The Full Abstraction of the UC Framework

Jesús F. Almansa
jfa@brics.dk

BRICS[*], Department of Computer Science
University of Aarhus
IT-parken, Aabogade 34
DK-8200 Aarhus N, Denmark

Abstract

We prove that security in the Universal Composability framework (UC) is equivalent to security in the probabilistic polynomial time calculus ppc. Security is defined under active and adaptive adversaries with synchronous and authenticated communication. In detail, we define an encoding from machines in UC to processes in ppc and show it is fully abstract with respect to UC-security and ppc-security, i.e., we show a protocol is UC-secure iff its encoding is ppc-secure. However, we restrict security in ppc to be quantified not over all possible contexts, but over those induced by UC-environments under encoding. This result is not overly-simplifying security in ppc, since the threat and communication models we assume are meaningful in both practice and theory.

[*]Basic Research in Computer Science (`www.brics.dk`), funded by the Danish National Research Foundation

Contents

1 Introduction

For as long as at least two decades there has been a gap in the analysis of security protocols as approached by different research communities with distinct methods: notably process calculi and (probabilistic) machine-based.

Calculi appeal especially because they are susceptible to automated proof-construction or at least automated proof-validation. However, they have traditionally fallen short of modeling realistic adversarial behaviours, thus disabling them as frameworks for reasoning about general security.

In contrast, probabilistic machine-based methods have traditionally better captured realistic adversarial threats. However, proofs are by nature hand-made, thus error-prone when applied to challenging scenarios like large distributed systems.

Because security-reasoning is subject to specifying an (realistic as it can be) adversarial model, given the above said, it seems natural to look for bridging the gap in the direction from calculi to the machine-based field, aiming at soundness and completeness results. Nonetheless, several attempts in this line have focused on specific security concerns, mainly secure message transmission, and/or on restricted adversarial models, mainly passive adversaries.

In this document we take the other direction for bridging the gap and show the existence of a fully-abstract encoding of a machine-based framework into a process-calculus framework, under active and adaptive adversaries, with authenticated and synchronous communication.

We emphasize these are frameworks for reasoning about *any* protocol-security concerns. The term fully-abstract refers to the soundness and completeness of the notion of security in the calculus with respect to that in the machine-based framework. The adversarial model is active and adaptive, meaning respectively that a corrupt party can arbitrarily deviate from the protocol and corruptions can occur at any point of the computation.

The machine-based framework is that introduced by Canetti in [Can01], where secure protocols enjoy general concurrent composition or, as it is called there, Universal Composability (UC). All computing entities are modeled as probabilistic polynomial time (PPT) Interactive Turing Machines (ITM's), which are basically PPT multi-tape Turing machines, that can exchange data by sharing some of their tapes. It adopts the simulatability paradigm to define security, where a protocol-run is compared to a simulated-run that has access to an ideal functionality (another ITM) defining the input/output behaviour of the protocol. The security definition states a protocol realizes the ideal functionality if no adversarial environment can tell apart interactions with the protocol and those with the simulation that uses the ideal functionality.

The calculus is the probabilistic polynomial-time process calculus ppc by Mateus, Mitchell and Scedrov in [MMS03], preceded by [LMMS99] and

[LMMS98], where a compositional property is derived. It adopts the simulatability paradigm to define security, where a real-process is compared to an ideal-process that simulates its data-exchange behaviour. The security definition states that real and ideal process are equivalent if no context can tell apart data-exchange with the real process and that with the ideal process.

In [Can01] it had already been stated how that framework could serve as a computationally-sound basis for automated analysis with formal methods. With the advent of [MMS03] it was left open the question whether there was a correspondence between their PPT process calculus and the UC framework. We therefore answer this question in affirmative and moreover prove that a protocol is UC secure if and only if its encoding is ppc secure, subject to the adversarial model mentioned above.

We observe that to that end we restrict security in ppc to contexts induced by UC environments under encoding, instead of quantifying over any context. But we argue that this is a natural restriction to make due to our basic assumptions on the adversarial and computation models.

An Intuition The interactive setting of the UC framework can be thought of as a laboratory we use to carry out our experiments under controlled and unbiased conditions. In particular, it relies on an interleaving mode of concurrency. Similarly, linking tapes do not model communication channels of real protocols.

This is an importan clarificatory observation since it is this interactive setting what we encode in the ppc calculus. As an effect, we separate from the traditional notion of attacks over channels on process calculi, in order to introduce that of corruption of parties.

We believe this is a major step towards automation of security proofs in the UC framework. Having translated the framework once and for all, protocol design reduces to providing the code of parties, and security analysis reduces to study patterns of observable traces with the existent tools for process calculi.

Related Work Many research efforts stemmed from the process calculus approach. Among the most relevant is the project in [MMS98, LMMS98, LMMS99, MMS03], where PPT processes are considered. This is the source of the calculus we base our result on.

A parallel and independent work to ours is that in [DKM$^+$04, DKMR04, DKMR05], where they prove the equivalence of several simulatability-based security definitions in any process calculus satisfying certain axioms. Their result concentrates on three essential aspects of the simulatability paradigm when aiming at composition-preserving properties: The logical structure of security definitions (or the order of quantifiers), the scheduling or timing assumptions of adversaries and the adversarial attributes of distinguishers (or separation between adversaries and distinguishers). Although they relate their results to machine-based models in terms of expressivity they do not formally provide an evidence[1].

Their work and ours are thus complementary in that we treat explicitely the three above mentioned aspects, while crossing from a PPT machine-based model to a process-calculus approach, thus closing a gap up to now existing in literature.

Finally, the work by Backes, Pfitzmann and Waidner [PW00, BPW04], stands out as an alternative model for simulatability-based security analysis. By formalizing the notion of a reactive system using input/output automata, they have also achieved general concurrent composition properties while at the same time already provided tools for automated reasoning.

Organization The rest of this document is organized as follows. Section 2 begins with a general discussion about ITM's and proceeds by describing the UC framework and its security definition. Section 3 describes the process calculus ppc and its security definition. Section 4 introduces an encoding from UC machines into ppc processes and ends by stating our main result and providing its proof.

