Composition of Cryptographic Protocols in a Probabilistic Polynomial-Time Process Calculus

P. Mateus[1]*, J. Mitchell[2]**, and A. Scedrov[3]***

[1] Center for Logic and Computation, IST, Lisbon, Portugal
[2] Department of Computer Science, Stanford University, Stanford, USA
[3] Department of Mathematics, University of Pennsylvania, Philadelphia, USA

Abstract. We describe a probabilistic polynomial-time process calculus for analyzing cryptographic protocols and use it to derive compositionality properties of protocols in the presence of computationally bounded adversaries. We illustrate these concepts on oblivious transfer, an example from cryptography. We also compare our approach with a framework based on interactive Turing machines.
Keywords: cryptographic protocols, probabilistic process calculus, computational security, composition theorem.

1 Introduction

The design and verification of security protocols is a difficult problem. Some of the difficulties come from subtleties of cryptographic primitives. Further difficulties arise because security protocols are required to work properly when multiple instances of the protocol are carried out in parallel, where a malicious intruder may combine data from separate sessions in order to confuse honest participants. Moreover, although the protocols themselves are often very simple, the security properties they are supposed to achieve are rather subtle and should be formulated with great care.

A variety of methods are used for analyzing and reasoning about security protocols. Although such methods differ in significant ways, many of them reflect the same basic assumptions about the way an adversary may interact with the protocol or attempt to decrypt encrypted messages . In the common model, largely derived from [10] and suggestions found in [24], a protocol adversary is allowed to choose among possible actions nondeterministically. This

* Partially supported by FCT grant SFRH/BPD/5625/2001, and by FEDER/FCT project Fiblog POCTI/2001/MAT/37239.

** Partially supported by OSD/ONR MURI "Semantic Consistency in Information Exchange" as ONR Grant N00014-97-1-0505, and by OSD/ONR CIP/SW URI "Software Quality and Infrastructure Protection for Diffuse Computing" as ONR Grant N00014-01-1-0795.

*** Partially supported by OSD/ONR MURI "Semantic Consistency in Information Exchange" as ONR Grant N00014-97-1-0505, by OSD/ONR CIP/SW URI "Software Quality, Infrastructure Protection for Diffuse Computing" as ONR Grant N00014-01-1-0795, and by NSF Grant CCR-0098096.

is a convenient idealization, intended to give the adversary a chance to find an attack if there is one. In the presence of nondeterminism, however, the set of messages an adversary may use to interfere with a protocol must be restricted severely. Although the idealized assumptions make protocol analysis tractable, they also make it possible to "verify" protocols that are in fact susceptible to simple attacks that lie outside the adversary model. Another limitation is that a deterministic or nondeterministic setting does not allow us to analyze probabilistic protocols. In other words, actual protocols use actual cryptosystems that may have their own weaknesses, or might employ probabilistic techniques not expressed in the idealized model.

Recently there have been several research efforts to relate the idealized model to cryptographic techniques and the computational model based on probabilistic polynomial-time computation, including [7, 16, 23, 25, 26, 3, 2, 9, 4]. While these efforts develop rigorous mathematical settings carried out so far only "by hand", it is hoped that they will eventually lead to a new generation of "high fidelity" automated tools for security analysis that will be able to express the methods and concepts of modern cryptography.

Our initial contribution to this line of research was a formulation of a process calculus proposed in [16, 23] as the basis for a form of protocol analysis that is formal, yet closer in foundations to the mathematical setting of modern cryptography. The framework relies on a language for defining communicating polynomial-time processes. The reason we restrict processes to probabilistic polynomial time is so that we can reason about the security of protocols by quantifying over all "adversarial" processes definable in the language. In effect, establishing a bound on the running time of an adversary allows us to relax the simplifying assumptions. Specifically, it is possible to consider adversaries that might send randomly chosen messages, or perform sophisticated (yet probabilistic polynomial-time) computation to derive an attack from messages it overhears on the network. An important aspect of our framework is that we can analyze probabilistic as well as deterministic encryption functions and protocols. Without a probabilistic framework, it would not be possible to analyze an encryption function such as [11], for which a single plaintext may have more than one ciphertext.

Some of the basic ideas of our prior work are outlined in [16, 23]. Further example protocols are considered in [17]. The closest technical precursor is [1], which uses observational equivalence and channel abstraction but does not involve probability or computational complexity bounds. Prior work on CSP and security protocols, *e.g.*, [28, 29], also uses process calculus and security specifications in the form of equivalence or related approximation orderings on processes.

This approach is based on the intuition that security properties of a protocol P may be expressed by means of existence of an idealized protocol Q such that for any adversary M, the interactions between M and P have the same observable behavior as the interactions between M and Q. The idea of expressing security properties in terms of some comparison to an ideal protocol goes back at least to [15, 6, 5, 20]. Here we emphasize a formalization of this idea by using observational equivalence, a standard notion from programming language theory. That is, two protocols P and Q are observationally equivalent if any program $C[P]$ has the same observable behavior as the program $C[Q]$, with Q instead of P. The reason observational equivalence is applicable to security analysis is that it involves quantifying over all possible additional processes represented by the contexts $C[\ ]$ that might interact with P and Q, in precisely the same way that security properties involve quantifying over all possible adversaries. Our framework is a refinement of this approach. In our asymptotic formulation [16, 23], observational equivalence between probabilistic polynomial-time processes coincides with the traditional notion of indistinguishability by polynomial-time statistical tests [13, 30], a standard way of characterizing cryptographic primitives.

In this paper we derive a compositionality property from inherent structural properties of our process calculus. Basically, compositionality states that composing secure protocols remains secure. We obtain a general result of this kind in two steps. We consider a notion of a secure realization, or, *emulation* of an ideal protocol, motivated by [7] but here expressed by means of asymptotic observational equivalence. We show that the notion of emulation is congruent with the primitives of the calculus. Compositionality follows because the security requirements are expressed in the form that a real protocol securely realizes an ideal protocol.

We also illustrate some of these concepts on a traditional cryptographic example of oblivious transfer [27, 12, 14, 9]. We show how the natural security requirements may be expressed in our calculus in the form that a real protocol emulates an ideal protocol. Finally, we establish an important relationship between the process calculus framework and the interactive Turing machine framework discussed in [7, 9, 25, 26, 3]. Indeed, the work based on [7] provides an encyclopedic treatment of a number of security requirements in a compositional setting. However, the framework of interactive Turing machines, even if optimal to deal with complexity results, is rather low-level and does not seem naturally suited for specification of and reasoning about cryptographic protocols. Moreover, the framework of interactive Turing machines comes about from the connections between cryptography and complexity, and therefore, some effort must be spent to obtain structural results, such as the composition theorem.

Basic definition and properties of the process calculus are discussed in Section 2. In Section 3 we discuss the notion of emulation, prove a general composition theorem, and analyze the example of oblivious transfer. A comparison to the interactive Turing machine model is given in Section 4. We conclude the paper in Section 5.

2 Probabilistic polynomial-time process calculus

In this section, we describe a version of the probabilistic polynomial-time process calculus [16, 23], with the intention of using of the calculus to derive compositionality properties of secure protocols.

2.1 Syntax

We assume as given once and for all a countable set C of *channels*. In a discussion of security protocols it is common to consider a security parameter $n \in \mathbb{N}$. From now on, the symbol n is reserved to designate such security parameter. The role of this parameter is twofold. It bounds the length of expressions that can be sent through each channel by a polynomial in $|n|$, the length of n. This is written into the syntax: we introduce a *bandwidth map* $w : C \to \mathbf{q}$, where $\mathbf{q}$ is the set of all polynomials in one variable taking positive values on natural numbers. Given a value n for the security parameter, a channel c can send messages with at most $w(c)(|n|)$ bits. It turns out that the security parameter also bounds all the computation in the calculus by probabilistic polynomial time. This property the calculus is proved in [23].

The protocol language consists of a set of *terms,* or functional expressions that do not perform any communication, and *processes,* which can communicate with each other.

