLIC}}\langle g\rangle \\
\quad \mid \text{let } a \in_r [2, p-1] \text{ in } \overline{AB_1}\langle a\rangle \\
\quad \mid \text{let } a \in_r [2, p-1] \text{ in } BA(x).\,\overline{AB_2}\langle c\rangle \\
\quad \mid \text{let } b \in_r [2, p-1] \text{ in } AB_1(y).\,\overline{BA}\langle b\rangle
\end{array}
$$

In the idealized protocol $Q_n$, each secret message in $P_n$ is replaced by a random number in the appropriate range. Intuitively, our aim is to specify that to any other process observing $P_n$, the network traffic appears to be a series of encrypted random numbers. The reason we send encrypted random numbers in $Q_n$ 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 secrecy is that the content of the messages must be hidden. While we could also specify that no observer can tell that encryption is used, we choose not to require this stronger property.

The next part of our specification requires a definition of equivalence with passive observers. First, a process $P$ *passively reads port n* if every subexpression of $P$ that mentions port n is of the form $n_{pasv}(x)$.. We say $\mathcal{C}[\ ]$ is a *passive observer of ports* $n_1, \ldots, n_k$ if every process expression in $\mathcal{C}[\ ]$ passively reads $n_1, \ldots, n_k$. Finally, we define passive observational congruence by limiting the definition of observational congruence to passive contexts. Specifically, we say $P$ and $Q$ are passively indistinguishable within function $f$ if

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

for all context families such that each $\mathcal{C}_n[\ ]$ is a passive observer of all ports that occur free (not bound by $\nu$) in $P_n$ and $Q_n$. We say $P$ and $Q$ are passively indistinguishable, and write $P \simeq_{passive} Q$ is if $P$ and $Q$ are passively indistinguishable within function $1/p(n)$, for every polynomial $p$.

Our secrecy specification for protocol family $P$ above is that $P \simeq_{passive} Q$, where $Q$ is the protocol family above that sends encrypted random numbers in place of data exchanged by $P$.

**Equivalent Game** We cannot hope to prove $P \simeq_{passive} Q$ without establishing significant new results about the difficulty of computing discrete logarithms. Instead, we will prove that any context that passively distinguishes $P$ and $Q$ provides a method for asymptotically winning the following family of games, based on the decision form of Diffie-Hellman (see [24]):

Game $G_n$:

> Player $A$: Announces a prime and generator $\langle g, p\rangle$ 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 numbers of the form $\langle g^u, g^v, g^{uv}\rangle \bmod p$, with $u$ and $v$ chosen randomly.
>
> Player $B$: Chooses one of the triples, winning the game if the triple has the form $\langle g^u, g^v, g^{uv}\rangle$.

For simplicity, we assume that prime $p$, generator $g$, and random $a, b, c, u, v$ are chosen according to exactly the same distribution as in runs of $P_n$ and $Q_n$.

Intuitively, the first two numbers of a triple $\langle g^u, g^v, g^{uv}\rangle$ computed as above will chosen randomly from 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** We outline the proof that the protocol is correct. Specifically, if there is a context passively distinguishing $P$ from $Q$, then there is an probabilitistic polynomial-time strategy for winning the game with related probability.

Since $[2, p-1]$ is a multiplicative group, multiplication by $\mathsf{msg}$ is simply a permutation. This allows us to argue that 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 passive observer contexts and $v \in \mathit{Obs}$ an observable such that

$$
\begin{array}{l}
\forall n_0.\, \exists n \geq n_0. \\
|\mathrm{Prob}[\mathcal{C}_n[P_n] \overset{eval}{\leadsto} v] - \mathrm{Prob}[\mathcal{C}_n[Q_n] \overset{eval}{\leadsto} v]| > 1/\mathit{poly}(n)
\end{array}
$$

and let $n$ be any number where the difference in probabilities is greater than $1/\mathit{poly}(n)$. Let $\langle a, b, c\rangle$ and $\langle a', b', c'\rangle$ 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 do this using a process template

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

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

For cards $\langle a, b, c\rangle$ and $\langle a', b', c'\rangle$ dealt from $G'_n$, one of $R(p, g, a, b, c)$, $R(p, g, a', b', c')$ will behave like a run of $P_n$ and the other a run of $Q_n$. However, our assumption is that that 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 $1/\mathit{poly}(n)$. This gives us a method for choosing which of the triples $\langle a, b, c\rangle$ and $\langle a', b', c'\rangle$ has the form $\langle g^u, g^v, g^{uv}\rangle$. For the purpose of simplifying the argument, let us assume that $\mathrm{Prob}[\mathcal{C}_n[P_n] \overset{eval}{\leadsto} v] > \mathrm{Prob}[\mathcal{C}_n[Q_n] \overset{eval}{\leadsto} v]$. The opposite case is symmetric.