Acknowledgments The author would like to thank Marco Carbone, Ivan Damgård, Karl Krukow, Peter D. Mosses, and Mogens Nielsen for numerous discussions and invaluable suggestions and insights. Likewise, I express my gratitude to Frank Valencia and Daniele Varacca for their comments on early versions or this document.

2 UC framework

2.1 Interactive Setting

Following [Can01], we adapt the notion of ITM from [GMR89, Gol01] to the multiparty case. An ITM $\mathcal{E}$ is a PPT multi-tape Turing Machine that interacts by sharing some of its tapes. We use $!_{tape_{\mathcal{E},\mathcal{E}'}}\ v$ to indicate the action of writing v on the (write-only) tape named $tape_{\mathcal{E},\mathcal{E}'}$. Similarly, we use $?_{tape_{\mathcal{E}'',\mathcal{E}}}\ v$ to indicate the action of reading v from the (read-only) tape named $tape_{\mathcal{E}'',\mathcal{E}}$.

We say $\mathcal{E}_1$, $\mathcal{E}_2$ are *linked* by (*linking*) tapes $tape_{\mathcal{E}_1,\mathcal{E}_2}$ and $tape_{\mathcal{E}_2,\mathcal{E}_1}$, if whenever $\mathcal{E}_1$ executes $!_{tape_{\mathcal{E}_1,\mathcal{E}_2}}\ v$ then $\mathcal{E}_2$ *will* execute $?_{tape_{\mathcal{E}_1,\mathcal{E}_2}}\ v$, and vice-versa with the other tape. A machine's *signature* is the set of its linking tapes.

The execution-mode is interleaving, i.e., only one machine is *activated* at a time. This is modeled by a tape *activate* shared by all machines and containing

[1]To the best of the author's knowledge, this is still the case when [DKMR04] made transit to [DKMR05].

the identity of the currently running machine. A machine that is not activated is said to be *waiting*.

The behaviour of an ITM $\mathcal{E}$ is defined by its associated *interactive function* $(\vec{v}', \varsigma') \leftarrow \mathcal{E}(\vec{v}, \varsigma)$. It takes $(\vec{v}, \varsigma)$ as argument, formed by the vector of received values found at the moment of its activation, and its current state. It computes $(\vec{v}', \varsigma')$, consisting of the values to be sent at the end of its activation, and its next state. The initial state contains only the system's security parameter, the machine's random tape and possibly some auxiliary input.

We define $\mathcal{E}(\vec{v}, \varsigma)$ as a (finite) set of interactive rules of form

$$\frac{?_{rtape_1,\ldots,rtape_R}\ v_1,\ldots,v_R}{\substack{(\vec{v}',\varsigma')\leftarrow\mathcal{E}(\vec{v},\varsigma)\\ !_{wtape_1,\ldots,wtape_W}\ v'_1,\ldots,v'_W}}\ ,$$

according to the values $\vec{v}$ can take. Such a rule is read: Upon activation, if $\mathcal{E}$ has received values $\vec{v} = v_1, \ldots, v_R$, with v_i on $rtape_i$, $i = 1, \ldots, R$, and has current state ς, then compute $\mathcal{E}(\vec{v}, \varsigma)$, to result in values $\vec{v}' = v'_1, \ldots, v'_W$ and new state ς', write v'_j on $wtape_j$ sequentially, and pick machine $\mathcal{E}_j$, $j = 1, \ldots, W$, to be activated next.

Thus, to describe an ITM $\mathcal{E}$ is sufficient to provide its signature and its set of interactive rules.

We define an *n-party interactive protocol* as consisting of n ITM's $P_1, \ldots, P_n$ having distinct *id*'s, that are *not* linked among them. Such protocol is geared towards holding an ongoing interaction with an environment, which is explicitely modeled by an ITM $\mathcal{Z}$ that is linked to each party. With this definition we are to embed communication between each two parties through interactions with the environment. This setting shows particularly useful when providing adversarial attributes to the environment.

2.2 Security Definition

Security is defined by comparing a protocol execution with an ideal process that *specifies* its input/output (I/O) interaction with the environment. We declare security if no environment can distinguish interactions with the real protocol from those with the ideal process.

2.2.1 The Hybrid Model

We adopt the activation model of [DN03, Nie04], which is a specialization of the UC framework to synchronous networks. For space reasons we consider only the *∅-hybrid model* or *real-life* model, but we observe that our result holds straightforwardly in the more general case.

The model consists of an n-party interactive protocol π, where P_i is the identity of the i-th machine running in π. The environment $\mathcal{Z}$ provides inputs to and receives outputs from the parties. Furthermore, it models point-to-point open but authenticated channels between each pair of parties and is assumed to provide (partially) synchronous communication.

This entity also models the adversary and so it schedules the order of activation of parties, and breaks into parties actively and adaptively. We emphasize that although $\mathcal{Z}$ reads the messages exchanged between parties, it is not allowed to inject, delete or modify any message.

Definition 1 *The signature $\Sigma(\mathcal{E})$ of $\mathcal{E} \in \{\mathcal{Z}, P_i\}$, $i = 1, \ldots, n$, in the $\varnothing$-hybrid model is defined as follows, where -r-,-w- stand for* read-only *and* write-only*:*

$$\begin{array}{lll} \Sigma(\mathcal{Z}) \overset{def}{=} \{ & \text{-}r\text{-}outByP_i, & \text{-}w\text{-}inToP_i, \\ & \text{-}r\text{-}msgByP_i, & \text{-}w\text{-}msgToP_i, \\ & \text{-}r\text{-}corrByP_i, & \text{-}w\text{-}corrToP_i \ \} \\ \Sigma(P_i) \overset{def}{=} \{ & \text{-}r\text{-}inToP_i, & \text{-}w\text{-}outByP_i, \\ & \text{-}r\text{-}msgToP_i, & \text{-}w\text{-}msgByP_i, \\ & \text{-}r\text{-}corrToP_i, & \text{-}w\text{-}corrByP_i \ \} \end{array}$$

The environment is activated first and the execution proceeds by rounds, where $\mathcal{Z}$ runs commands (*beg_rnd*) and (*end_rnd*), to begin and end a round, respectively. It can also run commands (*corrupt*) and (*guess*), to corrupt some party and to decide its observation of the execution and halt, respectively. The output of the $\varnothing$-hybrid execution is a bit b output by $\mathcal{Z}$. The interactive rules appear in Fig 2 in Appendix, where it is formally defined how $\mathcal{Z}$ runs commands, how parties respond to them, and what linking tapes are used.