We assume a set of numerical terms T (endowed with a set of variables Var) with the following two properties. For any probabilistic polynomial (in the length of the security parameter n) time function f there is a term $t \in T$ and the associated probabilistic Turing machine M_t that computes f. Furthermore, given any term $t \in T$, the associated probabilistic Turing machine M_t is always probabilistic polynomial time, PPT, with input n and the numerical values of variables of t. (An example of such a set of terms T is described in [22], but the details of the construction are not needed here.) In order to ease notation, we shall confuse a term $t(\boldsymbol{x})$ with the a probabilistic polynomial time function $f(\boldsymbol{x}, n)$ associated to the PPT machine M_t. Hence, one can envision T simply as the set of all probabilistic polynomial time functions, neglecting any further syntax. Once again, mind that all terms denote probabilistic polynomial time functions in the length of the security parameter n. After fixing n, we denote by $P(t(\boldsymbol{a}) \to a)$ the probability of $M_t(\boldsymbol{a}, n)$ converging to a and $P(t(\boldsymbol{a}) = t'(\boldsymbol{b}))$ the probability of both associated Turing machines converging to the same value.

We now present our language of process expressions, a version of Milner's Calculus of Communicating Systems, CCS [21]. Bear in mind, though, that for us the overall computation must be probabilistic polynomial time and hence we use only polynomially bounded replication.

Definition 1. The *language of process expressions* $\mathcal{L}$ is obtained inductively as follows:

1. $0 \in \mathcal{L}$ (empty process: does nothing);
2. $\nu_c.Q \in \mathcal{L}$ where $c \in C$ and $Q \in \mathcal{L}$ (private channel: do Q with channel c considered private);
3. $\langle t \rangle_c \in \mathcal{L}$ where $t \in T$ and $c \in C$ (output: transmit the value of t on channel c);
4. $(x)_c.Q \in \mathcal{L}$ where $c \in C$, $x \in Var$ and $Q \in \mathcal{L}$ (input: read value for x on channel c and do Q);
5. $[t_1 = t_2].Q \in \mathcal{L}$ where $t_1, t_2 \in T$ and $Q \in \mathcal{L}$ (match: if $t_1 = t_2$ then do Q);

6. $(Q_1|Q_2) \in \mathcal{L}$ where $Q_1, Q_2 \in \mathcal{L}$ (parallel composition: do Q_1 in parallel with Q_2);
7. $!_q Q \in \mathcal{L}$ where $Q \in \mathcal{L}$ and $q \in \mathbf{q}$ (polynomially bounded replication: execute $q(|n|)$ copies of Q in parallel).

Every input or output on a private channel must appear within the scope of a ν-operator binding that channel, that is, the channel name in the scope of a ν-operator is considered bound. A *process* is a process expression in which the security parameter is replaced with a fixed natural number. Observe that the length of any process expression in $\mathcal{L}$ is polynomial in $|n|$.

For each fixed value k of the security parameter, we can remove replication by replacing each subexpression $!_q R$ of an expression Q by $q(|k|)$ copies of R in parallel, denoted $\overline{Q}_k$.

Let us also fix the following terminology and useful conventions. We assume that in any process expression in $\mathcal{L}$ private channels are named apart from other channels, which we call *public.* Analogously to first-order logic, a variable x is said to occur *free* in a process expression $Q \in \mathcal{L}$ if there is an occurrence of x that does not appear in the scope of a binding operator $(x)_c$. The set of all free variables of Q is called the *parameters* of Q and is denoted by $\mathbb{P}_Q$. A process expression without parameters is called *closed.*

Intuitively, messages are essentially pairs consisting of a "channel name" and a data value. The expression $\langle M \rangle_c$ places a pair $\langle c, M \rangle$ onto the network. The expression $(x)_c.P$ matches any pair $\langle c, m \rangle$ and continues process P with x bound to the least significant $w(c)(|n|)$ bits of value m, because of the bandwidth restrictions. When $(x)_c.P$ corresponds to a pair $\langle c, M \rangle$, the pair $\langle c, M \rangle$ is removed from the network and is no longer available to be read by another process. Evaluation of $(x)_c.P$ does not proceed unless or until a pair $\langle c, m \rangle$ is available.

Although we use channel names to indicate the intended source and destination of a communication, any process can potentially read any message, except when a channel is bound by the ν-operator, hiding communication. However, we only intend to use private channels for ideal specifications and for modeling various initial conditions in protocols regarding secret data, but we do not use private channels for modeling actual protocols. This communication model allows an adversary (or any process) to intercept a message between any two participants in a protocol. Once read, a pair is removed so that an adversary has the power to prevent the intended recipient from receiving a message. An adversary (or other process) may overhear a transaction without interfering simply by retransmitting every communication that it intercepts.

Observe that the output primitive of the calculus allows us to compute the image of a probabilistic polynomial time function f into some value a and send a through a channel c (or part of it if the bandwidth of c is too small). Moreover, the matching condition endows the calculus with the possibility of checking whether two terms converge to the same value. As we shall see, by combining these primitives we give a lot of power to the calculus.

In order to illustrate the flexibility of the process calculus we present the following two examples:

Example 1 (RSA encryption and decryption).

Start by considering a very simple process S that knows some message M and integers a, m and just outputs $M^a \mod m$, this dummy process can be presented as $S(M) := \langle M^a \mod m \rangle_u$. Let us develop this just a bit more, and consider a remote procedure that receives x and outputs $x^a \mod m$. This procedure can be modeled by the following process: $RP := (x)_c.S(x)$.

Finally, consider that p, q are primes, and a, b are integers such that $ab \equiv 1 \mod \phi(pq)$. Consider the process $RSA(a, b, pq) := \text{Send}(M) \mid \text{Rec}$, where $\text{Send}(M) := \langle M^a \mod pq \rangle_u$ and $\text{Rec} := \nu_{u'}.(x)_u.\langle x^b \mod pq \rangle_{u'}$.

Here the sender sends a message encrypted with the receiver's encryption key a and the receiver decrypts with its decryption key b and stores the plaintext privately.

Example 2 (Modular sequential composition). Suppose that a process $Q(\boldsymbol{c} \to \boldsymbol{u})$ receives inputs through public channels $\boldsymbol{c}$, works these inputs in some way, and returns relevant outputs through public channels $\boldsymbol{u}$. If another process R needs at some point to use Q, R just needs to feed Q with the required inputs, say $\boldsymbol{i}$ and wait for Q to output through channels $\boldsymbol{u}$. Indeed, process R could be defined, for instance, as $R := \langle \boldsymbol{i} \rangle_{\boldsymbol{c}} | (\boldsymbol{x})_{\boldsymbol{u}}.\ R' | Q(\boldsymbol{c} \to \boldsymbol{u})$.

2.2 Semantics

The semantics of a process Q is a Markov chain $\mathcal{S}(Q)$ over multisets of a special kind of processes, which we call eligible. Intuitively, the states of $\mathcal{S}(Q)$ represent reduction stages of the process and the transitions denote probabilistic choices between reductions. Recall that only the values on public channels are observed, and thus in the semantics these channels have special status.

The initial state S^0 of $\mathcal{S}(Q)$ is the multiset consisting of certain subprocesses of Q running in parallel, that is, if $Q = Q_1 | \ldots | Q_m$ then $S^0 = \{Q_1, \ldots Q_m\}$ where the head operator of each Q_i is not parallel composition. This setting captures the idea that in the initial state all such subprocesses are available for reduction. Actually, one obvious exception to this construction needs to be considered: if $Q = 0$ then there is no process to be reduced, and so, S^0 is the empty multiset. At this stage, we assume that the security parameter n is fixed, and therefore, all iterations have been replaced by parallel compositions.

Taking into account the discussion above, it is clear that we have to distinguish processes with head operator different from parallel composition. We call all these processes *eligible for reduction* and they can be defined formally as follows:

Definition 2. The *set of eligible processes* $\mathcal{E}$ is defined inductively as follows:

- $0 \in \mathcal{E}$;
- $\langle t \rangle_c \in \mathcal{E}$ where $t \in T$ and $c \in C$;
- $(x)_c.Q \in \mathcal{E}$ where $c \in C$, $x \in Var$ and $Q \in \mathcal{L}$;
- $[t_1 = t_2].Q \in \mathcal{E}$ where $t_1, t_2 \in T$ and $Q \in \mathcal{L}$;

In order to present the operational semantics, we set some notation on finite multisets. A finite multiset $\mathcal{M}$ over a set L is a map $\mathcal{M} : L \to \mathbb{N}$ such that $\mathcal{M}^{-1}(\mathbb{N}^+)$ is finite. The difference of $\mathcal{M}$ and $\mathcal{M}'$ is the multiset $\mathcal{M} \setminus \mathcal{M}'$ where $(\mathcal{M} \setminus \mathcal{M}')(l) = \max(0, \mathcal{M}(l) - \mathcal{M}'(l))$. The union of two multisets $\mathcal{M}$ and $\mathcal{M}'$ is the multiset $\mathcal{M} \cup \mathcal{M}'$ where $(\mathcal{M} \cup \mathcal{M}')(l) = \mathcal{M}(l) + \mathcal{M}'(l)$. We say that $\mathcal{M} \subset \mathcal{M}'$ iff $\mathcal{M}(l) \leq \mathcal{M}'(l)$ for all $l \in L$. Furthermore, we say that $l \in \mathcal{M}$ iff $\{l\} \subset \mathcal{M}$. Finally, we call $\wp_{fm}(L)$ the set of all finite multisets over L.

