Probabilistic Polynomial-Time Semantics for a Protocol Security Logic*

Anupam Datta[1], Ante Derek[1], John C. Mitchell[1],
Vitaly Shmatikov[2], and Mathieu Turuani[3]

[1] Dept. Computer Science, Stanford University, Stanford, CA
[2] Dept. Computer Science, University of Texas, Austin, TX
[3] LORIA-INRIA Nancy, France

Abstract. We describe a cryptographically sound formal logic for proving protocol security properties without explicitly reasoning about probability, asymptotic complexity, or the actions of a malicious attacker. The approach rests on a new probabilistic, polynomial-time semantics for an existing protocol security logic, replacing an earlier semantics that uses nondeterministic symbolic evaluation. While the basic form of the protocol logic remains unchanged from previous work, there are some interesting technical problems involving the difference between efficiently recognizing and efficiently producing a value, and involving a reinterpretation of standard logical connectives that seems necessary to support certain forms of reasoning.

1 Introduction

Security analysis of network protocols is a successful scientific area with two important but historically independent foundations, one based on logic and symbolic computation, and one based on computational complexity theory. The symbolic approach, which uses a highly idealized representation of cryptographic primitives, has been a successful basis for formal logics and automated tools. Conversely, the computational approach yields more insight into the strength and vulnerabilities of protocols, but it is more difficult to apply and it involves explicit reasoning about probability and computational complexity. The purpose of this paper is to suggest that formal reasoning, based on an abstract treatment of cryptographic primitives, can be used to reason about probabilistic polynomial-time protocols in the face of probabilistic polynomial-time attacks.

* This work was partially supported by NSF CyberTrust Grant 0430594, Collaborative research: High-fidelity methods for security protocols, by the DoD University Research Initiative (URI) program administered by the Office of Naval Research under Grant N00014-01-1-0795, by OSD/ONR CIP/SW URI through ONR Grant N00014-04-1-0725, by NSF CCR-0121403, Computational Logic Tools for Research and Education, and by the NSF Cybertrust grant to the PORTIA project. M. Turuani's activities at Stanford were also funded by a postdoctoral grant from INRIA.

L. Caires et al. (Eds.): ICALP 2005, LNCS 3580, pp. 16–29, 2005.
© Springer-Verlag Berlin Heidelberg 2005

We do this by proposing a new semantics for a variant of an existing logic. The new semantics brings forward some interesting distinctions that were not available in the coarser symbolic model, and also raises some apparently fundamental issues about the inherent logic of asymptotic probabilistic properties.

The *Protocol Composition Logic* [2, 7, 8, 10] uses a modal operator similar to Floyd-Hoare logic. Intuitively, the formula $\psi\,[P]_X\,\varphi$ means that if ψ is true at some point in the execution of a protocol (in the presence of a malicious attacker), then φ will be true after agent X performs the sequence P of actions. The pre- and post-conditions may describe actions taken by various principals and characterize the information that is available to or hidden from them. The semantics we explore in this paper recasts the methods of [15] in a logical setting, and reflects accepted modelling approaches used in the field of cryptography, particularly [5, 17].

Our central organizing idea is to interpret formulas as operators on probability distributions on traces. Informally, representing a probability distribution by a set of equi-probable traces (each tagged by the random sequence used to produce it), the meaning of a formula φ on a set T of traces is the subset $T' \subseteq T$ in which φ holds. This interpretation yields a probability: the probability that φ holds is the ratio $|T'|/|T|$. Conjunction and disjunction are simply intersection and union. There are several possible interpretations for implication, and it is not clear at this point which will prove most fruitful in the long run. In the present paper, we interpret $\varphi \implies \psi$ as the union of $\neg\varphi$ and the composition of ψ with φ; the latter is also the conditional probability of ψ given φ. This interpretation supports a soundness proof for a sizable fragment of the protocol logic, and resembles the probabilistic interpretation of implication in [16]. Since the logic does not mention probability explicitly, we consider a formula "true" if it holds with asymptotically overwhelming probability.

In previous work [2, 7, 8, 10] over a symbolic semantic model, the atomic formula $\mathsf{Has}(X, m)$ means that m is in the set of values "derivable," by a simple fixed algorithm, from information visible to X. The simple fixed algorithm is central to what is called the Dolev-Yao model, after [9] and much subsequent work by others. In replacing the symbolic semantics with a computational semantics based on probabilistic polynomial time, we replace the predicate Has with two predicates, Possess and Indist. Intuitively, $\mathsf{Possess}(X, m)$ means that there is an algorithm that computes the value of m with high probability from information available to X, while $\mathsf{Indist}(X, m)$ means that X cannot feasibly distinguish m from a random value chosen according to the same distribution. However, certain technical problems discussed in Section 7 lead us to work with slightly simplified semantics of these predicates that capture our intuition most strongly when the possessing principal is assumed honest (in the sense of following the protocol) and the predicate Indist only appears with positive polarity. Fortunately, these syntactic conditions are met in many formulas expressing authentication and secrecy properties.