Notation 2 *We denote $\mathtt{HYB}^{\varnothing}_{\pi,\mathcal{Z}}(k, z, \vec{\mathfrak{r}})$, the bit output by $\mathcal{Z}$ on security parameter k, auxiliary input z to $\mathcal{Z}$ and random input $\vec{\mathfrak{r}} = \mathfrak{r}_{\mathcal{Z}}, \mathfrak{r}_1, \ldots, \mathfrak{r}_n$. This gives rise to a random variable $\mathtt{HYB}^{\varnothing}_{\pi,\mathcal{Z}}(k, z)$, when $\vec{\mathfrak{r}}$ is uniformly chosen. We denote $\mathtt{HYB}^{\varnothing}_{\pi,\mathcal{Z}}$ the ensemble $\{\mathtt{HYB}^{\varnothing}_{\pi,\mathcal{Z}}(k, z)\}_{k \in \mathbb{N}, z \in \{0,1\}^*}$.*

2.2.2 The Ideal Process

As mentioned in the begining of this section, security is defined with respect to an ideal functionality, ITM $\mathcal{F}$, that specifies the I/O behaviour of the protocol. It also specifies information allowed to be *leaked* from the protocol.

We can think of $\mathcal{F}$ as all P_i's comprised in one single machine. In fact, there are no parties in the ideal process. A special ITM $\mathcal{T}$ called *interface* is introduced to respond on behalf of parties to commands by $\mathcal{Z}$.

We say π is secure if there exists an interface $\mathcal{T}$ capable of simulating a run of π maintaining its I/O behaviour having access to $\mathcal{F}$. However, we note the interface works by being given solely the corrupt parties' inputs as well as all leaked information from $\mathcal{F}$. This captures both secrecy and correctness of the protocol.

The signature of $\mathcal{Z}$ is the same as in the $\varnothing$-hybrid model, but we include it in the next definition for the sake of completeness.

Definition 3 *The signature $\Sigma(\mathcal{E})$ of $\mathcal{E} \in \{\mathcal{Z}, \mathcal{F}, \mathcal{T}\}$ in the ideal process is defined as follows, where -r-,-w- stand for* read-only *and* write-only*, and $i = 1, \ldots, n$:*

$$\begin{array}{lll}
\Sigma(\mathcal{Z}) \overset{def}{=} \{ & \textit{-r-outBy}P_i, & \textit{-w-inTo}P_i, \\
& \textit{-r-msgBy}P_i, & \textit{-w-msgTo}P_i, \\
& \textit{-r-corrBy}P_i, & \textit{-w-corrTo}P_i \ \} \\
\Sigma(\mathcal{T}) \overset{def}{=} \{ & \textit{-r-msgTo}P_i, & \textit{-w-msgBy}P_i, \\
& \textit{-r-corrTo}P_i, & \textit{-w-corrBy}P_i, \\
& \textit{-r-leaked}, & \textit{-w-probe} \ \} \\
\Sigma(\mathcal{F}) \overset{def}{=} \{ & \textit{-r-inTo}P_i, & \textit{-w-outBy}P_i, \\
& \textit{-r-probe}, & \textit{-w-leaked} \ \}
\end{array}$$

The output of the ideal process is a bit b output by $\mathcal{Z}$. Fig. 3 in Appendix formally describes the simulation.

Notation 4 *We denote $\texttt{IDEAL}_{\mathcal{F},\mathcal{T},\mathcal{Z}}(k, z, \vec{\mathfrak{r}})$, the bit output by $\mathcal{Z}$ on security parameter k, auxiliary input z to $\mathcal{Z}$ and random input $\vec{\mathfrak{r}} = \mathfrak{r}_{\mathcal{Z}}, \mathfrak{r}_{\mathcal{T}}, \mathfrak{r}_{\mathcal{F}}$. This gives rise to a random variable $\texttt{IDEAL}_{\mathcal{F},\mathcal{T},\mathcal{Z}}(k, z)$ when $\vec{\mathfrak{r}}$ is uniformly chosen. We denote $\texttt{IDEAL}_{\mathcal{F},\mathcal{T},\mathcal{Z}}$ the ensemble $\{\texttt{IDEAL}_{\mathcal{F},\mathcal{T},\mathcal{Z}}(k,\ z)\}_{k\in\mathbb{N}, z\in\{0,1\}^*}$.*

In our next definition, $\overset{c}{\approx}$ stands for computational indistinguishability.

Definition 5 (UC Security) *A protocol π t-securely realizes a functionality $\mathcal{F}$ (in the $\varnothing$-hybrid model) if there exists an interface $\mathcal{T}$ such that for all environments $\mathcal{Z}$ corrupting at most t parties it holds that $\texttt{IDEAL}_{\mathcal{F},\mathcal{T},\mathcal{Z}} \overset{c}{\approx} \texttt{HYB}^{\varnothing}_{\pi,\mathcal{Z}}$.*

We finish this section by stating two properties about the interactive pattern of machines in the UC framework.

Proposition 1 *In both the $\varnothing$-hybrid model and the ideal process it holds that:*

i) No execution leads to a deadlock.

ii) For each machine other than $\mathcal{Z}$, there is one and only one interactive rule to be applied upon activation. For $\mathcal{Z}$ this is also the case except for the corruption rule and the guessing rule which can always be applied.

3 Probabilistic calculus ppc

We provide an overview of the probabilistic polynomial-time calculus ppc. The reader is referred to [MMS03] for full details.

