A Framework for Analyzing Probabilistic Protocols and its Application to the Partial Secrets Exchange

Konstantinos Chatzikokolakis, Catuscia Palamidessi

A Framework for Analyzing Probabilistic Protocols and its Application to the Partial Secrets Exchange*

Konstantinos Chatzikokolakis and Catuscia Palamidessi

LIX, École Polytechnique, 91128 Palaiseau, France
{kostas,catuscia}@lix.polytechnique.fr

Abstract. We propose a probabilistic variant of the pi-calculus as a framework to specify randomized security protocols and their intended properties. In order to express an verify the correctness of the protocols, we develop a probabilistic version of the testing semantics. We then illustrate these concepts on an extended example: the Partial Secret Exchange, a protocol which uses a randomized primitive, the Oblivious Transfer, to achieve fairness of information exchange between two parties.

1 Introduction

Probabilistic security protocols involve *probabilistic choices* and are used for many purposes including signing contracts, sending certified email and protecting the anonymity of communication agents. Some probabilistic protocols rely on specific random primitives such as the *Oblivious Transfer* ([13]). There are various examples in this category, notably the contract signing protocol in [5] and the privacy-preserving auction protocol in [8].

A large effort has been dedicated to the formal verification of security protocols, and several approaches based on process-calculi techniques have been proposed. However, in the particular case of probabilistic protocols, they have been analyzes mainly by using model checking methods, while only few attempts of applying process calculi techniques have been made. The only work we are aware of is [2], which defines a probabilistic version of the noninterference property, and uses a probabilistic variant of CCS and of bisimulation to analyse protocols wrt this property.

In this paper we present a framework for analyzing probabilistic security protocols using the π_{prob}-calculus, a probabilistic extension of the π-calculus inspired by the work in [6]. In order to express security properties in this calculus, we extend the notion of testing equivalence ([9]) to the probabilistic setting. We propose a preorder based on the probability of passing a certain class of tests: a process P is considered smaller than a process Q if, for each test, the probability

* Supported by the Project Rossignol of the ACI Sécurité Informatique (Ministère de la recherche et nouvelles technologies)

of passing it is smaller for P than for Q. Following the lines of [1], a test can be seen as an adversary who interacts with an agent in order to break some security property. In order to verify a security property, then, we can create a specification which satisfies it and show that the protocol is smaller than the specification with respect to the testing preorder. If this holds, then the adversary has smaller probability of succeeding with the protocol than with the specification, so the protocol is correct with respect to the examined property.

We illustrate the above described framework with an extended example of fair exchange protocol, where the property to verify is fairness. In this type of protocols two agents, A and B, want to exchange information simultaneously, that is each of them is willing to send its secrets only if he receives the ones of the other party. We consider the Partial Secrets Exchange protocol (PSE, [5]) which uses the Oblivious Transfer as its main primitive. An important characteristic of fair exchange protocols is that the adversary is in fact one of the agents and not an external party. As a consequence the behavior of A will be different when B behaves normally from when he is trying to cheat. After encoding the protocol in the π_{prob}-calculus, we give a specification which models the behavior of A in case he is being cheated. We then express fairness by means of a testing relation between the protocol and the specification and we prove that it holds.

The rest of the paper is organized as follows: in the next section we introduce π_{prob}, our variant of the probabilistic π-calculus. We present its semantics and propose a notion of probabilistic testing preorder. In Section 3 we illustrate the Oblivious Transfer primitive, the Partial Secrets Exchange protocol (PSE), and their encoding in the π_{prob}-calculus. In Section 4 we specify the fairness property and we prove the correctness of PSE. In Section 5 we discuss related work, notably the analysis of the PSE protocol using probabilistic model checking. Finally, Section 6 concludes and presents some ideas for future work.

For reasons of space, the proofs have been removed from the main text. They can be found in the report version of this paper ([3]).

2 A probabilistic variant of π-calculus

In this section we define a probabilistic process calculus suitable for implementing security protocols. This calculus, which will be referred as the π_{prob}-calculus, is a probabilistic extension of the π-calculus, similar to the probabilistic asynchronous π-calculus presented in [6].

A common feature of π_{prob} and the calculus in [6] is that there is a distinction between probabilistic and deterministic behavior. The former, represented by the choice operator, is associated with the random choices performed by the process itself. The latter, represented by the parallel operator, is related to the decisions of an external scheduler.

The π_{prob}-calculus differs from the calculus in [6] in that it allows only blind choices. This simplifies considerably semantics and reasoning, while the calculus remains rich enough to model probabilistic security protocols. Furthermore,

the π_{prob}-calculus contains some extra constructs, like output prefix and pair splitting, that are useful to express the protocols we have considered.

We could also add certain cryptographic primitives like the shared-key encryption of the spi-calculus, however this wasn't necessary in the protocols examined so far.

2.1 Syntax

Let x, y range over a countable set of variables and n, m over a countable set of *channel names*. The terms and processes of the π_{prob}-calculus are defined by the following grammar:

$M, N ::=$	**terms**	$P, Q ::=$	**processes**
x	variable	$\overline{M}N.P$	output
$\mid n$	name	$\mid M(x).P$	input
$\mid \langle M, N \rangle$	pair	$\mid P \mid Q$	composition
		$\mid \sum_i p_i P_i$	prob. choice
		$\mid \nu n P$	restriction
		$\mid\ !P$	replication
		$\mid [M \text{ is } N]P$	match
		$\mid$ let $\langle x, y \rangle = M$ in P	pair splitting
		$\mid\ 0$	nil

The distinction between variables and channel names does not exist in the original π-calculus but simplifies the treatment of some relations.