Several groups of researchers have either formulated connections between symbolic logic and feasible probabilistic computation, or developed relationships

between symbolic and computational models. In particular, Abadi and Rogaway [1] propose a logical characterization of indistinguishability by passive eavesdroppers that has been studied by a number of others, and Kapron and Impagliazzo suggest a formal logic for reasoning about probabilistic polynomial-time indistinguishability [13]. Some semantic connections between symbolic and computational models have been developed by a team at IBM Zurich, *e.g.,* [3], with other connections explored in a series of related papers by Micciancio, Warinschi, and collaborators [15, 18, 6]. Herzog [11, 12] shows that if a protocol attack exists in a Dolev-Yao model, there is an attack in a computational model. More recent related work also appears in [14, 6].

Section 2 presents the syntax for defining roles of a protocol, while the syntax of the logic appears in Section 3. Some axioms and proof rules are described in Section 4, followed by a short proof example in Section 5. Section 6 presents the probabilistic polynomial-time execution and attacker model. The semantics of the logic are given in Section 7, and concluding remarks in Section 8.

2 Protocol Syntax

We use a simple "protocol programming language" based on [10, 7, 8] to represent a protocol by a set of roles, such as "Initiator", "Responder" or "Server", each specifying a sequence of actions to be executed by a honest participant. The syntax of terms and actions is given in Table 1.

Names, sessions and threads: We use $\hat{X}, \hat{Y}, \ldots$ as *names* for protocol participants. Since a particular participant might be involved in more than one session at a time, we will give unique names to sessions and use $(\hat{X}, s)$ to designate a particular *thread* being executed by $\hat{X}$. All threads of a participant $\hat{X}$ share the same asymmetric key denoted by X. As a notational convenience, we will sometimes write $\tilde{X}$ for an arbitrary thread of $\hat{X}$.

Terms, actions, and action lists: Terms name messages and their parts, such as nonces, keys, variables and pairs. For technical reasons, we distinguish *basic terms* from *terms* that may contain encryption. To account for probabilistic encryption, encrypted terms explicitly identify the randomness used for encryption.

Table 1. Syntax of protocol terms and actions

Terms:		**Actions:**
$N ::= \hat{X}$	(*name*)	$\mathtt{a} ::=$
$K ::= X$	(*key*)	$\mid \ \mathtt{new}\ T, n$
$S ::= s$	(*session*)	$\mid \ V := \mathtt{enc}\ T, t, K$
$n ::= r$	(*nonce*)	$\mid \ V := \mathtt{dec}\ T, t, K$
$T ::= (N, S)$	(*thread*)	$\mid \ \mathtt{match}\ T, t/t$
$V ::= x$	(*term variable*)	$\mid \ \mathtt{send}\ T, t$
$t_B ::= V \mid K \mid T \mid N \mid n \mid \langle t_B, t_B \rangle$	(*basic term*)	$\mid \ \mathtt{receive}\ T, V$
$t ::= t_B \mid \{t\}^n_K \mid \langle t, t \rangle$	(*term*)	

Specifically, $\{t\}_K^n$ indicates the encryption of t with key K using randomness n generated for the purpose of encryption. We write $m \sqsubseteq m'$ when m is a subterm of $m' \in t$.

Actions include nonce generation, encryption, decryption, pattern matching, and communication steps (sending and receiving). An *ActionList* consists of a sequence of actions that contain only basic terms. This means that encryption cannot be performed implicitly; explicit enc actions, written as assignment, must be used instead. We assume that each variable will be assigned at most once, at its first occurrence. For any $s \in \mathit{ActionList}$, we write $s_{|X}$ to denote the subsequence of s containing only actions of a participant (or a thread) X.

Strands, roles, protocols and execution: A *strand* is an *ActionList*, containing actions of only one thread. Typically we will use notation $[\mathit{ActionList}]_{\tilde{X}}$ to denote a strand executed by thread $\tilde{X}$ and drop the thread identifier from the actions themselves. A *role* is a strand together with a basic term representing the initial knowledge of the thread. A *protocol* is a finite set of *Roles*, together with a basic term representing the initial intruder knowledge.

An *execution strand* is a pair $\mathit{ExecStrand} ::= \mathit{InitialState}(\mathcal{I}); \mathit{ActionList}$ where $\mathcal{I}$ is a data structure representing the initial state of the protocol, as produced by the initialization phase from Section 6. In particular, this includes the list of agents and threads, the public/private keys and honesty/dishonesty tokens of each agent, and the roles played by each thread.

3 Logic Syntax

The syntax of formulas is given in Table 2. Protocol proofs will usually use modal formulas of the form $\psi[P]_{\tilde{X}}\varphi$, as explained intuitively in the introduction of the paper. Most formulas have the same intuitive meaning in the computational semantics as in the symbolic model [7, 8], except for predicates Possess and Indist. We summarize the meaning of formulas informally below, with precise semantics in the next section.