The calculus is given a security parameter k and sets *Var*, *Chan* of variables and channels. Its set *Term* of *terms* represents all PPT functions on k, $PPT(k)$, and only these. Moreover, if $t_f \in Term$ corresponds to $f \in PPT(k)$, then the probability that $t_f(x)$ evaluates to some a is defined as the probability that the associated machine M_f computing f converges to a on input x, i.e., $P(t_f(x) \to a) \overset{\mathsf{def}}{=} P(a \leftarrow M_f(x,k))$.

In addition, there is a *bandwidth map* $w : Chan \to \mathbf{q}$, where $\mathbf{q}$ is the set of all $\mathbb{N}$-valued polynomials in one variable. This map bounds the length of data sent through channels and guarantees that computations are polynomial time.

We use the synchronous version of ppc with input choice. Its *expressions* are of the following kind: *Inaction* 0, *private channel* $\nu_c.Q$, *output* $\langle t\rangle_c.Q$, *input* $(x)_c.Q$, *matching* $[t_1 = t_2].Q$, *parallel* $Q_1|Q_2$, *input-choice* $\langle t_1\rangle_{c1}.Q_1{+}\langle t_2\rangle_{c2}.Q_2$, and *replication* $!_qQ$, where q is a polynomial in $|k|$, and the expression is expanded to polynomially in $|k|$ many copies of Q. An expression Q where the security parameter takes a fix natural number m is called a *process* and denoted Q_m.

A *free variable* is one not in the scope of any $(\cdot)_c$. A process is *closed* if it has no free variables.

It is customary to assume that private channels are named apart from other called *public.* Typically, private channels are used for modeling internal data exchange in a single machine, hence it is considered an unobservable behaviour. In contrast, data exchange through public channels models the observable behaviour. We will come back to this later.

The meaning of a process Q is a Markov chain over multisets of subprocesses of Q without parallel operator. Recall a Markov chain can be modelled as a state machine, where transiting from one state to another is assigned a probability and where the fan-out probabilities for each state sum up to one. Thus, the intuition for the Markov chain of a process is that states are stages of reduction and transitions are probabilistic choices between reductions.

The only two sources of probabilities in ppc are evaluation of terms and selection among simultaneous process reductions. The first case was already mentioned above. For the second case, a policy of reduction has to be fixed, so that the next process to be reduced can be selected *probabilistically.*

Given a process Q, a Markov chain $S(Q)$ where a policy of reduction has been fixed is called a *scheduler.* The initial multiset (state) is obtained from Q by expanding all replications and next removing all parallel operators. Private channel operators are also removed by renaming private channels with fresh channels, from a set distinct to *Chan*, that conserve the bandwidth. In the sequel, unless the contrary is indicated, we will assume all replication and private operators have been removed.

The elegible processes for reduction are terms, matchings (mismatchings) and communications, in this order. It is defined that only communications over public channels are *observable.* They are pairs $\langle c, v \rangle$ of public channels and values. Any other reductions are *silent* actions and are modeled by τ. Private communications thus are silent. The set Ob includes both τ and all observable communications.

By establishing that τ actions have higher priority of reduction than public communication, it is ensured that an adversary will not distinguish two processes by sampling their internal (unobservable) behaviour.

Finally, in order to determine which observations take place in a process (out of the set Ob of all possible observations), a *modulated Markov process* $K(Q) = (S(Q), O(Q))$ is given, where $S(Q)$ is a fix scheduler and $O(Q)$ is the stochastic process of observations of Q over Ob. Roughly, $K(Q)$ is like $S(Q)$, but assigns probability zero to any other observation in Ob not ocurring in $S(Q)$. Details on this are out of the scope of this document and can be found in [MMS03].

Given $K(Q)$, the probability of observing some $o \in Ob$ at any point of the reduction trace is written $P(Tr(Q)_k^S = o)$. It is an infinite series indexed $i = 1, 2, \ldots$, where series-term i is the probability that o is output for the first time at the i-th reduction step. Notice random variable Tr is parameterized with the security parameter k and scheduler S. We abbreviate $Tr(Q)$ the random ensemble $\{Tr(Q)_k^S\}_{k \in \mathbb{N}, S}$.

For the next definition recall contexts are processes with holes.

Definition 6 *A closed process Q_1 is observationally equivalent to another Q_2, written $Q_1 \simeq Q_2$, iff for all schedulers and for all contexts $C[\ ]$ it holds that $Tr(C[Q_1]) \overset{c}{\approx} Tr(C[Q_2])$.*

The definition of protocol security follows by relating a protocol with an ideal specification and showing they are observationally equivalent, with respect to classes $\mathcal{A}, \mathcal{B}$ of contexts modeling real-process and ideal-process adversaries, respectively.

Definition 7 ([MMS03]) *A closed process Q* emulates *an ideal specification closed process I with respect to sets of contexts $\mathcal{A}$ and $\mathcal{B}$, written $Q \equiv_{\mathcal{A},\mathcal{B}} I$, iff for all $A[\ ] \in \mathcal{A}$ there exists $B[\ ] \in \mathcal{B}$ such that $A[Q] \simeq B[I]$.*

We note definition 7, seems to demand the class $\mathcal{B}$ of ideal-specification adversaries is defined independently from the class $\mathcal{A}$ of real-process adversaries. In fact, it does not really make sense to reason about an *a priori* class of ideal-process adversaries. While it is true that the specification I of a cryptographic task is provided with respect to and adversarial model, this model is but a generic description of threats (to later prove the attacks can be thwarted) that the calculus should be able to express but independently on whether they mean ideal-process attacks or real-process attacks.

For instance, an adaptive adversary is one that can make a corruption at any point of the reduction (execution). The term "at any moment" is what defines adaptiveness and should be expressed by the calculus, but it does not depend on whether the reduction is in an ideal or a real process. Clearly, "to make a corruption" is something to be defined too.

Only once the threat model is set, the notion of security is captured by taking any real-process adversarial strategy (constrained to the threat model) and proving there is some ideal-process adversarial strategy (constrained to the same threat model) but deemed unsuccessful in the ideal-process reduction, hence showing that a real-process is as secure as the ideal-process specification it emulates. As we will see later, a technical contribution of our work shows that different threat models can be expressed in the calculus, since they can already be expressed as (adversarial) machine behaviours that can be translated via encoding.