2.2 Probabilistic automata

The semantics of π_{prob} is based on Probabilistic Automata, which were introduced in [14]. We briefly recall here the main notions, simplified and adapted for our needs.

A *discrete probabilistic space* is a pair (X, pb) where X is a set and pb a function $pb : X \mapsto (0, 1]$ s.t. $\sum_{x \in X} pb(x) = 1$. Given a set Y we define the set of all probabilistic spaces on Y:

$$Prob(Y) = \{(X, pb) \mid X \subseteq Y \text{ and } (X, pb) \text{ is a discrete probabilistic space}\}$$

Let S be a set of states and A a set of actions. A *probabilistic automaton* is a triple $(\mathcal{S}, \mathcal{T}, s_0)$ where $s_0 \in S$ (initial state) and $\mathcal{T} \subseteq S \times Prob(A \times S)$. The elements of $\mathcal{T}$ are called *transition groups*. The idea is that the choice between transition groups is made non-deterministically by an external scheduler while the choice of a transition within a group is made probabilistically by the process itself.

Given a probabilistic automaton $M = (\mathcal{S}, \mathcal{T}, s_0)$ we define $tree(M)$ as the tree obtained by unfolding the transition system. The root n_0 of $tree(M)$ is

$$\text{IN} \quad m(x).P \ \{\xrightarrow[1]{m(x)} P\} \qquad \text{OUT} \quad \overline{m}M.P \ \{\xrightarrow[1]{\overline{m}M} P\}$$

$$\text{SUM} \quad \textstyle\sum_i p_i P_i \ \{\xrightarrow[p_i]{\tau} P_i\}_i \qquad \text{OPEN} \ \frac{P \ \{\xrightarrow[1]{\overline{m}n} P'\}}{\nu n P \ \{\xrightarrow[1]{\overline{m}(n)} P'\}} \quad m \neq n$$

$$\text{RES1} \ \frac{P \ \{\xrightarrow[1]{\mu} P'\}}{\nu n P \ \{\xrightarrow[1]{\mu} \nu n P'\}} \ {\scriptstyle \mu \neq \tau, \ n \notin nm(\mu)} \qquad \text{RES2} \ \frac{P \ \{\xrightarrow[p_i]{\tau} P_i\}_i}{\nu n P \ \{\xrightarrow[p_i]{\tau} \nu n P_i\}_i}$$

$$\text{COM} \ \frac{P \ \{\xrightarrow[1]{\overline{m}M} P'\} \qquad Q \ \{\xrightarrow[1]{m(x)} Q'\}}{P \mid Q \ \{\xrightarrow[1]{\tau} P' \mid Q'[M/x]\}} \qquad \text{PAR} \ \frac{P \ \{\xrightarrow[p_i]{\mu_i} P_i\}_i}{P \mid Q \ \{\xrightarrow[p_i]{\mu_i} P_i \mid Q\}_i} \quad \forall i \ fn(\mu_i) \cap bn(Q) = \emptyset$$

$$\text{CLOSE} \ \frac{P \ \{\xrightarrow[1]{\overline{m}(n)} P'\} \qquad Q \ \{\xrightarrow[1]{m(x)} Q'\}}{P \mid Q \ \{\xrightarrow[1]{\tau} \nu n(P' \mid Q'[n/x])\}} \qquad \text{CONG} \ \frac{P \equiv P' \quad P' \ \{\xrightarrow[p_i]{\mu_i} Q'_i\}_i \quad \forall i. Q'_i \equiv Q_i}{P \ \{\xrightarrow[p_i]{\mu_i} Q_i\}_i}$$

Fig. 1. The late-instantiation semantics of the π_{prob}-calculus. The functions *fn*,*bn* and *nm* give the free, bound and total names of their argument respectively.

labeled by s_0 and if n is a node labeled by s then for each $(s, (X, pb)) \in \mathcal{T}$ and each $(\mu, s') \in X$ there is a node n' labeled by s' and an arc from n to n' labeled by μ and $pb(\mu, s')$.

A *scheduler* ζ is a function which solves the nondeterminism by selecting, at each moment of the computation, a transition group among the ones allowed at the current state. The *execution tree* of an automaton M under a scheduler ζ, denoted by $etree(M, \zeta)$ is the tree obtained from $tree(M)$ by pruning all the arcs corresponding to transitions in groups not selected by ζ.

2.3 Semantics of π_{prob}

The operational semantics of the π_{prob}-calculus is given by means of probabilistic automata defined inductively on the basis of the syntax. In order to simplify the notation, we write

$$s \ \{\xrightarrow[p_i]{\mu_i} s_i \mid i \in I\}$$

iff $(s, (\{(\mu_i, s_i) \mid i \in I\}, pb)) \in \mathcal{T}$ and $\forall i \in I : p_i = pb(\mu_i, s_i)$, where I is an index set. When I is not relevant we will use the notation $s \ \{\xrightarrow[p_i]{\mu_i} s_i\}_i$.

The transitions of the automaton associated to a process are defined by the rules in Figure 1.