Action Predicates:
$\mathsf{a} ::= \mathsf{Send}(T, t) \mid \mathsf{Receive}(T, t) \mid \mathsf{New}(T, n)$

Formulas:
$\varphi ::= \mathsf{a} \mid t = t \mid \mathsf{Start}(T) \mid \mathsf{Possess}(T,\, t) \mid \mathsf{Indist}(T,\, t) \mid \mathsf{Fresh}(T,\, t) \mid \mathsf{Honest}(N) \mid$
$\mathsf{Start}(T) \mid \mathsf{Contains}(t,\, t) \mid \mathsf{ContainsOut}(t,\, t,\, t) \mid \mathsf{DecryptsHonest}(T,\, t) \mid$
$\mathsf{Source}(T,\, t,\, t) \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists \mathit{Var}.\, \varphi \mid \forall \mathit{Var}.\, \varphi \mid \neg\varphi \mid \varphi \supset \varphi \mid \varphi \Rightarrow \varphi$

Modal formulas:
$\Psi ::= \varphi\ [\mathit{Strand}]_T\ \varphi$

Table 2. Syntax of the logic

For every protocol action, there is a corresponding action predicate which asserts that the action has occurred in the run. For example, $\mathsf{Send}(\tilde{X}, t)$ holds in a run where the thread $\tilde{X}$ has sent the term t. $\mathsf{Fresh}(\tilde{X}, t)$ means that the value of t generated by $\tilde{X}$ is "fresh" in the sense that no one else has seen any messages containing t, while $\mathsf{Honest}(\hat{X})$ means that $\hat{X}$ is acting honestly, *i.e.,* the actions of every thread of $\hat{X}$ precisely follows some role of the protocol. The Source predicate is used to reason about the source of a piece of information, such as a nonce. Intuitively, the formula $\mathsf{Source}(\tilde{Y}, u, \{m\}^r_X)$ means that the only way for a thread $\tilde{X}$ different from $\tilde{Y}$ to know u is to learn u from the term $\{m\}^r_X$, possibly by some indirect path.

The predicate Fresh is definable by $\mathsf{Fresh}(\tilde{X}, v) \equiv \mathsf{New}(\tilde{X}, v) \wedge \neg(\exists u.\ \mathsf{Send}(\tilde{X}, u) \wedge \mathsf{Contains}(u, v))$ and classical implication is definable by $A \supset B \equiv \neg A \vee B$.

In the symbolic model [7, 8],the predicate Has states that a principal can "derive" a message or its contents from the information gathered during protocol execution. We use $\mathsf{Possess}(\tilde{X}, t)$ to state that it is possible to derive t by Dolev-Yao rules from $\tilde{X}$'s view of the run and $\mathsf{Indist}(\tilde{X}, t)$ to state that no probabilistic polynomial-time algorithm, given $\tilde{X}$'s view of the run, can distinguish t from a random value from the same distribution. Typically, we use $\mathsf{Possess}$ to say that some honest party obtained some secret, and Indist to say that the attacker does not have any partial information about a secret.

4 Proof System

The proof system used in this paper is based on the proof system developed in [7, 8, 2]. Some example axioms and rules are given in Table 3; the full presentation is deferred to the extended version of this paper. These axioms express reasoning principles that can be justified using complexity-theoretic reductions, information-theoretic arguments, and asymptotic calculations. However, the advantage of the proof system is that its justification using cryptographic-style arguments is a one-time mathematical effort; protocol proofs can be carried out symbolically using the proof system without explicitly reasoning about probability and complexity. Another advantage of the axiomatic approach is that different axioms and rules rest on different cryptographic assumptions. Therefore, by examining the axioms and rules used in a specific proof, we can identify specific properties of the cryptographic primitives that are needed to guarantee protocol correctness. This provides useful information in protocol design because primitives that provide weaker properties often have more efficient constructions.

Axioms: Axioms **AN2** and **AN3** capture some of the properties of nonce generation. Informally, **AN2** states that if a thread $\tilde{X}$ generates a fresh nonce x and does not perform any additional actions, then x is indistinguishable from a random value for all other threads. The soundness of this axiom is established by a simple information-theoretic argument. The informal interpretation of axiom **S1** (also called the "Source" axiom) is that, unless a ciphertext is decrypted, a thread which does not possess the decryption key cannot extract any par-

Axioms:

$$\mathbf{AN2} : \top[\ \texttt{new}\ x]_{\tilde{X}} \tilde{Y} \neq \tilde{X} \Rightarrow \mathsf{Indist}(\tilde{Y}, x)$$
$$\mathbf{AN3} : \top[\ \texttt{new}\ x]_{\tilde{X}} \mathsf{Fresh}(\tilde{X}, x)$$
$$\mathbf{S1} : \mathsf{Source}(\tilde{Y}, u, \{m\}^r_X) \wedge \neg\mathsf{DecryptsHonest}(\hat{X}, \{m\}^r_X) \wedge \hat{Z} \neq \hat{X} \wedge \hat{Z} \neq \hat{Y} \wedge \mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \Rightarrow \mathsf{Indist}(\tilde{Z}, u)$$

Proof rules:

$$\frac{\theta[P]_X\varphi \quad \theta' \supset \theta \quad \varphi \supset \varphi'}{\theta'[P]_X\varphi'}\ \mathbf{G3} \qquad \frac{\theta[P_1]_X\varphi \quad \varphi[P_2]_X\psi}{\theta[P_1P_2]_X\psi}\ \mathbf{SEQ}$$

$$\frac{\varphi \quad \varphi \Rightarrow \psi}{\psi}\ \mathbf{MP} \qquad \frac{\varphi}{\forall x.\varphi}\ \mathbf{GEN}$$

Table 3. Fragment of the proof system

tial information about the plaintext. The soundness of this axiom is proved by a complexity-theoretic reduction. Specifically, we show that if an attacker can break this property, then there is another attacker that can break the underlying IND-CCA2 secure encryption scheme [4].