Under the above considerations, it would just suffice to drop $\mathcal{A}, \mathcal{B}$ in definition 7:

Definition 7' (Revised) *Fix a threat model. A closed process Q* emulates *an ideal specification closed process I, written $Q \equiv I$, iff for all $A[\,]$ bound to the threat model there exists $B[\,]$ bound to the same threat model such that $A[Q] \simeq B[I]$.*

One has to be careful with the meaning of quantifying over *all* contexts $C[\,]$ in definition 7'. There, adversaries are separate entities from distinguishers, and so the process-interaction a context $C[\,]$ sustains is limited to (public data-exchange) providing inputs and receiving outputs but *without* the capability of corrupting.

We are precisely now in a position similar to that in [Can01][2] where it could be argued there is not loss of generality in allowing distinguishers model adversaries too, since they both are contexts. Let us present the alternative security definition before resuming our discussion.

Definition 8 *Fix a threat model. A closed process Q* emulates *an ideal specification closed process I iff there exists a context $D[\,]$ bound to the threat model such that $Q \simeq D[I]$, where observational equivalence is defined with respect to all contexts whose corrupting capabilities are bound to the threat model.*

Contexts $C[\,]$ in the definition above could be thought of as having the form $C'[A'[\,]]$ where $A'[\,]$ models its corrupting capabilities (bound to the underlying threat model) and $C'[\,]$ models its reactive feature (of providing inputs and receiving outputs).

Notice well here that definitions 7' and 8 are essentially swapping the order

[2]See Definition 4 in [Can01], where environments play also the role of adversaries.

of quantifiers, from $\forall A[\ ]\ \exists B[\ ]$ to $\exists D[\ ]\ \forall A'[\ ]$, respectively[3]. It follows the natural question of under which conditions the two definitions are the same.

One direction is obvious[4]. The implication from definition 7' to definition 8 is not immediate. At its core lies what black-box simulatability is, which is the most-known form of simulatability so far in the literature.

For concreteness, we mention two ways to get around this implication: We could in the first place prove our result of full abstraction of UC in ppc, with respect to both definition 7' and 8 separately, and then use the fact that in the UC framework the corresponding two definitions are equivalent. But deriving the equivalence of the two definitions of security in ppc is not our goal. We aim at showing there is a correspondence between security in the UC framework and security in a non-machine-based framework as ppc.

An alternative is to directly use the result in [DKM$^+$04, DKMR04], where it is proved definitions 7' and 8 are equivalent for any process calculus whose notion of process-equivalence satisfies certain axioms, and apply it in particular to ppc.

In the remaining of this document we will make use of definition 8. Our choice is motivated by the belief this definition is technically simpler and intuitively well justified, although arguably less pedagogical.

Finally, it is worth noticing the compositional-security guarantee of ppc in terms of definition 8 also holds (See section 5.6 in Appendix).

4 Encoding and Main Result

We define an encoding that translates the ITM's of UC into processes in ppc. Since we are interested in transporting the interactive setting of UC, for an encoding to be "reasonable" we require it to preserve: machine signatures, interleaving mode, activation orders, and probability distributions over linking tapes.

Intuitively, a machine's encoding consists of a choice of (the encoding of) its interactive rules wrapped up by a bigger process reminiscent of the machine's shell.

A salient disctinction, however, is that while the premise of a rule is a predicate for a machine, the same premise is actually a prefix of inputs on channels for the machine's encoding. This poses a challenge when encoding, in the case there are two *intersecting* rules, i.e., there is a tape in the premises of two rules from where they expect different values.

One solution is to group intersecting rules into a single one by prefixing with the common input channel (tape) and then branching according to the

[3]To see why, develop in both definitions the meaning of observational equivalence and use the fact that $C[\ ] = C'[A'[\ ]]$ in definition 8.

[4]Simply take $B[\ ] = A[D[\ ]]$.

values they expect.

We observe the only difference between encodings of intersecting and non-intersecting rules is their input prefixes. But this does not affect the interactive pattern of the machine under encoding.

For simplicity, our definition of encoding does not consider intersecting rules, and we postpone encoding non-intersecting rules to section 5.2 in Appendix.

Definition 9 *Let $n \in \mathbb{N}$ and $k \in \mathbb{N}$ be a security parameter. Let $\mathcal{E} \in \{P_i, \mathcal{F}\}$, $i = 1, \ldots, n$, having l pairs of linking tapes*[5]*, and m non-intersecting interactive rules of form*

$$\frac{?_{rtape_{s_1,\ldots,s_{R_i}}}\ v_{s_1}, \ldots, v_{s_{R_i}}}{\begin{array}{l}(\vec{v}'_t, \varsigma') \leftarrow \mathcal{E}(\vec{v}_s, \varsigma)\\ \forall_{t1,\ldots,t_{W_i}} !_{wtape_{t.}}\ v'_{t.}\\ !_{activate}\ \mathcal{E}_a\\ \textit{waiting}\end{array}},$$

where $s_1, \ldots, s_{R_i}, t_1, \ldots, t_{W_i} \in \{1, \ldots, l\}$, and $a \in \{t_1, \ldots, t_{W_i}\}$. The encoding $[\![\mathcal{E}]\!]$ is defined as:

$$\begin{array}{rcll}[\![\mathcal{E}(k)]\!] & = & !_{q(k)}(\ )_{unblock_{\mathcal{E}}}\cdot(\ )_{activate}\cdot(& \langle\ \rangle_{isToDeliver}\\ & & & \mid (b)_{deliverySet}\cdot(\\ & & & \quad [b = empty].(\langle\ \rangle_{getState_{\mathcal{E}}} \mid Rules)\\ & & & \mid [b = notempty].\langle\ \rangle_{deliver}))\\ & & \mid !_{q(k)} Store_{\mathcal{E}} & \end{array}$$

with $Rules = (\varsigma)_{currentState_{\mathcal{E}}}\cdot(Rule_1 + \cdots + Rule_m)$, and