The behavior of the choice operator is defined by the SUM rule. The transition to every member of the sum is possible with a τ action (blind choice). Note

that all transitions belong to the same group which means that the choice is not controlled by the scheduler but is made by the process itself. IN and OUT are self-explanatory. RES models restriction on channel n: actions on that channel are not allowed by the restricted process. Note that we have split this rule in two for the sake of clarity: for the transition groups which contain only τ actions there is no need to check the channel name. PAR models interleaving, in which each process maintains its transition groups. COM models communication by handshaking. Since input/output transitions are always alone in their group, this rule is rather simple and very similar to the non-probabilistic case. CLOSE is similar to COM but works together with OPEN in order to implement scope extrusion, that is the transfer of a new channel name between processes. Finally CONG states that equivalent processes perform the same actions. The *structural equivalence* $\equiv$ used in CONG is defined as follows:

$$(\alpha\text{-renaming}) \quad P \equiv Q \quad \text{iff } P \equiv_\alpha Q \qquad\qquad P \mid Q \equiv Q \mid P$$
$$P \mid 0 \equiv P \qquad\qquad !P \equiv P \mid !P$$
$$\text{let } \langle x, y\rangle = \langle M, N\rangle \text{ in } P \equiv P[M/x][N/y] \qquad [M \text{ is } M]P \equiv P$$

In the following sections we define some relations between π_{prob} processes which will help us expressing some properties of probabilistic protocols and reasoning about them. We will also examine some properties of these relations.

2.4 Testing relations between π_{prob} processes

Testing is a well-known method of comparing processes, resulting in equivalences weaker than the ones of the bisimulation family. The idea, proposed by De Nicola and Hennessy ([9]), is that two processes are equivalent if they both pass the same set of tests. A *test* is a process running in parallel with the one being tested and which can perform a distinguished action ω which represents success. This idea is very useful for the analysis of security protocols, as suggested in [1], since a test can be seen as an adversary who interferes with a communication agent and declares his success with an ω action. Then two processes are testing equivalent if they are vulnerable to the same attacks.

In the probabilistic setting there are different approaches for defining testing equivalence. For example [12] proposes a probabilistic extension of testing equivalence which considers the ability of each process to pass a test with non-zero probability (may testing) or probability one (must testing). However, when analyzing security protocols we are not only interested in the ability of passing a test, but also in the exact probability of success. Thus our definition resembles more to the one of [7] and the result is no longer an equivalence but a preorder.

We start by defining the probability of a set of executions. Given a probabilistic automaton M and a scheduler ζ, an *execution fragment* ξ is a path (finite or infinite) from the root of $etree(M, \zeta)$. The probability of an execution fragment $\xi = n_0 \xrightarrow[p_0]{\mu_0} n_1 \xrightarrow[p_1]{\mu_1} n_2 \xrightarrow[p_2]{\mu_2} \dots$ is defined as $pb(\xi) = \prod_i p_i$. An *execution* is a maximal execution fragment. The set of all executions of M under ζ is denoted by $exec(M, \zeta)$.

Given an execution fragment ξ, a *cone* with prefix ξ is defined as $C_\xi = \{\xi' \in exec(M,\zeta) \mid \xi \leq \xi'\}$ where $\leq$ is the prefix relation. We define $pb(C_\xi) = pb(\xi)$. Let $\{C_i\}_{i\in I}$ be a countable set of disjoint cones. We define $pb(\bigcup_{i\in I} C_i) = \sum_{i\in I} pb(C_i)$. We can show that this probability is well defined, that is two different sets of disjoint cones with the same union give the same probability.

A *test* O is a π_{prob}-calculus process able to perform a distinguished action ω. An *interaction* between O and a process P is a sequence of τ transitions starting from $P|O$. In order to allow only τ actions we define $\nu P = \nu n_1 \ldots \nu n_k P$, where $n_1, \ldots, n_k$ are all the free names in P. Then an interaction between P and O is an element of $exec(\nu(P|O),\zeta)$[1]:

$$\nu(P|O) = Q_0 \xrightarrow[p_0]{\tau} Q_1 \xrightarrow[p_1]{\tau} Q_2 \xrightarrow[p_2]{\tau} \ldots$$

An interaction ξ is *successful* if $Q_i \xrightarrow[p]{\omega}$ for some i. Let $sexec(\nu(P|O),\zeta) = \{\xi \in exec(\nu(P|O),\zeta) \mid \xi \text{ is successful}\}$. This set can be obtained as a countable union of disjoint cones [6], so the probability of a successful execution can be defined as $pb(sexec(\nu(P|O),\zeta))$.

We now define the upper and lower probability for P to pass O.

Definition 1. *Let P be a process and O a test. We define*

$$P\lceil O\rceil = \sup\{pb(sexec(\nu(P \mid O),\zeta)) \mid \zeta \textit{ is a scheduler}\}$$
$$P\lfloor O\rfloor = \inf\{pb(sexec(\nu(P \mid O),\zeta)) \mid \zeta \textit{ is a scheduler}\}$$

Then we define the testing preorders for π_{prob}-processes.

Definition 2. *Let P, Q be processes. We define must and may-testing preorders as follows:*

$$P \sqsubseteq_{\textbf{may}} Q \qquad \textit{iff for all tests } O : P\lceil O\rceil \leq Q\lceil O\rceil$$
$$P \sqsubseteq_{\textbf{must}} Q \qquad \textit{iff for all tests } O : P\lfloor O\rfloor \leq Q\lfloor O\rfloor$$

In this paper we will only use may-testing to express safety properties of security protocols, so we will write just $\sqsubseteq$ for $\sqsubseteq_{\textbf{may}}$.

As we will see in the following sections, an agent of a probabilistic security protocol behaves differently when his partner deviates from the protocol in an attempt to cheat. In order to model this behavior we introduce a *conditional testing preorder*, which is exactly the same as may-testing except that it only considers tests that satisfy a certain condition.