Inference rules: Inference rules include generic rules from modal logics (e.g. **G3**), sequencing rule **SEQ** used for reasoning about sequential composition of protocol actions and a rule (called the honesty rule) for proving protocol invariants using induction. These rules are analogous to proof rules from our earlier work [7, 8].

First-order axioms and rules: We use two implications: a conditional implication $\Rightarrow$, discussed and defined precisely in section 7, and a classical implication $\supset$ with $A \supset B \equiv \neg A \vee B$. While standard classical tautologies hold for classical implication, some familiar propositional or first-order tautologies may not hold when written using $\Rightarrow$ instead of $\supset$. However, modus ponens and the generalization rule above are sound. The soundness of modus ponens relies on the simple asymptotic fact that the sum of two negligible functions is a negligible function. In future work, we hope to develop a more complete proof system for the first-order fragment of this logic.

5 Example

In this section, we present a simple protocol and state a secrecy property that can be proved using the proof system. The interested reader is referred to [10, 7, 8] for further explanation and examples. The two protocol roles are:

$$\mathbf{Init} \equiv [\ \texttt{new}\ x;\ y := \texttt{enc}\langle x, \tilde{X}\rangle, Y;\ \ \texttt{send}\ \hat{X}, \hat{Y}, y]_{\tilde{X}}$$
$$\mathbf{Resp} \equiv [\ \texttt{receive}\ z;\ \ \texttt{match}\ z/\langle \hat{X}, \hat{Y}, z'\rangle;\ z'' := \texttt{dec}\ z', Y]_{\tilde{Y}}$$

The initiator generates a new nonce and sends it encrypted to the responder. The responder receives the message and recovers the nonce by decrypting the ciphertext. We can prove that if $\tilde{X}$ completes the protocol with $\tilde{Y}$, then x will be a shared secret between them, provided both agents are honest. Formally,

$$\mathsf{Start}(\tilde{X})[\mathbf{Init}]_{\tilde{X}}\mathsf{Honest}(\hat{X}) \wedge \mathsf{Honest}(\hat{Y}) \wedge (\tilde{Z} \neq \tilde{X}) \wedge (\tilde{Z} \neq \tilde{Y}) \Rightarrow \mathsf{Indist}(\tilde{Z}, x)$$

Since the meaning of $\mathsf{Indist}(\tilde{Z}, x)$ (formally defined in Section 7) is that $\tilde{Z}$ cannot distinguish the secret nonce x from a randomly chosen nonce, this formula expresses a standard form of secrecy used in the cryptographic literature.

The axiomatic proof uses **AN2**, a variant of **S1**, and modus ponens **MP**. The proof idea is that at the point the initiator produces the nonce x, by **AN2**, it is indistinguishable from random to everyone else other than $\tilde{X}$ and $\tilde{Y}$. It continues to remain indistinguishable since it appears on the network under encryption with a public key whose corresponding private key is not available to the attacker. This part of the reasoning is codified by an axiom that is similar to **S1** and relies on the fact that the encryption scheme used is IND-CCA2 secure. Modus ponens is used in the general first-order reasoning involved in the proof.

6 Protocol Execution

Given a protocol, adversary, and value of the security parameter, we define a set of protocol traces, each associated with the random bits that produce this sequence of actions and additional randomness for algorithms used in the semantics of formulas about the run. The definition proceeds in two phases. In the initialization phase, we assign a set of roles to each principal, identify a subset which is honest, and provide all entities with private-public key pairs and random bits. In the execution phase, the adversary executes the protocol by interacting with honest principals, as in the accepted cryptographic model of [5].

Initialization: We fix the protocol Q, adversary A, security parameter η, and some randomness R of size polynomially bounded in η. Each principal and each thread (*i.e.*, an instance of a protocol role executed by the principal) is assigned a unique bitstring identifier. We choose a sufficiently large polynomial number of bitstrings $i \in I \subseteq \{0,1\}^\eta$ to represent the names of principals and threads. Randomness R is split into r_i for each honest $i \in I$ (referred to as "coin tosses of honest party i") and R_A (referred to as "adversarial randomness").

The adversary designates some of the principals as *honest* and the rest of the principals as *dishonest*. Intuitively, honest principles will follow one or more roles of the protocol faithfully. The adversary chooses a set of threads, and to each thread it assigns a strand (a program to be executed by that thread), under the restriction that all threads of honest principals are assigned roles of protocol Q.

The key generation algorithm $\mathcal{K}$ of a public-key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is run on $\mathbf{1}^\eta$ for each participant a using randomness r_a, and producing a public-private key pair (pk_a, sk_a). The public key pk_a is given to all participants and to the adversary A; the private key is given to all threads belonging to this principal and to the adversary if the principal is dishonest.

Generating Computational Traces: Following [5], we view an agent i trying to communicate with agent j in protocol session s as a (stateful) oracle $\Pi^s_{i,j}$. The state of each oracle is defined by a mapping λ from atomic symbols to bitstrings (with variables and nonces renamed to be unique for each role) and a counter c. Each oracle proceeds to execute a step of the protocol as defined by actions in the corresponding role's action list, when activated by the adversary.

We omit the details of communication between the adversary and the oracles, and focus on computational interpretation of symbolic protocol actions. Let a_c be the current action in the *ActionList* defining some role of participant i in session s, *i.e.*, $Thread = (i', s')$ where $i = \lambda(i'), s = \lambda(s')$.