To decide probabilistically which of the triples $\langle a, b, c\rangle$ and $\langle a', b', c'\rangle$ has form $\langle g^u, g^v, g^{uv}\rangle$, we run $R(p, g, a, b, c)$ and $R(p, g, a', b', c')$ to completion; this requires time bounded by a polynomial in $n$. If observable $v$ occurs in both runs, or in neither, then we have no useful information. Therefore, we flip a coin and choose among triples $\langle a, b, c\rangle$ and $\langle a', b', c'\rangle$ with equal probability. Otherwise, we rationally suspect that the triple used in the the process producing observable $v$ is more likely to have the form $\langle g^u, g^v, g^{uv}\rangle$ and choose this triple. A simple calculation, using the fact that the distribution of cards in the game is the same as the distribution of triples generated by runs of $P_n$ and $Q_n$, reveals that the probability of choosing the correct triple is $1/(2poly(n))$. This completes the proof.

### 4.2 Part of Needham-Schroeder Private-Key Protocol

Our second example involves authentication as well as secrecy and uses arbitrary contexts that may intercept and replace network messages.

**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 $k$. Alice chooses a random binary string $i$ of length $n$ and Bob uses some numerical function $f$ computable in probabilistic polynomial time to respond.

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

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 $P_n$:

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

This process calculus expression contains three sequential processes, each corresponding to one step of the protocol. The first sends $\mathit{encrypt}(k,i)$ on port $AB$, representing communication from Alice to Bob. In the second process, "Bob" receives input $x$ and transmits $\mathit{encrypt}(k,\ f(\mathit{decrypt}(k,x)))$. In the third process, "Alice" checks whether the message $y$ she receives from Bob is what she expected. (This check should be written slightly differently if the encryption function is probabilistic instead of deterministic.)

**Specification** Our specification is given using a similar family of processes, sending encrypted random numbers in place of the data sent in the protocol. We specify a form of authentication by causing the protocol to halt if a message is altered. The specification process accomplishes this by using a private channel $s$ to send the data received by Bob back to Alice securely. In addition, we specify that $i$ must be kept secret. (For example, $i$ may be a session key.) These decisions about what should be secret and what should be authentic are not made explicit in the usual notation, but must be established before correctness can be shown.

Protocol $Q_n$:

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

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 this case $S$) 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** We show that if a context $\mathcal{C}[\;]$ asymptotically distinguishes $P$ from $Q$, then this context provides a strategy for winning a number-theoretic game related to the encryption function and the function $f$, so far unspecified, used in the protocol. To illustrate some of the reasoning involved, we break this down into various cases, depending on the kind of "attacks" used in the context family.

Suppose $\mathcal{C}[\;]$ asymptotically distinguishes $P$ from $Q$ and $v \in \mathit{Obs}$ is an observable such that

$$\forall n_0.\, \exists n \geq n_0. \\ |\text{Prob}[\mathcal{C}_n[P_n] \overset{eval}{\leadsto} v] - \text{Prob}[\mathcal{C}_n[Q_n] \overset{eval}{\leadsto} v]| > 1/poly(n)$$

Assume for simplicity that the first probability is greater than the second and let $n$ be any number where the difference in probabilities is greater than $1/poly(n)$.

In outline, we consider all possible executions (traces) of $\mathcal{C}_n[P_n]$ that produce observable $v$, each with an associated probability. For some runs of $\mathcal{C}_n[P_n]$, we can find a corresponding run of $\mathcal{C}_n[Q_n]$, with exactly the same communication on ports $AB$ and $BA$, and $v$ is also produced. However, since it is the same context in both cases, and the branching structures of $P_n$ and $Q_n$ are similar, we can show that there must be some number of traces of $\mathcal{C}_n[P_n] \overset{eval}{\leadsto} v$ that do not correspond to any trace of $\mathcal{C}_n[Q_n]$. The total sum of probabilities associated with these traces is at least $1/poly(n)$. Furthermore, these traces can be divided into three cases:

(i) The context does not alter any communication on $AB$ or $BA$. This is a passive attack in which the context observes something different about the behavior of $P_n$ from $Q_n$.