Definition 3. *Let P, Q be processes. We define the conditional may-testing preorder as follows:*

$$P \sqsubseteq^{\phi} Q \quad \textit{iff for all tests } O : \phi(O) \Rightarrow P\lceil O\rceil \leq Q\lceil O\rceil$$

where $\phi(O)$ is a condition on O.

[1] With a slight abuse of notation we will sometimes use a process to denote its corresponding probabilistic automaton.

Finally we define a useful preorder between pairs of processes:

Definition 4. *Let P_1, P_2, Q_1, Q_2 be processes. We define the relation $\sqsubseteq_p$ between pairs of processes as follows*

$$(P_1, P_2) \sqsubseteq_p (Q_1, Q_2) \quad \textit{iff} \quad P_1 +_p P_2 \sqsubseteq Q_1 +_p Q_2$$

where $P_1 +_p P_2$ stands for $\sum_{i=1}^{2} p_i P_i$ with $p_1 = p$ and $p_2 = 1 - p$.

2.5 Properties of testing preorders

In this section we examine some properties of the previously defined relations. We present only the corresponding lemmas, all proofs can be found in [3].

The following lemma is very useful for reasoning about the upper probability of passing a test. It crucially relies on the fact that in π_{prob} probabilistic choices are blind.

Lemma 1. *Let P, Q be π_{prob} processes and $p \in [0, 1]$. Then for all tests O*

$$P +_p Q\lceil O \rceil = pP\lceil O \rceil + (1 - p)Q\lceil O \rceil$$

A *context* C is a process containing a "hole". We will denote by $C[P]$ the process obtained by replacing the hole in C by P. A preorder is a *precongruence* if it is closed under any context.

May-testing is not a precongruence on arbitrary processes since for $P = [x \text{ is } y]P', Q = [x \text{ is } z]Q', C = n(x).[\,]$, we have $P \sqsubseteq Q$ but $C[P] \sqsubseteq C[Q]$ does not hold for all P', Q'. However all previous relations become precongruences if we restrict to closed processes.

Definition 5. *A process is called* closed *if it contains no free variables.*

Remark 1. Because of the distinction between variables and channel names, a closed process can still have free channel names and therefore be able to communicate with the environment.

Lemma 2. *$\sqsubseteq_p$ is a precongruence on closed processes, that is for all contexts C and all closed processes P_1, P_2, Q_1, Q_2*

$$(P_1, P_2) \sqsubseteq_p (Q_1, Q_2) \Rightarrow (C[P_1], C[P_2]) \sqsubseteq_p (C[Q_1], C[Q_2])$$

The following corollary is an immediate consequence of lemmas 1 and 2.

Corollary 1. *$\sqsubseteq$ is a precongruence on closed processes.*

3 Probabilistic Security Protocols

In this section we discuss probabilistic security protocols based on the Oblivious Transfer and we show how to model them using the π_{prob}-calculus.

3.1 1-out-of-2 Oblivious Transfer

The Oblivious Transfer is a primitive operation used in various probabilistic security protocols. In this particular version a sender A sends exactly one of the messages M_1, M_2 to a receiver B. The latter receives i and M_i where i is 1 or 2, each with probability 1/2. Moreover A should get no information about which message was received by B. More precisely the protocol $\mathrm{OT}^1_2(A, B, M_1, M_2)$ should satisfy the following conditions:

1. If A executes $\mathrm{OT}^1_2(A, B, M_1, M_2)$ properly then B receives exactly one message, $(1, M_1)$ or $(2, M_2)$, each with probability 1/2.
2. After the execution of $\mathrm{OT}^1_2(A, B, M_1, M_2)$, if it is properly executed, for A the probability that B got M_i remains 1/2.
3. If A deviates from the protocol, in order to increase his probability of learning what B received, then B can detect his attempt with probability at least 1/2.

It is worth noting that in the literature the reception of the index i by B is often not mentioned, at least not explicitly ([5]). However, omitting the index can lead to possible attacks. Consider the case where A executes (properly) $\mathrm{OT}^1_2(M_1, M_1)$. Then B will receive M_1 with probability one, but he cannot distinguish it from the case where he receives M_1 as a result of $\mathrm{OT}^1_2(M_1, M_2)$. So A is forcing B to receive M_1. We will see that, in the case of the PSE protocol, A could exploit this situation in order to get an unfair advantage. Note that the condition 3 does not apply to this situation since this cannot be considered as a deviation from the Oblivious Transfer. A generic implementation of the Oblivious Transfer could not detect such behavior since A executes OT properly, the problem lies only in the data being transfered.

Using the indexes, however, solves the problem since B will receive $(2, M_1)$ with probability one half. This is distinguishable from any outcome of $\mathrm{OT}^1_2(M_1, M_1)$ so, in the case of PSE, B could detect that he's being cheated. Implementations of the Oblivious Transfer do provide the index information, even though sometimes it is not mentioned ([5]). In other formulations of the OT the receiver can actually select which message he wants to receive, so this problem is irrelevant.

Encoding in the π_{prob}-calculus. The Oblivious Transfer can be implemented in the π_{prob}-calculus, using the probabilistic choice operator. In order to make it impossible to cheat, a server process is used to coordinate the transfer. The processes of the sender and the server are the following:

$$\mathrm{OT}^1_2(m_1, m_2, c_{as}) \triangleq \overline{c_{as}}m_1.\overline{c_{as}}m_2.0$$
$$S(c_{as}, c_{sb}) \triangleq c_{as}(m_1).c_{as}(m_2).(\overline{c_{bs}}\langle 1, m_1\rangle +_{0.5} \overline{c_{bs}}\langle 2, m_2\rangle)$$

where m_1, m_2 are the names to be sent. c_{as} is a channel private to A and S and c_{sb} a channel private to B and S. Each agent communicates only with the server

```
PSE (A, B, {a_i}_i, {b_i}_i) {
    for i = 1 to n do
        OT^1_2(A,B, a_i, a_{i+n})
        OT^1_2(B, A, b_i, b_{i+n})
    next
    for j = 1 to m do
        for i = 1 to 2n do
            A sends jth bit of a_i to B
        for i = 1 to 2n do
            B sends jth bit of b_i to B
    next
}
```

Fig. 2. Partial Secrets Exchange protocol

and not directly with the other agent. B receives the message from the server (which should be in parallel with A and B) by making an input action on c_{sb}.