As discussed above, given a process $Q = Q_1 | \dots | Q_m$ we need to construct the initial state $S^0 = \{Q_1, \dots Q_m\}$ where Q_i is an eligible process. This construction is also useful during reduction, since after reducing some processes more parallel compositions may appear.

To deal with the binding channel operator ν we consider a set of fresh channels F and a fresh function

$$\text{fresh} : C \to F$$

that maps a channel c to a fresh channel c' (that is, that does not occur anywhere else) such that $w(c) = w(c')$. This fresh function insures that one can find a channel c' not occurring in any other process and therefore c' can be considered private to some process at hand. As expected, the communication through these channels is not observed.

Once again, at this step we assume a fixed security parameter n and that all iterations have been replaced by parallel compositions.

Definition 3. Given a process $Q \in \mathcal{L}$ without iteration we obtain the *multiset of sequences* of Q, which we denote by $\mathcal{M}_Q$, as follows:

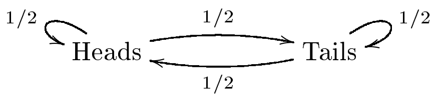
- $\mathcal{M}_Q = \{\}$ whenever $Q = 0$;
- $\mathcal{M}_Q = \{Q\}$ whenever Q is eligible and different from 0;
- $\mathcal{M}_Q = \mathcal{M}_{R^c_{\text{fresh}(c)}}$ whenever $Q = \nu_c.R$ and where $R^c_{\text{fresh}(c)}$ is the process where all free occurrences of c where replaced by $\text{fresh}(c)$;
- $\mathcal{M}_Q = \mathcal{M}_{Q'} \cup \mathcal{M}_{Q''}$ whenever $Q = Q'|Q''$.

Instead of presenting the semantics with probabilistic labeled transition systems as in [16, 23], here we will use an alternative: Markov chains, a well-established concept from the stochastic processes community, following the style in [19].

Recall that a Markov chain A over a set S can be modeled as a state machine with state space S where transiting from state s to state s' has probability $0 \leq A(s, s') \leq 1$. Obviously, these probabilities are such that for any $s \in S$

$$\sum_{s' \in S} A(s, s') = 1.$$

Example 3. The following simple Markov chain models the stochastic process of independently tossing a fair coin ad nauseam:



Markov chains are specially suited to model the semantics of the process algebra, since, like in [16, 23], process reduction is probabilistic and depends only of the (multi)set of subprocesses that remain to be reduced. Thus, the semantics of a process Q is a Markov chain over the multisets of eligible process subprocesses of Q.

In order to establish the semantics for all processes, one can consider a huge Markov chain S, that given any multiset of eligible processes decides, accordingly to some probabilistic rules, which terms should be reduced. Such Markov chain is usually called a *scheduler*. Hence, given a multiset $\mathcal{M}$ of eligible processes there is a probability $S(\mathcal{M}, \mathcal{M}')$ of reducing $\mathcal{M}$ to $\mathcal{M}'$. The semantics of a single process Q is recovered by restricting the scheduler S to the states reachable from the multiset $\mathcal{M}_Q$ of sequences of Q.

Note that one can not accept any Markov chain as a scheduler. For instance, if the scheduler is at state $\{\}$, and therefore there is no process to reduce, the scheduler can not transit to, say, $\{Q\}$ with positive probability. In other words, $S(\{\}, \{Q\})$ must be zero. Hence, if the scheduler transits from one state to another with positive probability then at least one reduction must be enabled. For this reason, it is relevant to enumerate all possible reductions:

1. Term reduction: a term not in the scope of any input is reduced.
2. Match: a match between terms occurs.
3. Mismatch: a mismatch between terms occurs.
4. Communication: two processes communicate via an input and output.
5. Termination: none of the previous reductions is enabled (and so reduction has terminated).

Communication has lower priority than term reduction, match and mismatch. Hence, communication is only enabled when none of the previous mentioned reductions are enabled. As stated before, each reduction impose restrictions on the scheduler. For instance, *Termination* imposes that if there is no reduction enabled at state $\mathcal{M}$ then $S(\mathcal{M}, \mathcal{M}) = 1$. All other restrictions are more else obvious and are captured in the following definition.

Definition 4. A *scheduler* S for a security parameter n is a Markov chain with state space $\wp_{fm}(\mathcal{E})$ such that if $S(\mathcal{M}_1, \mathcal{M}_2) > 0$ then one of the following conditions hold:

1. *Term reduction:* $\langle t\rangle_c \in \mathcal{M}_1$ and $\mathcal{M}_2 = (\mathcal{M}_1 \setminus \{\langle t\rangle_c\}) \cup \{\langle m\rangle_c\}$; $\langle t\rangle_c$ not in the scope of any input, t does not have any free variables, t evaluates to m' with positive probability and m corresponds to the least significant $w(c)(|n|)$ bits of m'; the transition probability is given by
$$S(\{\langle t\rangle_c\}, \{\langle m\rangle_c\}) = \sum_{m' : m = m'|_{w(c)(n)}} P(t \to m').$$
2. *Match:* $[t_1 = t_2].Q \in \mathcal{M}_1$ where t_1, t_2 are closed terms and $\mathcal{M}_2 = (\mathcal{M}_1 \setminus \{[t_1 = t_2].Q\}) \cup \mathcal{M}_Q$; the transition probability is given by
$$S(\{[t_1 = t_2].Q\}, \{Q\}) = P(t_1 = t_2).$$

This transition is probable whenever there is a match expression in $\mathcal{M}_1$ and t_1 evaluates to the same value than t_2 with positive probability.

3. *Mismatch:* $[t_1 = t_2].Q \in \mathcal{M}_1$ where t_1, t_2 are closed terms and $\mathcal{M}_2 = (\mathcal{M}_1 \setminus \{[t_1 = t_2].Q\})$; the transition probability is given by

$$S(\{[t_1 = t_2].Q\}, \{\}) = P(t_1 \neq t_2).$$

This transition is probable whenever there is a match expression in $\mathcal{M}_1$ and t_1 evaluates to a different value than t_2 with positive probability.

4. *Communication*: $\{\langle m\rangle_c, (x)_c.Q\} \subset \mathcal{M}_1$; all other transitions are not probable for $\mathcal{M}_1$; $\mathcal{M}_2 = (\mathcal{M}_1 \setminus \{\langle m\rangle_c, (x)_c.Q\}) \cup \mathcal{M}_{Q^x_m}$ where Q^x_m stands for the process where we substitute all (free) occurrences of x by m in Q; This transition is probable whenever there is a pair input/output for a channel c in $\mathcal{M}_1$ and no term reduction, match or mismatch transition is probable.
5. *Termination:* $\mathcal{M}_1 = \mathcal{M}_2$; $S(\mathcal{M}_1, \mathcal{M}_2) = 1$; and all other transitions are not probable for $\mathcal{M}_1$. Whenever all reductions were made, the only enabled transition is the loop over the same state, which means that the reduction has terminated (and therefore $\mathcal{M}_1$ is an absorbing state).