If $a_c = (\texttt{new}\ (i', s'), v)$, then update λ so that $\lambda(v) = NonceGen(R_i)$, where $NonceGen$ is a nonce generation function(*e.g.*, $NonceGen$ simply extracts a fresh piece of R_i). If $a_c = (v := \texttt{enc}\ (i', s'), j, u)$, then update λ so that $\lambda(v) = \mathcal{E}(\lambda(u), pk_j, R_i)$ where $\mathcal{E}(\lambda(u), pk_j, R_i)$ is the result of executing the public-key encryption algorithm on plaintext $\lambda(u)$ with public key pk_j and fresh randomness extracted from R_i. For brevity, we omit computational interpretation of decryption and matching (pairing, unpairing, and equality-test) actions. Sending a variable $\texttt{send}\ (i', s'), v$ is executed by sending $\lambda(v)$ to the adversary, and receiving $\texttt{receive}\ (i', s'), v$ is executed by updating λ so that $\lambda(v) = m$ where m is the bitstring sent by the adversary.

At any time during the protocol execution, the adversary A may record any internal, private message on a special *knowledge tape*. This tape is not read by any participant of the protocol. However, its content will be made available to the test algorithms used to decide if a given security formula containing $\mathsf{Indist}(...)$ is valid or not. Let K be $[(i_1, m_1), .., (i_n, m_n)]$ the list of messages m_k written by A on the knowledge tape, indexed by the number of actions i_k already executed when m_k was written (position in the protocol execution). This index will be useful to remember a previous state of the knowledge tape.

At the end of the protocol execution, the adversary A outputs a pair of integers (p_1, p_2) on an *output tape*. When the security formula is a modal formula $\theta[P]_X\varphi$, these two integers represent two positions in the protocol execution where the adversary claims that the formula is violated, i.e. that θ is true in p_1 but φ is false in p_2, with P between p_1 and p_2. Let O be this pair (p_1, p_2) of integers written on the output tape.

The symbolic trace of the protocol is the execution strand $e \in ExecStrand$ which lists, in the order of execution, all honest participant actions and the dishonest participant's `send` and `receive` actions. This strand contains two parts: $InitialState(\mathcal{I})$ stores the initialization data, and the rest is an ordered list of all exchanged messages and honest participants' internal actions.

Definition 1. *(Computational Traces) Given a protocol Q, an adversary A, a security parameter η, and a sequence of random bits $R \in \{0,1\}^{p(\eta)}$ used by the honest principals and the adversary, a* run *of the protocol is the tuple $\langle e, \lambda, O, K, R\rangle$ where e is the symbolic execution strand, $\lambda : Term(e) \to \{0,1\}^{p(\eta)}$ maps the symbolic terms in e to bitstrings, O is the pair of integers written on*

the output tape, and K is the indexed list of messages written on the knowledge tape. Finally, $p(x)$ is a polynomial in x.

A computational trace *is a run with two additional elements: $R_T \in \{0,1\}^{p(\eta)}$, a sequence of random bits used for testing indistinguishability, and $\sigma : FVar(\varphi) \rightarrow \{0,1\}^{p(\eta)}$, a substitution that maps free variables in a formula to bitstrings. The set of computational traces is*

$$T_Q(A,\eta) = \{\langle e, \lambda, O, K, R, R_T, \sigma\rangle \mid R, R_T \text{ chosen uniformly}\}.$$

Definition 2. *(Participant's View) Given a protocol Q, an adversary A, a security parameter η, a participant $\tilde{X}$ and a trace $t = \langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T_Q(A,\eta)$, $View_t(\tilde{X})$ represents $\tilde{X}$'s view of the trace. It is defined precisely as follows:*

If $\hat{X}$ is honest, then $View_t(\tilde{X})$ is the initial knowledge of $\tilde{X}$, a representation of $e_{|\tilde{X}}$ and $\lambda(x)$ for any variable x in $e_{|\tilde{X}}$. If $\hat{X}$ is dishonest, then $View_t(\tilde{X})$ is the union of the knowledge of all dishonest participants $\tilde{X}'$ after the trace t (where $View_t(\tilde{X}')$ is defined as above for honest participants) plus K, the messages written on the knowledge tape by the adversary.

The following three definitions are used in the semantics of the predicate $\mathsf{Indist}()$. Informally, based on some trace knowledge K, the distinguisher D tries to determine which of two bitstrings is the value of a symbolic term. One of the bitstrings will be the computational value of the term in the current run, while the other will be a random bitstring of the same structure, chosen in a specific way. The order of the two bitstrings presented to the distinguisher is determined by an LR Oracle using a random selector bit.