It is easy to see that these processes correctly implement the Oblivious Transfer. The only requirement is that A should not contain c_{sb}, so that he can only communicate with B through the server.

3.2 Partial Secrets Exchange Protocol

This protocol is the core of three probabilistic protocols for contract signing, certified email and coin tossing, all presented in [5]. It involves two agents, each having $2n$ secrets split in pairs, $(a_1, a_{n+1}), ..., (a_n, a_{2n})$ for A and $(b_1, b_{n+1}), ...,$ (b_n, b_{2n}) for B. Each secret consists of m bits. The purpose is to exchange a single pair of secrets under the constraint that, if at a specific time B has one of A's pairs, then with high probability A should also have one of B's pairs and vice versa.

The protocol, displayed in figure 2, consists of two parts. During the first A and B exchange their pairs of secrets using OT^1_2. After this step A knows exactly one half of each of B's pairs and vice versa. During the second part, all secrets are exchanged bit per bit. Half of the bits received are already known from the first step, so both agents can check whether they are valid. Obviously, if both A and B execute the protocol properly then all secrets are revealed.

The problem arises when B tries to cheat and sends incorrectly some of his secrets. In this case it can be proved that with high probability some of the tests of A will fail causing A to stop the execution of the protocol and avoid revealing his secrets. The idea is that, in order for B to cheat, he must send at least one half of each of his pairs incorrectly. However he cannot know which of the two halves is already received by A during the first part of the protocol. So a pair sent incorrectly will have only one half probability of being accepted by A, leading to a total 2^{-n} probability of success.

Now imagine, as discussed in section 3.1, that B executes $\mathrm{OT}^1_2(B, A, b_i, b_i)$, thus forcing A to receive b_i. Now, in the second part, he can send all $\{b_{i+n} \mid 1 \leq$

$i \leq n\}$ incorrectly without failing any test. Moreover A cannot detect this situation. If indexes are available A will receive $(2, b_{i+n})$ with probability one half and since he knows that b_{i+n} is not the second half of the corresponding pair he will stop the protocol.

Encoding in the π_{prob}-calculus. In this paragraph we present an encoding of the PSE protocol in the π_{prob}-calculus. Before giving the corresponding process there are two points worth discussing.

- The secrets exchanged by PSE should be *recognizable*, which means that agent A cannot compute B's secrets, but he can recognize them when he receives them. Of course a secret can be recognized only as a whole, no single bit can be recognized by itself. To implement this feature we allow B's secrets to appear in A's process, as if A knew them. However we allow a secret to appear only as a whole (not decomposed) and only inside a test construct, which means that it can only be used to recognize another message.
- In our analysis we need to detect the fact that an agent sends a specific bit in a certain position of a specific message. Thus, in the implementation of PSE, each parameter a_{ij} (resp. b_{ij}) is considered to take values from the domain $\{0_{ij}, 1_{ij}\}$, where 0_{ij} (resp. 1_{ij}) is a public channel but different for each i, j.
  Note that having secrets composed by public bits can lead to guessing attacks by non-deterministic adversaries. Many analysis tools for security protocols, such as the spi-calculus, do not allow the decomposition of secrets to avoid such guesses. In our analysis, however, we express the correctness of a protocol as the equivalence with a properly constructed specification. This only proves that the protocol will not *reveal* any secrets and is not related with the adversary's ability of *guessing* the secrets without interfering with any partner (of course, this is known to happen with very small probability). Such attacks will apply to both the protocol and the specification.

The encoding for the general case of n pairs and m bits per message is displayed in figure 3. We denote by a_i (resp. b_i) the i-th secret of A (resp. B) and by a_{ij} (resp. b_{ij}) the j-th bit of a_i (resp. b_i). r_i is the i-th message received by Oblivious Transfer and k_i is the corresponding index.

The first part consists of the first 3 lines of the process definition. In this part A sends his pairs using OT^1_2, receives the ones of B and decomposes them. To check the received messages A starts a loop of n steps, each of whom is guarded by an input action on q_i for synchronization. During the i-th step, r_i is tested against b_i or b_{i+n} depending on the outcome of the OT, that is on the value of k_i.