Note that from a practical view point, a scheduler can be seen as a process dispatcher, that decides, based on some policy, which is the next process to reduce. Moreover, schedulers are expected to have the following good properties:

1. *Channel and variable independence*: probabilities are independent of the names of the channels and variables, that is:
   - $S(\mathcal{M}_1, \mathcal{M}_2) = S({\mathcal{M}_1}^x_y, {\mathcal{M}_2}^x_y)$ provided that x occurs free with respect to y in all processes of $\mathcal{M}_1$ and $\mathcal{M}_2$.
   - $S(\mathcal{M}_1, \mathcal{M}_2) = S({\mathcal{M}_1}^c_d, {\mathcal{M}_2}^c_d)$ where $w(d) = w(c)$ and d does not occur in all processes of $\mathcal{M}_1$ and $\mathcal{M}_2$;
2. *Environment independence*: probabilities are independent of the processes which are not involved in the transition, that is

$$S(\mathcal{M}_1, \mathcal{M}_2 | \mathcal{M} \subseteq \mathcal{M}_1 \cap \mathcal{M}_2) = S(\mathcal{M}_1 \setminus \mathcal{M}, \mathcal{M}_2 \setminus \mathcal{M}).$$

3. *Computational efficiency*: the scheduler is modeled by a probabilistic polynomial time Turing machine.

It is straightforward to check that the scheduler that gives uniform distribution to all possible transitions verifies the above properties.

As stated before, the operational semantics for a process Q is defined by restricting the scheduler.

Definition 5. Given a process Q and a scheduler S, the *operational semantics of* Q is the subMarkov chain $S(Q)$ of S consisting of all states reachable from $\mathcal{M}_Q$ such that $S(Q)^0 = \mathcal{M}_Q$ that is, the initial state of $S(Q)$ is $\mathcal{M}_Q$.

Note that the loops in $S(Q)$ are the absorbing transitions, and hence, all the states are either transient or absorbing. This fact implies that for k sufficiently large we have $P(S(Q)^k = S(Q)^{k+m}) = 1$ for all $m \in \mathbb{N}$. In other words, any random sampling of $S(Q)$ will end in an absorbing state, and therefore $S(Q)$ always terminates. This is more or less expected, since in [23] it has been shown that $S(Q)$ can be modeled by a PPT machine and so, $S(Q)$ always terminates.

2.3 Observations

In order to establish the observations of a process Q, we consider a modulated Markov process $K(Q) = (S(Q), O(Q))$ where $O(Q)$ is the stochastic process of observations of Q. The term *modulated* here means the probability distribution over the observations is computable from the distribution over the states.

This process is defined as expected: when a communication with a public channel occurs the pair (channel,output) is observed; when another type of transition occurs nothing is observed, which is modeled by the special symbol τ. Hence, the set of observations is $(C \times \mathbb{N}) \cup \{\tau\}$. Naturally, the probabilities of the observations are guided (modulated) by the probabilities on $S(Q)$. Given a scheduler S, we can obtain the global observation process $K(S, O)$ as follows:

Definition 6. Given a scheduler S and a security parameter n, we define the *observation modulated Markov process* $K = (S, O)$ where O is a stochastic process over $(U \times \mathbb{N}) \cup \{\tau\}$ such that:

- $K((\mathcal{M}_1, o_1), (\mathcal{M}_2, o_2)) = S(\mathcal{M}_1, \mathcal{M}_2)$ whenever one of the following conditions hold:
  - $o_2 = (c, m)$ and the public channel c outputs m in the transition of $\mathcal{M}_1$ to $\mathcal{M}_2$ in S (note that c can not be a fresh channel);
  - $o_2 = \tau$ and the transition from $\mathcal{M}_1$ to $\mathcal{M}_2$ in S was not a communication over a public channel.
- $K((\mathcal{M}_1, o_1), (\mathcal{M}_2, o_2)) = 0$ for all other cases.

Observe that K is indeed a Markov process modulated by S, since there exists a function f such that $K((\mathcal{M}_1, o_1), (\mathcal{M}_2, o_2)) = S(\mathcal{M}_1, \mathcal{M}_2) f(\mathcal{M}_1, \mathcal{M}_2)$. Once again, by restricting K to $S(Q)$ we obtain the required modulated stochastic process of observations $K(Q)$ for the process Q. For the sake of easing notation we denote the set of all observations by $Ob = (U \times \mathbb{N}) \cup \{\tau\}$.

In order to establish observational equivalence, we need to compute the probability of observing some output $o \in Ob$ at any point of the reduction trace. We denote this probability by $P(o \in T)$ and it can be computed as follows:

Definition 7. Given an observation process $K = (S, O)$ the probability of observing some output $o \in Ob$ at any point of the reduction trace is

$$P(o \in T) = \sum_{i=1}^{\infty} P\left(\bigwedge_{j=1}^{i} O^j \in C_j\right)$$

where $C_i = \begin{cases} \{o\} & \text{if } j = i \\ Ob \setminus \{o\} & \text{otherwise} \end{cases}$.

After fixing a scheduler S, the probability of the trace outputting o is calculated by an infinite series. Each term of the series represents the probability of o being output for the first time at the i-th reduction step. Hence, for any process Q, $T(Q)$ measures the probability of $\overline{Q}_n$ generating the output o at any point of its reduction.

Next, we present a toy example to articulate the concepts discussed above.

Example 4. Start by considering the following simple process expression Q

$$< \text{Rand} + 1 >_c \ \mid (x)_c.\langle x+1\rangle_d \mid \langle 2\rangle_d \mid (y)_d.\langle y+1\rangle_e$$

where Rand is a uniform Bernoulli random variable taking values over $\{0,1\}$ (that is, it has $\frac{1}{2}$ probability of taking value 0 or 1). The multiset of sequences of Q is

$$\mathcal{M}_Q = \{< \text{Rand} + 1 >_c, (x)_c.\langle x+1\rangle_d, \langle 2\rangle_d, (y)_d.\langle y+1\rangle_e\}.$$

We proceed by considering three different types of schedulers. For the sake of simplicity we skip over some additions.

1) Assume that the scheduler gives more priority to reducing the leftmost processes than to reducing the rightmost ones. For this particular example, we assume that $\mathcal{M}_Q$ is ordered just to express clearly which terms are reduced first by the scheduler. In that case $S(Q)$ is as follows:

$$\begin{tikzcd}
& \{\langle \text{Rand}+1\rangle, (x)_c.\langle x+1\rangle_d, \langle 2\rangle_d, (y)_d.\langle y+1\rangle_e\} \arrow[dl, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & \\
\begin{array}{c}\{\langle 1\rangle_c, (x)_c.\langle x+1\rangle_d,\\ \langle 2\rangle_d, (y)_d.\langle y+1\rangle_e\}\end{array} \arrow[d, "1"'] & & \begin{array}{c}\{\langle 2\rangle_c, (x)_c.\langle x+1\rangle_d,\\ \langle 2\rangle_d, (y)_d.\langle y+1\rangle_e\}\end{array} \arrow[d, "1"] \\
\{\langle 2\rangle_d, \langle 2\rangle_d, (y)_d.\langle y+1\rangle_e\} \arrow[d, "1"'] & & \{\langle 3\rangle_d, \langle 2\rangle_d, (y)_d.\langle y+1\rangle_e\} \arrow[d, "1"] \\
\{\langle 2\rangle_d, \langle 3\rangle_e\} \arrow[loop below, "1"] & & \{\langle 2\rangle_d, \langle 4\rangle_e\} \arrow[loop below, "1"]
\end{tikzcd}$$

The modulated observation Markov process $K(Q) = (S(Q), O(Q))$ is:

$$\begin{tikzcd}
& \{\langle \text{Rand}+1\rangle, (x)_c.\langle x+1\rangle_d, \langle 2\rangle_d, (y)_d.\langle y+1\rangle_e\} \arrow[dl, "\frac{1}{2}:\tau"'] \arrow[dr, "\frac{1}{2}:\tau"] & \\
\begin{array}{c}\{\langle 1\rangle_c, (x)_c.\langle x+1\rangle_d,\\ \langle 2\rangle_d, (y)_d.\langle y+1\rangle_e\}\end{array} \arrow[d, "1:\langle c,1\rangle"'] & & \begin{array}{c}\{\langle 2\rangle_c, (x)_c.\langle x+1\rangle_d,\\ \langle 2\rangle_d, (y)_d.\langle y+1\rangle_e\}\end{array} \arrow[d, "1:\langle c,2\rangle"] \\
\{\langle 2\rangle_d, \langle 2\rangle_d, (y)_d.\langle y+1\rangle_e\} \arrow[d, "1:\langle d,2\rangle"'] & & \{\langle 3\rangle_d, \langle 2\rangle_d, (y)_d.\langle y+1\rangle_e\} \arrow[d, "1:\langle d,3\rangle"] \\
\{\langle 2\rangle_d, \langle 3\rangle_e\} \arrow[loop below, "1:\tau"] & & \{\langle 2\rangle_d, \langle 4\rangle_e\} \arrow[loop below, "1:\tau"]
\end{tikzcd}$$

Hence, the probability of observing an output in the trace T is as presented in the following table:

$\langle c,1\rangle$	$\langle c,2\rangle$	$\langle d,2\rangle$	$\langle d,3\rangle$	τ	o
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	0

where o is any output in $Ob \setminus \{\langle c,1\rangle, \langle c,2\rangle, \langle d,2\rangle, \langle d,3\rangle, \tau\}$.