$$\begin{array}{rcl} Rule_i & = & (v_{s_1})_{rtape_{s_1}}\cdot\cdots\cdot(v_{s_{R_i}})_{rtape_{s_{R_i}}}\cdot\\ & & \Big(\langle proj_1(\mathcal{E})(\vec{v}_s,\varsigma)\rangle_{wtape_{t_1}} \mid \cdots \mid \langle proj_{W_i}(\mathcal{E})(\vec{v}_s,\varsigma)\rangle_{wtape_{t_{W_i}}} \mid\\ & & \langle proj_{W_i+1}(\mathcal{E})(\vec{v}_s.,\varsigma)\rangle_{updateStore_{\mathcal{E}}}\cdot\langle\ \rangle_{unblock_{\mathcal{E}}}\cdot\langle\mathcal{E}_a\rangle_{activate}\Big)\end{array}$$

where $proj_j(\mathcal{E})$ is the j-th projection of $\mathcal{E}$, $j = 1, 2, \ldots, W_i$.
Subprocess $Store_{\mathcal{E}}$ encodes $\mathcal{E}$'s state. Channels $(\ )_{rtape_{s.}}$, $\langle\ \rangle_{wtape_{t.}}$ are public and encode $\mathcal{E}$'s linking tapes; channel $(\ )_{activate}$ is private to the system and encodes its activation tape. All other channels are private to $\mathcal{E}$ but accessible to subprocess $Store_{\mathcal{E}}$.
We define the encoding of n ITM's $\mathcal{E}_1, \ldots, \mathcal{E}_n$, as $[\![\mathcal{E}_1, \ldots, \mathcal{E}_n]\!] \overset{def}{=} [\![\mathcal{E}_1]\!] \mid \ldots \mid [\![\mathcal{E}_n]\!]$.

Since $PPT(k)$ and *Term* are in one-to-one correspondence, we use the same expression for denoting functions $proj_j(\mathcal{E})$ and its terms.

Even though a key element of our encoding is a heavy use of updatable storage cells in order to model machines' states, the definition of $Store_{\mathcal{E}}$ is left out from this document. Encoding states does not require more than

[5]For an n-party protocol, $l_{P_i} = 3$ and $l_{\mathcal{F}} = n + 1$.

what is known in the literature for encoding data structures. On the other hand, this is a laborious effort that does not add clarity to our discourse. Nevertheless, we can informally specify what $\mathit{Store}_{\mathcal{E}}$ does and we observe its tasks are exclusively performed through private channels.

Several remarks justifying $\mathcal{E}$'s encoding are in place: First, the replication operator ensures the encoding is polynomial-time in the same security parameter k. Also, the reduction of terms $\mathit{proj}_j(\mathcal{E})$ on arguments $(v_{s_1}, \ldots, v_{s_{W_i}}, \varsigma)$ has the same distribution as $\mathcal{E}$'s computation on those arguments[6]. Finally, $\mathcal{E}$'s linking tapes with the environment are the only public channels, on which data-exchange is to be observed.

We recall a feature of ppc is that private data-exchange has higher priority than the public one. This is exploited to enforce some computations in sequence without altering the probability distributions of public events.

Process $E = [\![\mathcal{E}]\!]$ computes by activations turns, where there is one copy of its replicator per activation. Observe all copies of E are blocked. Each activation turn finishes with two actions: First, it releases a new copy of itself through private channel $\langle\ \rangle_{\mathit{unblock}_{\mathcal{E}}}$ and second, it activates another process E_a through public activation channel $\langle \mathcal{E}_a \rangle_{\mathit{activate}}$. This guarantees that only one copy of a process is available per activation.

When E is activated, it first asks through $\langle\ \rangle_{\mathit{isToDeliver}}$ whether there is any data to deliver. Subprocess $\mathit{Store}_{\mathcal{E}}$ answers through $(\ )_{\mathit{deliverySet}}$, based on a set containing the identities of processes for which there is data to deliver. If this set is not empty, then $\mathit{Store}_{\mathcal{E}}$ chooses a new process and activates it through its activation channel, and also releases a new copy of E. If the delivery-set is empty, $\mathit{Store}_{\mathcal{E}}$ is then asked to send E's current state and next one of the encoded interactive rules will be used.

Process *Rules* receives first E's state and then, in virtue of Proposition 1, reduces through the only rule whose input prefix is valid. Let Rule_i be this rule. It then computes the function $\mathcal{E}(v_{s_1}, \ldots, v_{s_{R_i}}, \varsigma)$, by using the projection functions proj_j, $j = 1, \ldots, W_i$, in order to write each of the resulting values $(v'_{t_1}, \ldots, v'_{t_{W_i}})$ in the corresponding public channels for processes $E_{t.}$, as well as writing its next state ς' to $\mathit{Store}_{\mathcal{E}}$.

Notice that writing $\langle v_{t.} \rangle_{\mathit{wtape}_{t.}}$ does not activate processes $E_{t.}$. First, $\mathit{Store}_{\mathcal{E}}$ constructs the delivery set with the identities of processes E_j to be activated in sequence, one at a time. Only then a new copy of E is released, exhausting the current one, and the activation is handed over to E_a.

Furthermore, at the end of an activation there are not dangling subprocesses of the current activated copy, since all possible simultaneous alternatives are exhausted either by matchings or input choices.