Definition 3. *(LR Oracle) The LR Oracle [4] is used to determine the order in which two bitstrings are presented depending on the value of the selector bit, i.e. $LR(s_0, s_1, b) = \langle s_b, s_{1-b}\rangle$.*

Definition 4. *(Distinguishing test input) Let u be a symbolic term and σ be a substitution that maps variables of u to bitstrings. We construct another bitstring $f(u, \sigma, r)$, whose symbolic representation is the same as u. Here, r is a sequence of bits chosen uniformly at random. The function f is defined by induction over the structure of the term u.*

- *Nonce u : $f(u, \sigma, r) = r$*
- *Name/Key u : $f(u, \sigma, r) = \sigma(u)$*
- *Pair $u = \langle u_1, u_2\rangle$: $f(\langle u_1, u_2\rangle, \sigma, r_1; r_2) = \langle f(u_1, \sigma, r_1), f(u_2, \sigma, r_2)\rangle$*
- *Encryption $u = \{v\}^n_K$: $f(\{v\}^n_K, \sigma, r_1; r_2) = \mathcal{E}(f(v, \sigma, r_1), \sigma(K), r_2)$*

Definition 5. *(Distinguisher) A distinguisher D is a polynomial time algorithm which takes as input a tuple $\langle K, t, \langle s_0, s_1\rangle, R, \eta\rangle$, consisting of knowledge K, symbolic term t, two bitstrings s_0 and s_1, randomness R and the security parameter η, and outputs a bit b'.*

The next definition is used while defining semantics of modal formulas. Given a set T of traces and a strand P of actions executed by a thread $\tilde{X}$, the set T_P includes only those traces from T which contain P. $Pre(T_P)$ is obtained from T_P by taking the initial segment of each trace upto the point where P starts. The precondition of a modal formula is evaluated over this set. $Post(T_P)$ is similarly defined; the only difference is now the trace is cut at the point that P ends. The postcondition of a modal formula is evaluated over this set. The begin and end positions are determined by the component O in the trace.

Definition 6. *(Splitting computational traces) Let T be a set of computational traces and $t = \langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$. $O = \langle p_1, p_2 \rangle$, $e = InitialState(\mathcal{I}); s$, and $s = s_1; s_2; s_3$ with p_1, p_2 the start and end positions of s_2 in s. Given a strand P executed by participant $\tilde{X}$, we denote by T_P the set of traces in T for which there exists a substitution σ' which extends σ to variables in P such that $\sigma'(P) = \lambda(s_{2\,|\tilde{X}})$. The complement of this set is denoted by $T_{\neg P}$ and contains all traces which do not have any occurrence of the strand P. We define the set of traces $Pre(T_P) = \{t[s \leftarrow s_1, K \leftarrow K_{\leq p_1}, \sigma \leftarrow \sigma'] \mid t \in T_P\}$, where $K_{\leq p}$ is the restriction of the knowledge tape K to messages written before the position p. We define the set of traces $Post(T_P) = \{t[s \leftarrow s_1; s_2, K \leftarrow K_{\leq p_2}, \sigma \leftarrow \sigma'] \mid t \in T_P\}$.*

7 Computational Semantics

The semantics of a formula φ on a set T of computational traces is a subset $T' \subseteq T$ that respects φ in some specific way. For many predicates and connectives, the semantics is essentially straightforward. For example, an action predicate such as Send selects a set of traces in which a send occurs. However, the semantics of predicates Indist and Possess is inherently more complex.

Intuitively, an agent possesses the value of an expression (such as another agent's nonce or key) if the agent can compute this value from information it has seen, with high probability. If an agent is honest, and therefore follows the rules of the protocol, then it suffices to use a simple, symbolic algorithm for computing values from information seen in the run of a protocol. For dishonest agents, we would prefer in principle to allow any probabilistic polynomial-time algorithm. However, quantifying over such algorithms, in a way that respects the difference between positive and negative occurrences of the predicate in a formula, appears to introduce some technical complications. Therefore, in the interest of outlining a relatively simple form of computational semantics, we will use a fixed algorithm. This gives a useful semantics for formulas where $\mathsf{Possess}(\tilde{X}, u)$ is used under the hypothesis that $\hat{X}$ is honest. We leave adequate treatment of the general case for future work.

Intuitively, an agent has partial information about the value of some expression if the agent can distinguish that value, when presented, from a random value generated according to the same distribution. More specifically, an agent has partial information about a nonce u if, when presented with two bitstrings of the appropriate length, one the value of u and the other chosen randomly, the agent

has a good chance of telling which is which. As with Possess, there are technical issues associated with positive and negative occurrences of the predicate. For positive occurrences of Indist, we should say that *no* probabilistic polynomial-time algorithm has more than a negligible chance, where as for $\neg$Indist(...) we want to say that *there exists* a probabilistic polynomial-time distinguisher. In order to present a reasonably understandable semantics, and establish a useful basis for further exploration of computational semantics of symbolic security logics, we give an interpretation that appears accurate for formulas that have only positive occurrences of Indist and could be somewhat anomalous for formulas that contain negative occurrences. This seems adequate for reasoning about many secrecy properties, since these are expressed by saying that at the end of any run of the protocol, a value used in the run is indistinguishable from random.

Conditional implication $\theta \Rightarrow \varphi$ is interpreted using the negation of θ and the conditional probability of φ given θ. This non-classical interpretation of implication seems to be essential for relating provable formulas to cryptographic-style reductions involving conditional probabilities. In particular, the soundness proof for the "source" axiom **S1**, not proved in this conference paper, uses the conditional aspect of this implication in a fundamental way. On the other hand, $\Rightarrow$ coincides with $\supset$ in formulas where Indist does not appear on the right hand size of the implication.