2) Now, suppose that the scheduler gives more priority to reducing the rightmost processes than to reducing the leftmost ones. Once again we assume that the multiset is ordered. In that case $S(Q)$ (together with $K(Q)$) is as follows:

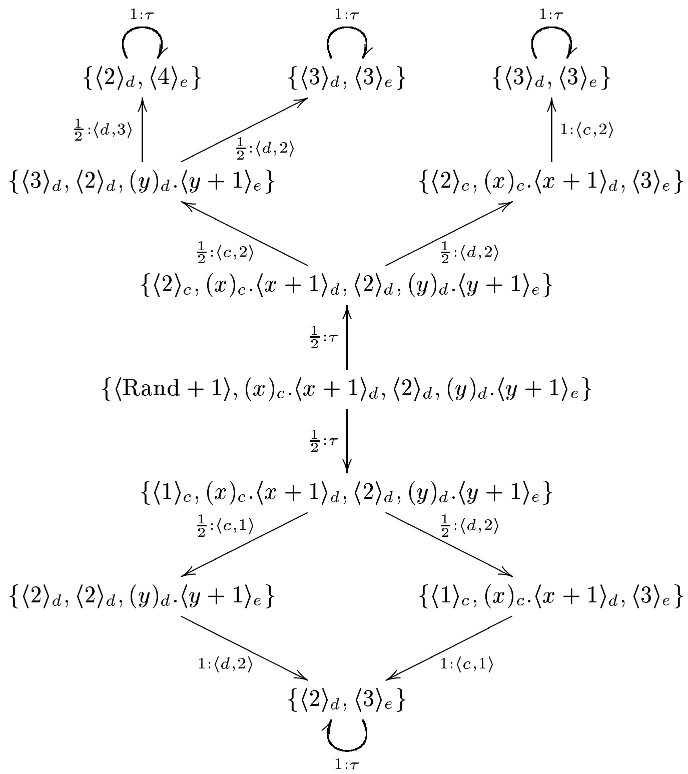
$$\begin{tikzcd}
 & \{\langle \mathrm{Rand}+1\rangle, (x)_c\cdot\langle x+1\rangle_d, \langle 2\rangle_d, (y)_d\cdot\langle y+1\rangle_e\} \arrow[dl, "\frac{1}{2}:\tau"'] \arrow[dr, "\frac{1}{2}:\tau"] & \\
\begin{array}{c}\{\langle 1\rangle_c, (x)_c\cdot\langle x+1\rangle_d,\\ \langle 2\rangle_d, (y)_d\cdot\langle y+1\rangle_e\}\end{array} \arrow[d, "1:\langle d,2\rangle"'] & & \begin{array}{c}\{\langle 2\rangle_c, (x)_c\cdot\langle x+1\rangle_d,\\ \langle 2\rangle_d, (y)_d\cdot\langle y+1\rangle_e\}\end{array} \arrow[d, "1:\langle d,2\rangle"] \\
\{\langle 1\rangle_c, (x)_c\cdot\langle x+1\rangle_d, \langle 3\rangle_e\} \arrow[d, "1:\langle c,1\rangle"'] & & \{\langle 2\rangle_c, (x)_c\cdot\langle x+1\rangle_d, \langle 3\rangle_e\} \arrow[d, "1:\langle c,2\rangle"] \\
\{\langle 2\rangle_d, \langle 3\rangle_e\} \arrow[loop below, "1:\tau"] & & \{\langle 3\rangle_d, \langle 3\rangle_e\} \arrow[loop below, "1:\tau"]
\end{tikzcd}$$

Note that even if reduction is from the right to the left, evaluating Rand has priority over any other reduction. The probability of observing an output in the trace T is:

$\langle c,1\rangle$	$\langle c,2\rangle$	$\langle d,2\rangle$	τ	o
$\frac{1}{2}$	$\frac{1}{2}$	1	1	0

where o is any output in $Ob \setminus \{\langle c,1\rangle, \langle c,2\rangle, \langle d,2\rangle, \tau\}$.

3) Finally, suppose that the scheduler chooses uniformly which processes to reduce. In that case $S(Q)$ (together with $K(Q)$) is of the following form:



The probability of observing an output in the trace T is:

$\langle c,1\rangle$	$\langle c,2\rangle$	$\langle d,2\rangle$	$\langle d,3\rangle$	τ	o
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{7}{8}$	$\frac{1}{8}$	1	0

where o is any output in $Ob \setminus \{\langle c,1\rangle, \langle c,2\rangle, \langle d,2\rangle, \langle d,3\rangle, \tau\}$.

From now on, we assume fixed once and for all a scheduler S. Thus, all random quantities mentioned in the sequel are bound to S.

2.4 Observational Equivalence

Let us now discuss the asymptotic formulation of observational equivalence for our process calculus introduced in [16, 23], which draws on two sources. One source is programming language theory with its standard notion of observational equivalence of two programs P and Q, which intuitively means that in any environment, P has the same observable behavior as Q does in that same environment.

Another source of our asymptotic formulation is the notion of computational indistinguishability by polynomial-time statistical tests, standard in cryptography [13, 30]. Intuitively, two probability distributions are computationally indistinguishable if it is not *feasible* to distinguish them. Stated somewhat more formally, this means that the two distributions cannot be distinguished, up to a negligible function of a security parameter, by probabilistic polynomial-time tests.

In order to present the asymptotic formulation of observational equivalence in detail in the setting of our calculus, we use the following definition of a *context*, intended to formalize an intuitive idea of a program environment:

Definition 8. The set of *contexts* $\mathbf{Ctx}$ is defined inductively as follows: $[\,] \in \mathbf{Ctx}$; $\nu_c.C[\,] \in \mathbf{Ctx}$ provided that $C[\,] \in \mathbf{Ctx}$; $(x)_c.C[\,] \in \mathbf{Ctx}$ provided that $C[\,] \in \mathbf{Ctx}$; $[t_1 = t_2].C[\,] \in \mathbf{Ctx}$ provided that $C[\,] \in \mathbf{Ctx}$; $C[\,]|Q \in \mathbf{Ctx}$ provided that $C[\,] \in \mathbf{Ctx}$ and $Q \in \mathcal{L}$; $Q|C[\,] \in \mathbf{Ctx}$ provided that $C[\,] \in \mathbf{Ctx}$ and $Q \in \mathcal{L}$; and $!_q C[\,] \in \mathbf{Ctx}$ provided that $C[\,] \in \mathbf{Ctx}$ and $q \in \mathbf{q}$.

Given a context $C[\,]$ and a process expression Q, the notation $C[Q]$ means that we substitute the process Q for the $[\,]$ in $C[\,]$. We recall that for each fixed value k of the security parameter, the process $\overline{C[Q]})_k$ is obtained from $C[Q]$ by replacing each subexpression $!_q R$ by $q(|k|)$ copies of R in parallel.

Let us also recall that to establish the trace process T we rely on a probabilistic polynomial-time scheduler S and the associated observation process K. Hence, any trace process T is parameterized by a probabilistic polynomial-time scheduler S.

Definition 9. Let Q_1 and Q_2 be closed process expressions. We say that Q_1 and Q_2 are *observationally equivalent* or *computationally indistinguishable* iff for every scheduler S, every context $C[\,]$, every polynomial $q()$, every observation $(\boldsymbol{u}, \boldsymbol{m})$ and n sufficiently large

$$|P(T(\overline{C[Q_1]})_n) = (\boldsymbol{u}, \boldsymbol{m})) - P(T(\overline{C[Q_2]})_n) = (\boldsymbol{u}, \boldsymbol{m}))| \leq 1/q(n).$$

In this case we write $Q_1 \simeq Q_2$.