[6]Consider e.g. the PPT machine that on input $(\vec{v}, \varsigma)$ computes $(\vec{v}', \varsigma')$ like $\mathcal{E}$, but then discharges say ς'.

$$
\begin{array}{lcl}
[\![\mathcal{Z}(k,z)]\!] & = & !_{q(k)}(\ )_{unblock_{\mathcal{Z}}}.(\ )_{activate}.(\\
 & & \qquad \langle\ \rangle_{corrupt}|(\ )_{corruptdone}.(\\
 & & \qquad\qquad \langle\ \rangle_{guess}|(\ )_{noguess}.(\\
 & & \qquad\qquad\qquad \langle\ \rangle_{isToDeliver}|(b)_{deliverySet}.(\\
 & & \qquad\qquad\qquad\qquad [b = empty].(\langle\ \rangle_{getState_{\mathcal{Z}}}|\ Rules)\\
 & & \qquad\qquad\qquad\quad\ \ |\ [b = notempty].\langle\ \rangle_{deliver}))))\\
 & & |\ !_{q(k)}Store_{\mathcal{Z}}\\
 & & |\ !_{q(k)}Corrupt\\
 & & |\ !_{q(k)}Guess
\end{array}
$$

with $Rules = (\varsigma)_{currentState_{\mathcal{Z}}}.((\ )_{beg\_rnd}.Rule\_BgnRnd + (\ )_{end\_rnd}.Rule\_EndRnd)$

Figure 1: Definition of Process Environment

4.1 Encoding Environments

Crucially, we can also encode both interfaces and environments, hence adversarial behaviours. Here we examine only the case of environments, but observe that the other case uses similar ideas.

The translation is somewhat different to that in definition 9, since an environment can corrupt a party at any moment or decide its guess on the execution and halt. Concretely, by the definition of an environment ITM in Fig. 2, its encoding has rules *Rule_BegRnd*, *Rule_EndRnd*, modeling $\mathcal{Z}$'s behaviour when receiving messages or outputs from parties, respectively. In contrast, both the corruption case and the guessing case are not just rules, but replicated subprocesses *Corrupt* and *Guess*, which are at the same level of $\mathcal{Z}$'s state. Figure 1 formally defines an environment.

We describe only *Rule_BegRnd*, but observe that *Rule_EndRnd* is similar. This rule is valid when a new round of computation is to begin and this happens when all honest parties have sent their previous-round outputs:

$$
\begin{array}{l}
Rule\_BegRnd = \langle P_1\rangle_{inHonest}|(a)_{isHonest}\,([a = no].0\ |\\
\qquad\qquad [a = yes].(v_1)_{msbByP_1}.\langle v_1\rangle_{fwdmsgByP_1}.\\
\qquad\qquad\qquad (\langle P_2\rangle_{inHonest}\ldots([\![(\cdot) \leftarrow \mathcal{Z}]\!])\ldots))
\end{array}
$$

We have written $[\![(\cdot) \leftarrow \mathcal{Z}]\!]$ to abbreviate $\mathcal{Z}$'s actions in that point of the rule. It consists in sending in parallel the new-round messages and inputs for each honest party through corresponding channels $\langle\ \rangle_{msgToP_i}$ and $\langle\ \rangle_{inToP_i}$, by using the projections functions of $\mathcal{Z}$.

This rule simply reads party P_i's channel as long as this party is honest, and if so, forwards the value through a private channel to its inner-most process $[\![(\cdot) \leftarrow \mathcal{Z}]\!]$.

When process *Corrupt* is called, it reduces just like defined in Fig. 2, by modifying $\mathcal{Z}$'s state. Given the current state, this process may corrupt up to t parties, one at a time. It selects a party P_j to corrupt and activates P_j's store, $Store_{P_j}$, through public channel $\langle\ \rangle_{corrToP_j}$. It then receives P_j's state back through public channel $(\ )_{corrByP_j}$, and updates $\mathcal{Z}$'s store, $Store_{\mathcal{Z}}$, which also updates sets $C := C \cup \{j\}, H := H \setminus \{j\}$ of corrupt and honest parties, respectively. From this moment on, party P_j will not be activated any more, since in $[\![(\cdot) \leftarrow \mathcal{Z}]\!]$ above, new-round messages and inputs are computed and sent only to honest parties.

Encoding environments gives rise to a prominent set of processes in ppc:

Definition 10 *Let $n \in \mathbb{N}$. We let $\mathbf{Z}_{\mathtt{UC}}(n, k, z)$ be the image under encoding of the set of all UC environments modeling threshold, active and adaptive adversaries with synchronous and authenticated communication, that interact in n-party protocols with variable security parameter $k \in \mathbb{N}$ and variable auxiliary input $z \in \{0, 1\}^*$. i.e., $\mathbf{Z}_{\mathtt{UC}}(n, k, z) \overset{def}{=} \{[\![\mathcal{Z}(k, z)]\!] : \mathcal{Z} \in UC\}$.*

We intend to use these processes as distinguishers in ppc. In order to do so, we restrict observational equivalence to the class of contexts of form $Z|[\ ]$, where $Z \in \mathbf{Z}_{\mathtt{UC}}(n, k, z)$. More precisely,

Definition 11 *A closed process Q_1 is* UC-observationally equivalent *to another Q_2, written $Q_1 \simeq_{\mathtt{UC}} Q_2$, iff for all schedulers and for all $Z \in \mathbf{Z}_{\mathtt{UC}}(n, k, z)$ it holds that $Tr(Z|[Q_1]) \overset{c}{\approx} Tr(Z|[Q_2])$.*

Consequently, we redefine the notion of secure processes:

Definition 12 *A process Q* UC-emulates *an ideal specification process I, written $Q \equiv_{\mathtt{UC}} I$, if there exists a context $D[\ ]$ such that $Q \simeq_{\mathtt{UC}} D[I]$.*

An important observation is that if Q_1, Q_2 are processes whose public data-exchange is exclusively with UC contexts[7], then the quantification over schedulers is virtually useless for the overall distribution, or in other words, distinguishing two processes is independent on any external scheduler. This is so because UC contexts (probabilistically) decide which process to send data to (activate) next, and choose one at a time. Hence, the only kind of simultaneous data-exchange are private, but these are silent actions, and therefore should not affect indistinguishability of processes. This reasoning is a direct consequence of the following:

Lemma 1 *Let $n \in \mathbb{N}$. Let Q be a closed process whose public channels are contained among the public channels of $\mathbf{Z}_{\mathtt{UC}}(n, \cdot, \cdot)$. Then, for any scheduler the transition probability on any public data-exchange is equal to one.*

A second implication of interacting with UC contexts is that the probability distribution of values transmitted through public channels is preserved up to a factor dependable exclusively on the probability distribution for private data-exchange.