(ii) The context alters at least one communication on $AB$ or $BA$ in such a way that neither $P_n$ nor $Q_n$ would issue $\overline{AB}\langle \text{“OK}''\rangle$. This is an active attack in which the protocol is aborted, but the context still observes some difference between $P_n$ and $Q_n$.

(iii) The context alters at least one communication on $AB$ or $BA$ in such a way that $P_n$ produces $\overline{AB}\langle \text{“OK}''\rangle$ but $Q_n$ cannot. This is an active attack in which the protocol is subverted so that one principal commits when it should not.

This division into cases applies to any $n$ where the probabilities of $\mathcal{C}_n[P_n] \overset{eval}{\leadsto} v$ and $\mathcal{C}_n[Q_n] \overset{eval}{\leadsto} v$ differ by $1/poly(n)$. Since there are infinitely many such $n$, there must be at least on case that arises with probability $1/(3poly(n))$ for infinitely many choices of $n$. Therefore, we can show that if there is a context asymptotically distinguishing $P$ from $Q$, there is a strategy for winning at least one of three possible games. This allows us to formulate somewhat simpler games than if we did not subdivide the set of possible attacks.

If case (i) arises infinitely often, then as with the previous example, we can obtain a reduction to a recognition game. In particular, the adversary will be able to tell a pair $\{i\}_k, \{f(i)\}_k$ from a pair $\{i\}_k, \{j\}_k$, for $i, j, k$ chosen randomly as in protocol $P$ and specification $Q$.

If case (ii) arises infinitely often, then the adversary is able to chose a function $g$ such that it can probabilistically distinguish a pair $\{i\}_k, \{f(decrypt(k, g(encrypt(k, i))))\}_k$ from a pair $\{i\}_k, \{j\}_k$. This is an interesting form of active observation of properties of the encryption function. Such observable properties could conceivably be used to assemble some useful statistical information about the choice of nonces or the behavior of function $f$ under encryption.

Finally, if case (iii) arises infinitely often, then the adversary must expose a malleability property of the encryption function [9]. Specifically, for some functions $g$ and $h$ computed by the adversary, at least one different from the identity function, we have

$$decrypt(k, h(\{f(decrypt(k, g(\{i\}_k)))\}_k, \{i\}_k)) = f(i)$$

This imposes an interesting condition on the relation between $f$ and the encryption function. A realistic scenario in which this form of attack is possible arises with RSA encryption [19] and $f(x) = 2x$. Since $\{2i\}_k = \{2\}_k\{i\}_k$, the intruder may take $g(x) = \{2\}_k x$ and $h(x, y) = g(y)$. In simple terms, this attack will blind Bob into thinking that Alice's nonce is $2i$, rather than $i$.

## 5 Comparison with related work

The framework described in this paper may be regarded as a probabilistic, polynomial-time variant of spi-calculus [1], with cryptographic primitives expressed directly in the probabilistic polynomial-time expression language instead of by the $\nu$ operator and additional primitives. Thus our framework differs from others in two ways: the use of a process calculus and observational equivalence to express security properties (as in the spi-calculus), and the probabilistic polynomial-time treatment of cryptographic primitives (in contrast to the spi-calculus). We view the spi-calculus as a convenient means to an end, a useful setting for expressing protocols and security properties. Our main goal has been to develop a formal framework for studying protocols under complexity-theoretic assumptions, rather than the traditional "perfect cryptography" assumptions associated with most logic- or linguistic-based approaches.

The closest work in the direction of protocol analysis under complexity-theoretic assumptions is a series of protocol studies by Bellare and Rogaway [2, 3]. In these studies, a protocol is represented as a set of oracles, each corresponding to one input-output step of one principal. These oracles are each available to the adversary, which is a probabilistic polynomial-time oracle Turing machine. This corresponds fairly closely to our setting, since an adversary has access to each input-output step by a principal by sending and receiving data on the appropriate ports. The main differences are that in our setting, the protocol and the adversary are both expressed in a formal language. This opens the possibility for proof techniques that are based on the syntactic structure of the protocol or semantic properties of all expressible adversaries. In addition, we have found the specification method adopted from spi-calculus relatively natural and more systematic than the protocol specifications used by Bellare and Rogaway.