Therefore two closed process expressions are computationally indistinguishable iff they are indistinguishable by contexts, that is, there is no context that can distinguish, up to a negligible function of the security parameter, the observable behavior of the given process expressions in that context. Intuitively, this definition merges with the standard definition of computational indistinguishability, since process expressions can be modeled by probabilistic polynomial-time Turing machines [23] and the contexts $C[\,]$ induce the required distinguishing probabilistic polynomial-time tests. However, one benefit of our process language-based approach is the following proposition:

Proposition 1. Computational indistinguishability is a congruence relation with respect to the primitives of $\mathcal{L}$.

Proof. Both symmetry and reflexivity are trivial to check. Transitivity follows by triangular inequality, and taking into account that $\frac{1}{2}q(n)$ is a polynomial. Finally, congruence on the operators follows by noticing that for any contexts $C[\ ]$ and $C'[\ ]$, $C'[C[\ ]]$ is also a context. □

3 Emulation and Composition Theorem

One rather flexible and expressive way of formulating the requirement that a given protocol satisfy some security property or fulfills a cryptographic objective or task is by relating the given protocol to an ideal protocol that clearly satisfies the property or fulfills the task. This idea appears in various forms already in [15, 6, 5, 20]. In our approach [22, 23, 17], motivated by [28, 29, 1], we formulate the relationship between the given and the ideal protocol by means of observational equivalence. It is also very useful, especially for security properties that allow protocol participants to behave in an adversarial way toward each other, to structure the notion of an ideal protocol so that the generic description of the security property or the cryptographic task itself is separated from the description of the intended adversarial behavior of the participants or even external adversaries. This may be presented by means of the so-called emulation relation [7]. Let us discuss how this method may be expressed in our process calculus framework.

Let I be a generic, ideal representation some cryptographic objective or task. One can think of I as a generic process that accomplishes the objective. Such a process I is sometimes called *a functionality.* The adversarial behavior, or the threat model, may be expressed as the kind of environment B or an *ideal adversary*, in the presence of which I is intended to be executed. In our setting the description of the environment is given by means of families of contexts.

A similar distinction may be made between actual protocols, written as process expressions Q, and their intended adversaries A which are defined as certain families of contexts. We say that a protocol Q *securely realizes the functionality* I, or that Q *emulates* I, if for any real adversary, say represented by a context $A[\ ] \in \mathcal{A}$, the trace process of $A[Q]$ is observationally equivalent to the trace process of $B[I]$ for some ideal adversary, represented by a context $B[\ ] \in \mathcal{B}$, where an ideal adversary is an adversary which cannot corrupt I. This property asserts that given a real adversary $A[\ ]$ we cannot computationally distinguish the public outputs of $A[Q]$ from the public outputs of the well-behaved process $B[I]$ for some $B[\ ] \in \mathcal{B}$. Therefore, we infer that $A[Q]$ is also well-behaved. Recall that we use outputs to model what information participants possess, so if A is able to obtain some data efficiently from Q that A should have not, then A can issue an output with such information. In this case, we would not find any ideal adversary B which is able to gather from I similar information (by choosing correctly the set $\mathcal{B}$of ideal adversaries), and hence, the trace process of $A[Q]$ is not going to be observationally equivalent from $B[I]$ for any possible ideal adversary $B[\ ] \in \mathcal{B}$.

This discussion leads to the concept of emulation with respect to a set of real adversaries $\mathcal{A}$ and ideal adversaries $\mathcal{B}$.

Definition 10. Let Q and I be closed process expressions and $\mathcal{A}$ and $\mathcal{B}$ sets of contexts, then Q *emulates* I with respect to $\mathcal{A}$ and $\mathcal{B}$ iff for all contexts $A[\ ] \in \mathcal{A}$ there exists a context $B[\ ] \in \mathcal{B}$ such that $A[Q] \simeq B[I]$. In such case we write $Q \equiv_{\mathcal{A},\mathcal{B}} I$ and say that Q is an emulation of I, or that Q is a secure implementation of I with respect to $\mathcal{A}$ and $\mathcal{B}$.

A desirable property of the emulation relation is a compositionality property, informally discussed in the setting for secure computation already in [20] and more recently in [7]. Intuitively, if Q is a secure implementation of I, if R and J are two protocols that use the ideal protocol I as a component, and if R is a secure implementation of J, then R^I_Q should be a secure implementation of J. This property may be formally captured in our process calculus as follows:

Theorem 1. Let Q, I be closed process expressions, let $J[\ ]$ and $R[\ ]$ be contexts, and let $\mathcal{A}, \mathcal{B}, \mathcal{C}$ and $\mathcal{D}$ be sets of contexts. If $R[B[I]] \equiv_{\mathcal{C},\mathcal{D}} J[B[I]]$ for any $B[\ ] \in \mathcal{B}$ and $Q \equiv_{\mathcal{A},\mathcal{B}} I$, $A[\ ] \in \mathcal{A}$ there exists $B[\ ] \in \mathcal{B}$ such that $R[A[Q]] \equiv_{\mathcal{C},\mathcal{D}} J[B[I]]$.

Proof. Let $A[\ ] \in \mathcal{A}$ and $B[\ ] \in \mathcal{B}$ be such that $A[Q] \simeq B[I]$. Now choose some $C[\ ] \in \mathcal{C}$. Clearly $C[R[A[Q]]] \simeq C[R[B[I]]]$ since $\simeq$ is a congruence relation. Moreover, since $R[B[I]] \equiv_{\mathcal{C},\mathcal{D}} J[B[I]]$, there is a $D[\ ] \in \mathcal{D}$ such that $C[R[B[I]]] \simeq D[J[B[I]]]$. Finally, by transitivity of $\simeq$, we have that $C[R[A[Q]]] \simeq D[J[B[I]]]$ and hence $R[A[Q]] \equiv_{\mathcal{C},\mathcal{D}} J[B[I]]$. □

Ideal protocols often consist of a generic, honest part I and an ideal adversary B, and are therefore of the form $B[I]$. This justifies why we consider $R[B[I]]$ in the proposition above instead of $R[I]$. Moreover, adversaries for R and J might be different from those of Q and I. Therefore, we need to consider two pairs of sets of contexts, $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{A}$, $\mathcal{B}$.

3.1 Example: Oblivious transfer

Oblivious transfer (OT) [27, 12, 14, 9] is a two-party protocol where one agent is called the *sender* and the other the *receiver*. The sender's input is a vector of k bits $\boldsymbol{b} = b_1 \dots b_k$ and the receiver's input is a number i, $1 \le i \le k$. The purpose of the protocol is, intuitively, to transfer the i-th bit $\boldsymbol{b}_i$ of the vector $\boldsymbol{b}$ to the receiver without revealing any other bit to the receiver and without revealing the address i to the sender. We will refer to these two informal security requirements as *sender security* and *receiver security*, respectively.

Following the general paradigm just discussed at the beginning of this Section, we would like to express either of these security requirements by means of observational equivalence to a certain ideal protocol, in this case, an ideal version of oblivious transfer. In an ideal setting there is a trusted and neutral third party, $T = (\boldsymbol{x})_v.(y)_{v'}.\langle \boldsymbol{x}_y \rangle_{v''}$ that expects the vector of k bits $\boldsymbol{b}$ from the sender and the value i from the receiver, and then sends $\boldsymbol{b}_i$ to the receiver, where by convention $\boldsymbol{b}_i = \boldsymbol{b}_k$ if $i \geq k$. Informally, we can think of the sender and the receiver as each communicating with T on separate private channels, or even more simply, that the sender and the receiver are subsumed into T. In any case, the only information T reveals to anyone is $\boldsymbol{x}_y$ on channel v''. T has no other outputs. T (or a copy of T with the channels renamed) is the oblivious transfer functionality.

What are the appropriate threats to consider? In the worst case the adversary may be *adaptive* [9], *i.e.,* the adversary can corrupt any of the parties at any point in response to data received during the protocol execution. We do not discuss this threat model here. Rather, we restrict ourselves to the simpler and somewhat weaker variant, the so-called *non-adaptive* or *static* adversaries [9], which can corrupt parties only once, at the beginning of the protocol execution. In this variant, it makes sense to consider several cases separately, depending on which party, the sender or the receiver, is honest and which is an adversary, and furthermore, the distinction between which is which does not change during the run of the protocol. We consider only one case, where the sender is honest and the receiver is an adversary, and in this case we are interested in sender security.