The second part consists of a loop of m steps, each of whom is guarded by an input action on s_j. During each step the j-th bit of each secret is sent and the corresponding bits of B are received in d_{ij}. Then there is nested loop of n tests controlled by the input actions on t_i. Each test, performed by the $Test$ subprocess, ensures that B's bits are valid. $Test(i, j)$ checks the j-th bit of the

$$
\begin{array}{l}
A(\{a_{ij}\}_{i=1..2n,j=1..m}, \{b_i\}_{i=1..2n}) \triangleq \\
\quad \prod_{i=1}^{n} \mathrm{OT}_2^1(\langle a_{i1}, \ldots, a_{im}\rangle, \langle a_{(i+n)1}, \ldots, a_{(i+n)m}\rangle, c_{as_i}) \mid \\
\quad c_{sa_1}(\langle k_1, r_1\rangle).\text{let } \langle r_{11}, \ldots, r_{1m}\rangle = r_1 \text{ in } \ldots c_{sa_n}(\langle k_n, r_n\rangle).\text{let } \langle r_{n1}, \ldots, r_{nm}\rangle = r_n \text{ in} \\
\quad \nu q_1 \ldots \nu q_{n+1}(\overline{q_1}\star \mid \prod_{i=1}^{n} q_i(x).TestOT(i) \mid \\
\qquad q_{n+1}(x).\nu s_1 \ldots \nu s_{m+1}(\overline{s_1}\star \mid \\
\qquad \prod_{j=1}^{m} s_j(x).\overline{c_p}a_{1j}. \ldots \overline{c_p}a_{(2n)j}.c_p(d_{1j}). \ldots c_p(d_{(2n)j}). \\
\qquad\qquad \nu t_1 \ldots \nu t_{n+1}(\overline{t_1}\star \mid \prod_{i=1}^{n} t_i(x).Test(i,j) \mid t_{n+1}(x).\overline{s_{j+1}}\star) \mid \\
\qquad s_{m+1}(x).\overline{c_p}ok))
\end{array}
$$

$$TestOT(i) \triangleq ([k_i \text{ is } 1][r_i \text{ is } b_i]\overline{q_{i+1}}\star \mid [k_i \text{ is } 2][r_i \text{ is } b_{i+n}]\overline{q_{i+1}}\star)$$

$$Test(i,j) \triangleq ([k_i \text{ is } 1][r_{ij} \text{ is } d_{ij}]\overline{t_{i+1}}\star \mid [k_i \text{ is } 2][r_{ij} \text{ is } d_{(i+n)j}]\overline{t_{i+1}}\star)$$

Fig. 3. Encoding of PSE protocol

i-th pair. The bit received during the first part, namely r_{ij}, is compared to d_{ij} or $d_{(i+n)j}$ depending on k_i. If the bit is valid, an output action on t_{i+j} is performed to continue to the next test.

Finally, an instance of the protocol is an agent A put in parallel with servers for all oblivious transfers:

$$I \triangleq A(\{a_{ij}\}_{i=1..2n,j=1..m}, \{b_i\}_{i=1..2n}) \mid \prod_{i=1}^{n}(S(c_{as_i}, c_{sb_i}) \mid S(c_{bs_i}, c_{sa_i}))$$

4 Verification of Security Properties

A well known method for expressing and proving security properties using process calculi is by means of *specifications*. A specification P_{spec} of a protocol P is a process which is simple enough in order to prove (or accept) that it models the correct behavior of the protocol. Then the correctness of P is implied by $P \simeq P_{spec}$ where $\simeq$ is a testing equivalence. The idea is that, if there exists an attack for P, this attack can be modeled by a test O which performs the attack and outputs ω if it succeeds. Then P should pass the test and since $P \simeq P_{spec}$, P_{spec} should also pass it, which is a contradiction (no attack exists for P_{spec}).

However, in case of probabilistic protocols, attacks do exist but only succeed with a very small probability. So examining only the ability of passing a test is not sufficient since the fact that P_{spec} has an attack is no longer contradictory. Instead we will use a specification which can be shown to have very small probability of been attacked and we will express the correctness of P as $P \sqsubseteq P_{spec}$ where $\sqsubseteq$ is the testing preorder defined in section 2.4. Then an attack of high probability for P should be applicable with at least the same probability for P_{spec} which is contradictory.

4.1 A specification for PSE

Let us recall the fairness property for the PSE protocol.

> If B receives one of A's pairs then with high probability A should also be able to receive one of B's pairs.

First of all we must point out two important differences between this type of protocols and the traditional cryptographic ones.

- In traditional protocols both A and B are considered honest. The purpose of the protocol is to ensure that no outside adversary can access the messages being transfered.
  On the other hand, in PSE the adversary is B himself, who might try to deviate from the protocol in order to get A's secrets without revealing his own ones.
- In traditional protocols the secrets must remain secret all the time. A and B always perform the same actions and always want to communicate with each other.
  On the other hand in PSE A shows different behavior when B is honest than in case of an attempt to cheat. A is willing to reveal his secrets, only when B wants the same too.

A specification of a protocol shows the correct behavior of the agents. Since A's behavior depends on B it makes sense to have different specifications depending on B's behavior. Since the case where B is honest is trivial, we are considering the case where B tries to deviate from the protocol. That is B will try to send some of his bits incorrectly. Moreover the behavior of A depends on which these bits are. If B stays honest for the first half bits then A will do the same.

It order to model B's intention to cheat, we will use a function $h : \{1..n\} \mapsto \{1..m\}$ that shows on which bit B is going to cheat for each pair. So $h(3) = 4$ means that B is going to send the 4th bit of (at least) one of the 3rd pair's secrets incorrectly. We consider "cheating" to be a deviation from the protocol in a way that leads to a violation of fairness. Thus, in order for B to cheat h must be defined on its whole domain. The goal is to exchange just one pair, if at least one pair is sent correctly by B then fairness is not violated.