Among approaches based on specific languages for defining protocols and intruders, the distinguishing feature of our effort is the use of probability and polynomial time in place of a conventional model of "perfect cryptography" that appears to have developed from positions taken by Needham and Schroeder [25] and a model presented by Dolev and Yao [10]. In the conventional model, an intruder may intercept or block communication, remember parts of messages, and construct new messages from data it has observed. These are similar to the capabilities of our intruders. The difference lies in precisely what messages can be constructed from data observed by the intruder. In the conventional model, the adversary is only allowed to concatenate or (possibly) encrypt data it has observed, or been given at the outset, or obtained by decryption with keys obtained in these limited ways. The adversary cannot make random guesses, accumulate information by statistical analysis of network traffic, or use partial information in other sophisticated ways. The general trade-off is this: the conventional model makes it possible to automatically search for attacks with some efficiency, or formally prove protocols correct. However, some attacks lie outside the model. In our approach (as in the work of Bellare and Rogaway described above), it is much harder to prove protocols correct, but the model encompasses a much wider range of possible attacks.

## 6 Conclusion and Future Directions

We introduce a framework for security protocol analysis that can account for interactions between protocols and the underlying cryptography. This allows us to refine protocol analysis beyond the basic model of adversary capabilities [10], which treats cryptographic operations as primitive. Our framework uses a process calculus for defining 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. Because every process definable in our framework is probabilistic polynomial time, this makes it possible to express security properties of a protocol in terms of its interactions with other definable processes in the calculus.

More precisely, security properties of a given protocol may be formulated in our framework by writing another, idealized protocol and showing that the environment behaviors, which represent definable adversaries, have the same observable interactions with either protocol. For this purpose we propose a definition of observational equivalence for probabilistic programs that is based on the view that large differences in probability are easier to observe than small differences. When we distinguish between "large" and "small" using asymptotic behavior, we arrive at a definition of observational equivalence that coincides with a standard concept from cryptography, namely, indistinguishability by polynomial-time statistical tests [30]. 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 [9].

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 [1], 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.

*Acknowledgements:* Thanks to M. Abadi, D. Boneh, C. Dwork, S. Kannan, and M. Naor for helpful discussions.

### References

[1] Abadi, M., and Gordon, A. A calculus for cryptographic protocols: the spi calculus. In *Proc. 4th ACM Conference on Computer and Communications Security* (1997), pp. 36–47. Revised and expanded versions to appear in *Information and Computation* and as SRC Research Report 149 (January 1998).

[2] Bellare, M., and Rogaway, P. Entity authentication and key distribution. In *Advances in Cryptology - CRYPTO '93, Lecture Notes in Computer Science, Vol. 773* (1994).

[3] Bellare, M., and Rogaway, P. Provably secure session key distribution - the three party case. In *Proc. 27th ACM Symposium on the Theory of Computing* (1995).

[4] Berry, G., and G. Boudol. The chemical abstract machine. In *Proc. 17th ACM Symp. Principles of Programming Languages* (1990), pp. 81–94.

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

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

[7] Derman, C. *Finite-state Markov decision processes.* Academic Press, 1970.

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

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

[10] Dolev, D., and Yao, A. On the security of public-key protocols. *IEEE Transactions on Information Theory 2*, 29 (1983).

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

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

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

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

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

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

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

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

[19] Menzies, A., van Oorschot, P., and Vanstone, S. *Handbook of Applied Cryptography.* CRC Press, 1997.

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

[21] Milner, R., Parrow, J., and Walker, D. A calculus of mobile processes, part i. *Information and Computation 100*, 1 (1992), 1–40.

[22] Mitchell, J., Mitchell, M., and Scedrov, A. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *IEEE Symp. Foundations of Computer Science* (1998). To appear.

[23] Mitchell, J., Mitchell, M., and Stern, U. Automated analysis of cryptographic protocols using Mur$\varphi$. In *Proc. IEEE Symp. Security and Privacy* (1997), pp. 141–151.

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

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

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

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

[28] Roscoe, A. W. Modelling and verifying key-exchange protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop* (1995), IEEE Computer Soc Press, pp. 98–107.

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

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

## A Operational Semantics

We use an operational semantics similar in outline to the spi-calculus semantics given in [1]. This is a variant of Milner's reaction relation approach [20], inspired by the Chemical Abstract Machine of Berry and Boudol [4]. 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 $\to$ 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.

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

$$\text{(Symm)} \quad \frac{P \equiv Q}{Q \equiv P} \qquad \text{(Par)} \quad \frac{P \equiv P'}{P|Q \equiv P'|Q}$$

$$\text{(Trans)} \quad \frac{P \equiv Q \quad Q \equiv R}{P \equiv R} \qquad \text{(Res)} \quad \frac{P \equiv P'}{\nu n.\, P \equiv \nu n.\, P'}$$

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

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

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 [6, 7]), 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 [6, 7]. 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 $P$.*