What should the static adversary receiver be able to do in the presence of the ideal oblivious transfer, the functionality T? In our setting everyone is bounded by probabilistic polynomial time, so in any case the receiver's output must be a probabilistic polynomial-time computable function, say f, of the receiver's input, i, and of any reply that the receiver can get from T, that is, one bit of the vector b. This may not be the very i-th bit because the receiver could have sent another request, j, to T in order to gain more information. But T gives only one reply so the receiver cannot learn more than one bit from T. That is, the adversary receiver's output must be of the form $f(i, b_{g(i)})$, where f and g are probabilistic polynomial-time computable functions. We will assume that an ideal adversary receiver is basically a parallel composition of processes, with one private call to a subroutine (intended to be T). The following definition describes this in a formal way:

Definition 11. An *ideal receiver adversary* is a context $B[\,]$ such that $B[T]$ is observationally equivalent to $R[T]$, where $R[\,]$ is a context of the form

$$P|(z_1)_{u_1}.\dots.(z_m)_{u_m}.(y)_{v_1}.\nu_{v'}.\nu_{v''}.(\langle t(y)\rangle_{v'}|[\,]|(z)_{v''}.Q),$$

where the other input channel of T (channel v in the description above) does not occur and where the only output not corresponding to any input is public and it occurs in Q, and this output is of the form $\langle t'(y,z)\rangle_u$, where t and t' are terms. We denote the *set of all ideal receiver adversaries* by $\mathcal{I}$.

Real adversaries that will attack a real sender have no restrictions whatsoever to the amount of information they might obtain from interacting with a real sender, other than that they must do that in probabilistic polynomial time, and that they cannot corrupt the sender. We shall assume that a real adversary is a process running in parallel with the sender, that is,

Definition 12. A *real receiver adversary* is a context of the form $[\,]|A$ We denote the set of all real receiver adversaries as $\mathcal{R}$.

We say that a protocol $Q_S|Q_R$ is a sender secure oblivious transfer protocol, if the sender Q_S running in parallel with *any* real adversary emulates the ideal setting, that is:

Definition 13. A protocol $Q_S|Q_R$ is a sender secure oblivious transfer protocol iff $Q_S \equiv_{\mathcal{R},\mathcal{I}} T$.

Note that the condition that the definition imposes only involves the real sender Q_S, not the real receiver Q_R. Furthermore, note that the correctness condition on the protocol may be expressed in a similar way, by requiring that $Q_S|Q_R$ be observationally equivalent to $R[T]$ for some *honest* ideal adversary receiver $R[\,]$ that makes an honest request to T and outputs T's reply, *i.e.,* such that $t(y) = y$ and $t'(z, y) = z_y$.

The following result is an immediate corollary of Theorem 1.

Proposition 2. The notion of sender secure is compositional. That is, let $J[\,]$ and $K[\,]$ be contexts and let $\mathcal{C}$ and $\mathcal{D}$ be sets of contexts. If $K[B[T]] \equiv_{\mathcal{C},\mathcal{D}} J[B[T]]$ for any $B[\,] \in \mathcal{I}$ and $Q_S \equiv_{\mathcal{R},\mathcal{I}} T$, then for any adversary $A \in \mathcal{R}$ there exists $B[\,] \in \mathcal{I}$ such that $K[Q_S|A] \equiv_{\mathcal{C},\mathcal{D}} J[B[T]]$.

We now present a well-known oblivious transfer protocol. We consider the version presented in [9, 14], which is an adaptation of the original protocol due to Rabin [27]. In order to establish this protocol, one needs to introduce some assumptions: a *collection of trapdoor permutations* $f = \{f_\alpha : D_\alpha \to D_\alpha\}_{\alpha \in I}$, where each f_α is probabilistic polynomial-time in n but hard to invert; a *trapdoor generator* G which is is probabilistic polynomial-time in n and generates a pair (α, t) with $\alpha \in I$; and a *hard-core predicate* B on f. Mind that for a pair (α, t) there is a function $f^{-1}(t, .)$, probabilistic polynomial-time in n, such that $f^{-1}(t, .)$ is the inverse of f_α. Moreover, a hard-core predicate $B : D_\alpha \to \{0, 1\}$ is a predicate computable in polynomial time (in its input) such that knowing $f_\alpha(x)$ does not help to predict $B(x)$, that is, it is hard to predict B from an image by f_α. We ask the reader to see [13] for more details on these assumptions.

Example 5 (Rabin OT protocol). The protocol is composed of two parallel agents, the *Sender* and the *Receiver*. First, on k-bit input $b_1 \dots b_k$, the *Sender* selects a trapdoor pair (α, t) using a term G and private channel v_S and then sends α to the *Receiver*. Upon reading an input i and receiving α, the *Receiver* chooses uniformly and independently at random k elements $e_1, \dots e_k$ of D_α and then, sends the elements $y_1 \dots y_k$ to the *Sender* with $y_i = f_\alpha(e_i)$, and $y_j = e_j$ when $j \neq i$. (Thus the receiver knows $f_\alpha^{-1}(y_i) = e_i$ but cannot predict $B(f_\alpha^{-1}(y_j))$ for any $j \neq i$.) When the *Sender* receives $y_1 \dots y_k$ it sends back the tuple $(b_j \oplus B(f^{-1}(t, y_j)))_{j \in \{1,\dots,k\}}$, where $\oplus$ denotes the usual bit addition operation. (Recall that $f^{-1}(t, y_j) = f_\alpha^{-1}(y_j)$ for every $j \in \{1, \dots, k\}$.) The *Receiver* upon receiving the tuple picks the i-th element c_i and gets b_i via $c_i \oplus B(e_i) = (b_i \oplus (B(f_\alpha^{-1}(f_\alpha(e_i)))) \oplus B(e_i) = b_i$. The protocol can be written in the process calculus as follows:

$$
\begin{aligned}
- \ S = (b_1, \ldots, b_k)_{v_0}.\nu_{v_S}(&\\
&\langle G \rangle_{v_S} |\\
&(\alpha, t)_{v_S}. (\\
&\qquad \langle \alpha \rangle_{v_2} |\\
&\qquad (y_1, \ldots, y_k)_{v_3} \langle b_1 \oplus B(f^{-1}(t, y_1)), \ldots, b_k \oplus B(f^{-1}(t, y_k)) \rangle_{v_4}\\
&\qquad)\\
&) \ ;
\end{aligned}
$$

$$
\begin{aligned}
- \ R = (i)_{v_1}.(\alpha)_{v_2}.\nu_{v_R}. (!_k \langle \mathrm{Rand}(D_\alpha) \rangle_{v_R})|&\\
&(e_1)_{v_R} \ldots (e_k)_{v_R}. (\\
&\qquad \langle e_1, \ldots, e_{i-1}, f_\alpha(e_i), e_{i+1}, \ldots, e_k \rangle_{v_3} |\\
&\qquad (c_1, \ldots, c_k)_{v_4}. \langle c_i \oplus B(e_i) \rangle_u\\
&\qquad).
\end{aligned}
$$

Proposition 3. The Rabin OT protocol is not sender secure.

Proof. The honest receiver in the protocol is too generous: an adversary receiver can easily do for each $j \neq i$ what the honest receiver in the Rabin protocol does only for i, and thus get from the sender all the b_j's. That is, consider the following real receiver adversary:

$$
\begin{aligned}
R[i] = (i)_{v_1}.(\alpha)_{v_2}.\nu_{v_R}. (!_k \langle \mathrm{Rand}(D_\alpha) \rangle_{v_R})|&\\
&(e_1)_{v_R} \ldots (e_k)_{v_R}. (\\
&\qquad \langle f(e_1), \ldots, f(e_k) \rangle_{v_3} |\\
&\qquad (c_1, \ldots, c_k)_{v_4}. \langle c_1 \oplus B(e_1), \ldots, c_k \oplus B(e_k) \rangle_u\\
&\qquad).
\end{aligned}
$$

It is easy to see that this receiver outputs *all* the b_i's. Clearly this is a successful attack by the receiver, who learns the entire input string of the sender. Formally, the output containing all the b_i's can be computationally distinguished from an output of an ideal receiver adversary, which is of the form $f(i, b_g(i))$, where f and g are probabilistic polynomial-time functions. $\square$

In Section 8 of the full paper [9] and in Section 7.4 of [14] it is shown how to *compile* this protocol into an oblivious transfer that is sender secure as well as receiver secure. A related compilation method is discussed in [8]. The details of the compiler itself can be expressed in our process calculus, but that falls beyond the scope of this paper.