The specification is displayed in figure 4. As already discussed, it depends on B's cheating behavior, that is on the function h. The specification resembles a lot the protocol, with two major differences:

1. The specification does not use any of its input (all input variables are replaced by x to point out this fact). Moreover b_i's are no longer used (thus they are removed from the parameter list).
2. The specification does not test the received bits. In the first part, $TestOT_{spec}$ accepts all messages. In the second, $Test_{spec}$ accepts all bits, except those on which B is known to cheat, which are accepted only with probability one half.

$$
\begin{array}{l}
A_{spec}(\{a_{ij}\}_{i=1..2n,j=1..m}, h) \triangleq \\
\quad \prod_{i=1}^{n} \mathrm{OT}_2^1(\langle a_{i1}, \ldots, a_{im}\rangle, \langle a_{(i+n)1}, \ldots, a_{(i+n)m}\rangle, c_{as_i}) \mid \\
\quad c_{sa_1}(x) \ldots c_{sa_n}(x). \\
\quad \nu q_1 \ldots \nu q_{n+1}(\overline{q_1}\star \mid \prod_{i=1}^{n} q_i(x).TestOT_{spec}(i) \mid \\
\qquad q_{n+1}(x).\nu s_1 \ldots \nu s_{m+1}(\overline{s_1}\star \mid \\
\qquad \prod_{j=1}^{m} s_j(x).\overline{c_p}a_{1j}. \ldots \overline{c_p}a_{(2n)j}.c_p(x). \ldots c_p(x). \\
\qquad\qquad \nu t_1 \ldots \nu t_{n+1}(\overline{t_1}\star \mid \prod_{i=1}^{n} t_i(x).Test_{spec}(i,j,h) \mid t_{n+1}(x).\overline{s_{j+1}}\star) \mid \\
\qquad s_{m+1}(x).\overline{c_p}ok))
\end{array}
$$

$$
TestOT_{spec}(i) \triangleq \overline{q_{i+1}}\star
$$

$$
Test_{spec}(i,j,h) \triangleq \begin{cases} \overline{t_{i+1}}\star +_{0.5} 0 & \text{if } h(i) = j \\ \overline{t_{i+1}}\star & \text{otherwise} \end{cases}
$$

Fig. 4. Specification for the PSE protocol

As a consequence the specification is much simpler than the protocol. As we will show in the next section, if h is total then A will make n choices and the probability of succeeding in all of them will be negligible.

4.2 Proving the correctness of PSE

Correctness of the specification. First we show that the specification is indeed a proper specification for PSE with respect to fairness, in case B tries to cheat. Let l be the maximum number of bits that B is willing to reveal for its secrets. So B's cheating behavior will be described by a function h, such that $h(i) \leq l+1$ for all $i \in \{1..n\}$. This is by definition of PSE, otherwise B would reveal $l+1$ bits of at least one pair of secrets and one pair is enough for A.

As we already discussed A_{spec} does not depend on its input. Moreover it is deterministic, that is only one transition is possible at any moment, except from $Test_{spec}(i,j,h)$ for $h(i) = j$ where the process stalls with probability one half. Since $h(i) \leq l+1, \forall i \in \{1..n\}$, all n of these tests will appear in the first $l+1$ steps of the second part of the protocol. If A fails in even one test then he stalls, so the total probability of advancing to step $l+2$ and reveal its $l+2$ pair is 2^{-n}.

This means that A_{spec} satisfies fairness. If B at some point of the protocol has l bits of one of A's pairs, then with probability at least $1-2^{-n}$ A will have $l-1$ bits of at least one of B's pairs. If $l = m$ (B has a whole pair) then A should have at least $m-1$ bits and the last bit can be easily computed by trying both 0 and 1. In other words B cannot gain an advantage of more than one bit with probability greater than 2^{-n}.

Relation between A and A_{spec}. Having proved the correctness of the specification with respect to fairness, it remains to show its relation with the original protocol.

Proving $A \sqsubseteq A_{spec}$ means to prove that if A is vulnerable with high probability to an attack O, then A_{spec} will be also vulnerable with at least the same probability. Since we know that the probability of a successful attack for A_{spec} is very small, we can conclude that an attack on A is very unlikely.

Note however that A_{spec} models the behavior of A only in case of an attack described by the function h. So $A \sqsubseteq A_{spec}$ cannot hold in general since A will pass with greater probability a test which models an honest agent. Thus, we need to use the conditional may-testing defined in section 2.4.

An instance of the specification is a process A_{spec} put in parallel with servers for all oblivious transfers:

$$I_{spec}(h) \triangleq A(\{a_{ij}\}_{i=1..2n,j=1..m}, h) \mid \prod_{i=1}^{n}(S(c_{as_i}, c_{sb_i}) \mid S(c_{bs_i}, c_{sa_i}))$$

Let H be the set of all total functions $h : \{1..n\} \mapsto \{1..m\}$. PSE will be considered correct wrt fairness if:

$$\begin{aligned}&\forall h \in H : I \sqsubseteq^{\phi_h} I_{spec}(h)\\ &\text{where } \phi_h(O) = true \text{ iff } \forall i \in \{1..n\} :\\ &\qquad O \text{ does not contain both } b_{ih(i)} \text{ and } b_{(i+n)h(i)}\end{aligned}$$

The condition ϕ_h ensures that the test will try to cheat on the $h(i)$-th bit of each pair i. The idea is that in order to cheat, an intruder should refuse to send at least one bit of each message. It can be proved that $\sqsubseteq^{\phi_h}$, for the specific condition ϕ_h described above, is a precongrunce on closed processes wrt the contexts that satisfy ϕ_h. More details can be found in [3].

We can now state the correctness of PSE, as defined above.