We inductively define the semantics $[\![\varphi]\!](T, D, \epsilon)$ of a formula φ on the set T of traces, with distinguisher D and tolerance ϵ. The distinguisher and tolerance are not used in any of the clauses except for Indist, where they are used to determine whether the distinguisher has more than a negligible chance of distinguishing the given value from a random value. In definition 7 below, the tolerance is set to a negligible function of the security parameter and $T = T_Q(A, \eta)$ is the set of traces of a protocol Q with adversary A.

- $\left[\!\left[\mathsf{Send}(\tilde{X}, u) \right]\!\right](T, D, \epsilon)$ is the collection of all $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ such that some action in the symbolic execution strand e has the form $\mathtt{send}\ \tilde{Y}, v$ with $\lambda(\tilde{Y}) = \sigma(\tilde{X})$ and $\lambda(v) = \sigma(u)$. Recall that σ maps formula variables to bitstrings and represents the environment in which the formula is evaluated.
- $[\![\mathsf{a}(\cdot, \cdot)]\!](T, D, \epsilon)$ for other action predicates a is similar to $\mathsf{Send}(\tilde{X}, u)$.
- $\left[\!\left[\mathsf{Honest}(\hat{X}) \right]\!\right](T, D, \epsilon)$ is the collection of all $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ where $e = InitialState(\mathcal{I}); s$ and $\sigma(X)$ is designated *honest* in the initial configuration $\mathcal{I}$. Since we are only dealing with static corruptions in this paper, the resulting set is either the whole set T or the empty set ϕ depending on whether a principal is honest or not.
- $\left[\!\left[\mathsf{Start}(\tilde{X}) \right]\!\right](T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ where $e = InitialState(\mathcal{I}); s$ and $\lambda(s)_{|\sigma(\tilde{X})} = \epsilon$. Intuitively, this set contains traces in which $\tilde{X}$ has executed no actions.
- $[\![\mathsf{Contains}(u, v)]\!](T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma \rangle \in T$ such that there exists a series of decryptions with $\{\lambda(k) \mid k \in Key\}$ and projections (π_1,π_2) constructing $\sigma(v)$ from $\sigma(u)$. This definition guarantees that the result is the whole set T if v is a symbolic subterm of u.

- $[\![\mathsf{ContainsOut}(u,\, v,\, t)]\!]\,(T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T$ such that there exists a series of projections (π_1,π_2) and decryptions with $\{\lambda(k) \mid k \in Key\}$, where $\sigma(t)$ is never decomposed, creating $\sigma(v)$ from $\sigma(u)$. This definition ensures that the result is the whole set T if v is a symbolic subterm of u but is not a subterm of t.
- $[\![\theta \wedge \varphi]\!]\,(T, D, \epsilon) = [\![\theta]\!]\,(T, D, \epsilon) \cap [\![\varphi]\!]\,(T, D, \epsilon)$.
- $[\![\theta \vee \varphi]\!]\,(T, D, \epsilon) = [\![\theta]\!]\,(T, D, \epsilon) \cup [\![\varphi]\!]\,(T, D, \epsilon)$.
- $[\![\neg\varphi]\!]\,(T, D, \epsilon) = T \setminus [\![\varphi]\!]\,(T, D, \epsilon)$.
- $[\![\exists x.\, \varphi]\!]\,(T, D, \epsilon) = \bigcup_\beta([\![\varphi]\!]\,(T[x \leftarrow \beta], D, \epsilon)[x \leftarrow \sigma(x)])$ with $T[x \leftarrow \beta] = \{t[\sigma[x \leftarrow \beta]] \mid t = \langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T\}$, and β any bitstring of polynomial size.
- $[\![\theta \Rightarrow \varphi]\!]\,(T, D, \epsilon) = [\![\neg\theta]\!]\,(T, D, \epsilon) \cup [\![\varphi]\!]\,(T', D, \epsilon)$, where $T' = [\![\theta]\!]\,(T, D, \epsilon)$. Note that the semantics of φ is taken over the set T' given by the semantics of θ, as discussed earlier in this section.
- $[\![u = v]\!]\,(T, D, \epsilon)$ includes all traces $\langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T$ such that $\sigma(u) = \sigma(v)$.
- $\left|\!\left[\mathsf{DecryptsHonest}(\tilde{Y},\, \{u\}^r_X) \right]\!\right|\,(T, D, \epsilon) = [\![\varphi]\!]\,(T, D, \epsilon)$ with $\varphi = \mathsf{Honest}(\hat{X}) \wedge \exists v.\, v := \mathsf{dec}\ \tilde{Y},\, \{u\}^r_X$.
- $\left|\!\left[\mathsf{Source}(\tilde{Y},\, u,\, \{m\}^r_X) \right]\!\right|\,(T, D, \epsilon) = [\![\exists v. \forall w.\, \varphi]\!]\,(T, D, \epsilon)$ with :

$$\begin{aligned}\varphi \;&= \mathsf{New}(\tilde{Y}, u) \wedge \mathsf{Contains}(m,\, u)\\ &\quad \wedge \mathsf{Contains}(v,\, \{m\}^r_X) \wedge \mathsf{Send}(\tilde{Y}, v)\\ &\quad \wedge \neg\mathsf{ContainsOut}(v,\, u,\, \{m\}^r_X)\\ &\quad \wedge (v \neq w \wedge \mathsf{Contains}(w,\, u)) \Rightarrow \neg\mathsf{Send}(\tilde{Y}, w)\end{aligned}$$