4 Related work

We briefly compare our approach with related work based on interactive Turing machines [7, 9] and secure reactive systems [25, 26, 3, 4].

The approach in [7, 9] is formulated in terms of interactive Turing machines (ITM), which are basically the familiar Turing machines with several additional tapes: read-only *random input* tape, read-and-write *switch* tape (consisting of a single cell), and a pair of *communication* tapes, one read-only and the other write-only. Several ITMs can be linked through their communication tapes. The security parameter, usually written in unary, is a shared input among a collection of linked ITMs, but each ITM may have separate, additional inputs. It is assumed that the linked ITMs are polynomial-time in the security parameter. Details may be found in [7, 13].

The framework proposed in [7] involves a relationship between a *real model* representing protocol execution of actual protocols and an *ideal model* representing a generic version of a cryptographic task. A protocol in the real model *securely realizes* the task if it emulates an ideal process for the task. Either model consists of a finite set of ITMs representing the parties in the protocol, another ITM representing the protocol adversary, and yet another ITM representing the computational environment, including other protocol executions and their adversaries, other users, *etc.* The environment has external input known only to itself. The environment provides the inputs to other parties and it reads their outputs. The basic idea is that from the environment's point of view, executing the protocol in the real model should look the same as the ideal process. In somewhat more detail, a real protocol P securely realizes an ideal process I if for any real adversary A there is an ideal adversary S such that no environment can tell with non-negligible probability whether it is interacting with P and A in the real model or with I and S in the ideal model. A general composition theorem is proved in [7] and a wide variety of protocols have been studied in this framework in [7, 9] and in several other papers. A related framework based on secure reactive systems, in which ITMs are seen from the perspective of input/output automata [18] is studied in [25, 26, 3, 4].

In comparison, keeping in mind that our language of functional terms is rich enough to reflect probabilistic polynomial-time computable functions (and only those functions), functions computed by single ITMs may be represented in the framework discussed in this paper by simple process expressions, with channels representing communication tapes. Keeping in mind that our language of functional terms is rich enough to reflect probabilistic polynomial-time computable functions. Finite sets of ITMs may be represented by a parallel composition of processes, or more generally, by contexts that involve parallel composition. Adversaries and the environment are represented by contexts. In this paper we presented this in more detail in the example of oblivious transfer in the case of non-adaptive adversaries. We have investigated several other protocols in this light and we believe there is a general correspondence between our framework and the frameworks based on ITMs. In this regard it is useful to remember that any process expression in our calculus is provably executable in probabilistic polynomial-time [23]. An interesting technical point is that we consider external probabilistic polynomial-time schedulers of input/output communications on channels while in [7] the scheduling of communications is done by the adversary. It is possible, however, to force the scheduling by structuring the contexts appropriately, in particular the contexts playing the role of the adversary. This feature is already present in the specific example in the previous section.

5 Conclusions

We have expressed security requirements for cryptographic protocols in the framework of a probabilistic polynomial-time process calculus. We have also proved an abstract composition theorem for security properties. These results provide a framework for a compositional analysis of security protocols. We showed how to express an oblivious transfer protocol and its security requirements in the process calculus. Finally, we have discussed a relationship between our process calculus and the interactive Turing machine approaches in [7, 25].

There are several advantages of using a process calculus instead of interactive Turing machines. Namely, the process calculus is a much more natural and clear language for specifying protocols than the low-level vocabulary of interactive Turing machines. Indeed, the precise, formal process calculus expressions remind one of high-level programming language code and are often actually shorter than even the informal descriptions of the protocol in English, let alone the low-level details of Turing machines. Another advantage lies in the fact that compositional issues are dealt with in an intrinsic, built-in way using process calculus. Indeed, in order to show the composition theorem, it is enough to prove a congruence property for the emulation relation. The candidate for the congruence relation is obvious: if A emulates B then $C[A]$ emulates $C[B]$ for any context C. Moreover, we note that probabilistic polynomial-time process calculus provides an adequate setting for the concepts related to computational security, since both the parties and the adversaries expressed in the process calculus are provably bounded by probabilistic polynomial-time algorithms. Indeed, the work presented here may be seen as a contribution to the more general effort of giving rigorous definitions of security properties independent or particular protocols.

Acknowledgments

The authors are most grateful to Ran Canetti for several helpful comments and fruitful suggestions about an early draft of this paper, especially on the relationship between the process calculus and interactive Turing machines.

References

1. M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi-calculus. *Information and Computation*, 143:1–70, 1999.
2. M. Abadi and P. Rogaway. Reconciling two views of cryptography (The computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
3. M. Backes, C. Jacobi, and B. Pfitzmann. Deriving cryptographically sound implementations using composition and formally verified bisimulation. In *Formal Methods Europe*, volume 2931 of *Lecture Notes in Computer Science*, pages 310–329. Springer-Verlag, 2002.
4. M. Backes, B. Pfitzmann, and M. Waidner. Universally composable cryptographic library. Manuscript available on eprint.iacr.org as 2003/015, 2003.

5. D. Beaver. Foundations of secure interactive computing. In *Crypto'91*, volume 576 of *Lecture Notes in Computer Science*, pages 377–391. Springer-Verlag, 1991.
6. D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4:75–122, 1991.
7. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42-nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE Press, 2001. Full paper available at eprint.iacr.org as 2000/067.
8. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Eurocrypt'01*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer-Verlag, 2001.
9. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *34-th ACM Symposium on Theory of Computing*, pages 484–503, 2002. Full paper available at eprint.iacr.org as 2002/140.
10. D. Dolev and A. Yao. On the security of public-key protocols. In *Proc. 22-nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 350–357, 1981.
11. T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31:469–472, 1985.
12. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.
13. O. Goldreich. *Foundations of Cryptography: Basic Tools.* Cambridge Univ. Press, 2001.
14. O. Goldreich. *Foundations of Cryptography – Volume 2.* Working Draft of Chapter 7, 2003. Available at www.wisdom.weizmann.ac.il/~oded.
15. S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Crypto'90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer-Verlag, 1990.
16. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time framework for protocol analysis. In M. Reiter, editor, *5-th ACM Conferece on Computer and Communication Security*, pages 112–121. ACM Press, 1998.
17. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security analysis. In *Formal Methods in the Development of Computing Systems*, volume 1708 of *Lecture Notes in Computer Science*, pages 776–793. Spriger-Verlag, 1999.
18. N. Lynch. *Distributed Algorithms.* Morgan Kaufman, 1996.
19. P. Mateus, A. Pacheco, J. Pinto, A. Sernadas, and C. Sernadas. Probabilistic situation calculus. *Annals of Mathematics and Artificial Intelligence*, 32(1):393–431, 2001.
20. S. Micali and P. Rogaway. Secure computation. In *Crypto'91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer-Verlag, 1991.
21. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
22. J. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *39-th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 725–733. IEEE Computer Society Press, 1998.
23. J. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for analysis of cryptographic protocols. *Electronic Notes in Theoretical Computer Science*, 45, 2001.

24. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–9, 1978.
25. B. Pfitzmann, M. Schunter, and M. Waidner. Cryptographic security of reactive systems. *Electronic Notes in Theoretical Computer Science*, 32, 2000.
26. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *7-th ACM Conference on Computer and Communications Security*, pages 245–254. ACM Press, 2000.
27. M. Rabin. How to exchange secrets by oblivious transfer. Tech. memo TR-81, Aiken Computation Laboratory, Harvard U., 1981.
28. A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *8-th IEEE Computer Security Foundations Workshop (CSFW)*. IEEE Computer Society Press, 1995.
29. S. Schneider. Security properties and CSP. In *IEEE Symposium Security and Privacy*, 1996.
30. A. Yao. Theory and applications of trapdoor functions. In *23-rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91. IEEE Press, 1982.