Theorem 1. *PSE is correct with respect to fairness.*

5 Related Work

Security protocols have been extensively studied during the last decade and many formal methods have been proposed for their analysis. However, the vast majority of these methods refer to non-deterministic protocols and are not suitable for the probabilistic setting, since they do not allow to model random choices. One exception is the work of Aldini and Gorrieri ([2]), where they use a probabilistic process algebra to analyze fairness in a non-reputation protocol. Their work is close to ours in spirit, although technically it is quite different. In particular, we base our analysis on a notion of testing while theirs is based on a notion of bisimulation.

With respect to the application, the results the most related to ours come from Norman and Schmatikov ([10], [11]), who use probabilistic model checking to study fairness in two probabilistic protocols, including the Partial Exchange Protocol. In particular, in [11] they model the PSE using Prism, a probabilistic model checker. Their treatment however is very different from ours: their model

describes only the "correct" behavior for both A and B, as specified by the protocol. B's ability to cheat is limited to prematurely stopping the execution, so attacks in which B deviates completely from the protocol are not taken into account. Having a simplified model is important in model checking since it helps overcoming the search state explosion problem, thus making the verification feasible.

The results in [11] show that with probability one B can gain a one bit advantage, that is he can get all m bits of a pair of A by revealing only $m-1$ bits of his. This is achieved simply by stopping the execution after receiving the last bit from A. Moreover a method of overcoming the problem is proposed, which gives this advantage to A or B, each with probability one half. Is is worth noting that this is a very weak form of attack and could be considered as negligible, since A can compute the last bit very easily by trying both 0 and 1. Besides a one bit advantage will always exist in contract signing protocols, simply because synchronous communication is not feasible.

In our approach, by modeling an adversary as an arbitrary π_{prob} process we allow him to perform a vast range of attacks including sending messages, performing calculations, monitoring public channels etc. Our analysis shows not only that a one bit attack is possible, but more important that no attack to obtain an advantage of two or more bits exists with non-negligible probability. Moreover our method has the advantage of being easily extendable. For example, treating more sessions, even an infinite number of ones, can be done by putting many copies of the processes in parallel.

Of course, the major advantage of the model checking approach, with respect to ours, is that it can be totally automated.

6 Conclusion

In this paper we examined a method to analyze probabilistic security protocols using process calculi. The main tool for this analysis is the π_{prob}-calculus, a probabilistic variant of the π-calculus. The probabilistic choice, provided by π_{prob}, allowed us to encode the Partial Exchange Protocol, a probabilistic protocol based on the Oblivious Transfer. In order to prove the correctness of this protocol, we defined various preorders between π_{prob} processes and examined their properties. Then we presented a properly constructed specification and showed that it is stronger than the original protocol, thus proving that the possibility of success for any attack is very small.

Our results show that process calculi techniques can be successfully applied to security protocol analysis. There are various advantages of this approach. First of all the use of process calculi allows the use of various tools from the corresponding theory. The proofs obtained are general, covering every possible adversary and are not instance-based as in model checking techniques. Moreover process calculi allow the analysis of a protocol in a more complex environment, having for example many agents and multiple simultaneous instances of a protocol. It is

worth noting that many attacks of well known protocols only appear in such situations.

In [4] an algorithm for deciding may-testing is presented, for fully probabilistic automata. We believe that this result can be extended to the probabilistic automata defined in section 1.2, giving the ability of automatically proving the correctness of probabilistic security protocols.

References

1. Martin Abadi and Andrew Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.
2. Alessandro Aldini and Roberto Gorrieri. Security analysis of a probabilistic nonrepudiation protocol. In Holger Hermanns and Roberto Segala, editors, *Process Algebra and Probabilist Methods. Performance Modeling and Verification: Second Joint International Workshop PAPM-PROBMIV 2002, Copenhagen, Denmark, July 25–26, 2002. Proceedings*, volume 2399 of *Lecture Notes in Computer Science*, page 17, Heidelberg, 2002. Springer-Verlag.
3. Konstantinos Chatzikokolakis and Catuscia Palamidessi. A framework for analyzing probabilistic protocols and its application to the partial secrets exchange. *Report version, available at http://www.lix.polytechnique.fr/ catuscia/papers/PartialSecrets/report.pdf*, 2005.
4. L. Christoff and I. Christoff. Efficient algorithms for verification of equivalences for probabilistic processes. In Larsen and Skou, editors, *Proc. Workshop on Computer Aided Verification*, volume 575 of *LNCS*. Springer Verlag, 1991.
5. Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.
6. Oltea Mihaela Herescu and Catuscia Palamidessi. Probabilistic asynchronous π-calculus. In Jerzy Tiuryn, editor, *Proceedings of FOSSACS 2000 (Part of ETAPS 2000)*, Lecture Notes in Computer Science, pages 146–160. Springer-Verlag, 2000.
7. Bengt Jonsson, Kim G. Larsen, and Wang Yi. Probabilistic extensions of process algebras. *Handbook of Process Algebras*, 2001.
8. Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM Press, 1999.
9. R. De Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
10. Gethin Norman and Vitaly Shmatikov. Analysis of probabilistic contract signing. In A. Abdallah, P. Ryan, and S. Schneider, editors, *Proc. BCS-FACS Formal Aspects of Security (FASec'02)*, volume 2629 of *LNCS*, pages 81–96. Springer, 2003.
11. Gethin Norman and Vitaly Shmatikov. Analysis of probabilistic contract signing. *Formal Aspects of Computing (to appear)*, 2005.
12. Catuscia Palamidessi and Oltea M. Herescu. A randomized encoding of the pi-calculus with mixed choice. In *Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science*, pages 537–549, 2002.
13. M Rabin. How to exchange secrets by oblivious transfer. *Technical Memo TR-81, Aiken Computation Laboratory, Harvard University*, 1981.
14. Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, Summer 1995.