- $\left|\!\left[\mathsf{Possess}(\tilde{X},\, u) \right]\!\right|\,(T, D, \epsilon)$ includes all traces $t = \langle e, \lambda, O, K, R, R_T, \sigma\rangle \in T$ such that $\sigma(u)$ can be built from $View_t(\sigma(\tilde{X}))$ with the Dolev-Yao deduction rules.
- $\left|\!\left[\mathsf{Indist}(\tilde{X},\, u) \right]\!\right|\,(T, \epsilon, D) = T$ if

$$\frac{|\{D(View_t(\sigma(\tilde{X})), u, LR(\sigma(u), f(u, \sigma, r), b), R_D, \eta) = b \mid t \in T\}|}{|T|} \leq \frac{1}{2} + \epsilon$$

  and the empty set ϕ otherwise. Here, the random sequence $b; r; R_D = R_T$, the testing randomness for the trace t.
- $[\![\theta[P]_{\tilde{X}}\varphi]\!]\,(T, D, \epsilon) = T_{\neg P} \cup [\![\neg\theta]\!]\,(Pre(T_P), D, \epsilon) \cup [\![\varphi]\!]\,(Post(T_P), D, \epsilon)$ with $T_{\neg P}$, $Pre(T_P)$, and $Post(T_P)$ as given by Definition 6.

Definition 7. *A protocol Q satisfies a formula φ, written $Q \models \varphi$, if $\forall A$ providing an active protocol adversary, $\forall D$ providing a probabilistic-polynomial-time distinguisher, $\forall \nu$ giving a negligible function, $\exists N$, $\forall \eta \geq N$,*

$$|\,[\![\varphi]\!]\,(T, D, \nu(\eta))\,| \;/\; |\,T\,| \;\geq\; 1 - \nu(\eta)$$

where $[\![\varphi]\!](T, D, \nu(\eta))$ *is the subset of* T *given by the semantics of* φ *and* $T = T_Q(A, \eta)$ *is the set of computational traces of protocol* Q *generated using adversary* A *and security parameter* η*, according to Definition 1.*

Theorem 1. *(Soundness)* $\forall Q, \forall \varphi, \quad Q \vdash \varphi \Rightarrow Q \models \varphi$

8 Conclusion and Future Work

We propose a computational semantics for a variant of the Protocol Composition Logic presented in [2, 7, 8, 10]. The associated soundness theorem implies that it is possible to reason symbolically, and at a high level, about probabilistic polynomial-time security properties. Although omitted from this conference paper, the soundness proof uses a combination of information-theoretic arguments, calculations about negligible functions, and cryptographic-style reductions involving encryption. While the semantics given here has some imperfections, such as an interpretation of indistinguishability that only seems appropriate for formulas where Indist appears with positive polarity, the general approach seems promising. We look forward to future efforts to lift certain restrictions on the logic, explore the semantics and axiomatization of logical connectives over probabilistic polynomial-time interpretations, and extend the approach suggested here to additional cryptographic primitives, such as signatures and hash functions. One interesting research direction might be to develop a version of this semantics based on information-theoretic security, since that may provide some useful insight into problems we encountered in developing the semantics.

Acknowledgments: Thanks to Bogdan Warinschi, Andre Scedrov, and Dan Boneh for many insightful comments and suggestions.

References

1. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
2. M. Backes, A. Datta, A. Derek, J. C. Mitchell, and M. Turuani. Compositional analysis of contract signing protocols. In *Proceedings of 18th IEEE Computer Security Foundations Workshop*. IEEE, 2005. To appear.
3. M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015, 2003.
4. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Advances in Cryptology - EUROCRYPT 2000, Proceedings*, pages 259–274, 2000.
5. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '93)*, pages 232–249. Springer-Verlag, 1994.
6. V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proceedings of 14th European Symposium on Programming (ESOP'05)*, Lecture Notes in Computer Science, pages 157–171. Springer-Verlag, 2005.

7. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system for security protocols and its logical formalization. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, pages 109–125. IEEE, 2003.
8. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 2005. To appear.
9. D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29):198–208, 1983.
10. N. Durgin, J. C. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11:677–721, 2003.
11. J. Herzog. The Diffie-Hellman key-agreement scheme in the strand-space model. In *Proceedings of 16th IEEE Computer Security Foundations Workshop*, pages 234–247, 2003.
12. J. Herzog. *Computational Soundness for Standard Assumptions of Formal Cryptography*. PhD thesis, MIT, 2004.
13. R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS '03)*, pages 372–383. IEEE, 2003.
14. R. Janvier, L. Mazare, and Y. Lakhnech. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *Proceedings of 14th European Symposium on Programming (ESOP'05)*, Lecture Notes in Computer Science, pages 172–185. Springer-Verlag, 2005.
15. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Theory of Cryptography Conference - Proceedings of TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer-Verlag, 2004.
16. N. J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–87, 1986.
17. V. Shoup. On formal models for secure key exchange (version 4). Technical Report RZ 3120, IBM Research, 1999.
18. B. Warinschi. A computational analysis of the Needham-Schroeder(-Lowe) protocol. In *Proceedings of 16th Computer Science Foundation Workshop*, pages 248–262. ACM Press, 2003.