[7]This is precisely the case of the encodings of machines P_i, $\mathcal{F}$, and $\mathcal{T}$.

Lemma 2 *Let $n \in \mathbb{N}$. Let $\mathcal{Z}, \mathcal{E}$ be linked, and let $\mathcal{E}$ compute $v' \leftarrow \mathcal{E}(v, \varsigma)$, on value v and current state ς, and send v' to $\mathcal{Z}$ through $tape_{\mathcal{E},\mathcal{Z}}$. Then, $p_{v'} = P(v' \leftarrow \mathcal{E}(v, \varsigma))$ iff $P(\langle tape_{\mathcal{E},\mathcal{Z}}, v' \rangle \in Tr(Z|Q)^S_k) = p_{v'} \cdot u_S$, where Q has $E = [\![\mathcal{E}]\!]$ as a subprocess.*
When private data-exchange is sequentialized, then u_S is always one.

Our last observation is that UC-equivalence is weaker than that in definition 6, i.e., $Q_1 \simeq_{\mathtt{UC}} Q_2$ does *not* necessarily imply $Q_1 \simeq Q_2$, or colloquially, quantification over the class of UC-adversaries does *not* necessarily imply quantification over all contexts.

This is an expected effect, since the environments we are considering are constrained to the assumptions of the adversarial and communication models, e.g., activation of all parties once per round, message-delivery guaranteed, authenticated channels, threshold corruptions, etc. Nonetheless, this definition of security is not trivial at all, since useful realizations can be proved secure under such assumptions. Hence, by defining equivalence of processes in ppc restricted to the contexts induced by UC environments, we are not over-simplifying the notion of security in ppc.

4.2 Main Result

We can now prove a correspondence between the UC framework and the process calculus ppc under UC-observational equivalence:

Theorem 1 (Full Abstraction) *Let $n \in \mathbb{N}$ and $t < n$. Let π be a n-party reactive protocol, $\mathcal{F}$ be an ideal functionality, and $\mathcal{T}$ be an interface.*

1. *(Soundness) If $[\![\pi]\!] \equiv_{\mathtt{UC}} [\![\mathcal{T}]\!] \mid [\![\mathcal{F}]\!]$ then $\mathcal{T}$ proves π t-realizes $\mathcal{F}$; and reciprocally,*

2. *(Completeness) If $\mathcal{T}$ proves π t-realizes $\mathcal{F}$, then $[\![\pi]\!] \equiv_{\mathtt{UC}} [\![\mathcal{T}]\!] \mid [\![\mathcal{F}]\!]$.*

Proof:

1. If it was not the case, there would exist an environment $\mathcal{Z}$ negating indistinguishability in UC. But this would refute the validity of $[\![\mathcal{T}]\!]|[\ ]$ in the hypothesis, since the context $[\![\mathcal{Z}]\!]|[\ ]$ induced by $\mathcal{Z}$ would also negate indistinguishability in ppc. Indeed, by lemma 2, both Z and $\mathcal{Z}$ would observe $\langle guess, b\rangle$ with comparable probabilities, and this would be sufficient, according to lemma 1 to call for distinguishability in ppc.

2. If $\mathcal{T}$ is an interface proving π t-realizes $\mathcal{F}$, then it should hold that $[\![\pi]\!] \simeq_{\mathtt{UC}} [\![\mathcal{T}]\!] | [[\![\mathcal{F}]\!]]$, since otherwise, there would exist according to lemma 1 $[\![\mathcal{Z}]\!]|[\ ] \in \mathbf{Z}_{\mathtt{UC}}$ negating indistinguishability in ppc, and consequently inferring that $\mathcal{Z}$ would negate indistinguishability in UC. Indeed, if Z

observes $\langle public\_tape, v\rangle$ with overwhelming distinct probabilities, then by lemma 2 $\mathcal{Z}$ does so too and at that point can *halt* and output the guessing bit b. $\diamondsuit$

References

[BPW04] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *LNCS*, volume 2951, pages 336–354, 2004. TCC'04.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, page 136, 2001.

[DKM$^+$04] A. Datta, R. K usters, J. Mitchell, A. Ramanathan, and V. Shmatikov. Unifying equivalence-based definitions of protocol security. In *Workshop on Issues in the Theory of Security*, 2004. WITS'04 Satellite Workshop at ETAPS'04.

[DKMR04] A. Datta, R. K usters, J. Mitchell, and A. Ramanathan. Sequential probabilistic process calculus and simulation-based security. Technical report, Stanford University and Christian-Albrechts-Universität zu Kiel, 2004. TR-SPPC-2004.

[DKMR05] A. Datta, R. K usters, J. Mitchell, and A. Ramanathan. On the relationships between notions of simulation-based security. In *Proceedings of Theory of Cryptography Conference (to appear)*, 2005.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *LNCS*, volume 2729, pages 247–264, 2003. CRYPTO'03.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[Gol01] O. Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, 2001.

[LMMS98] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 112–121, 1998.

[LMMS99] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security analysis. In *World Congress on Formal Methods*, pages 776–793, 1999.

[MMS98] J. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 725–733, 1998. FOCS'98.

[MMS03] P. Mateus, J. Mitchell, and A. Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time process calculus. In *LNCS*, volume 2761, pages 327–349, 2003. CONCUR 2003.

[Nie04] Jesper Buus Nielsen. *On Protocol Security in the Cryptographic Model*. PhD thesis, BRICS, University of Aarhus, Department of Computer Science, IT-parken, Aabogade 34, DK-8200 Århus N, Denmark, 2004.

[PW00] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000.

5 Appendix:

5.1 $\varnothing$-hybrid Model and Ideal Process

Figures 2 and 3 define the $\varnothing$-hybrid model and the ideal process, respectively. The code of each rule should be self-explanatory. The piece of code under the horizontal line is to be executed indivisibly, as a single transition.

However, we allow nested rules, hence, we assume resumption: If there is a point where the activation is handed over to another machine, then once recovered the execution resumes at the point it was left before.

We could be more succinct to the expense of rigorousity in regard to machines' states. Our main purpose is to illustrate the interactive pattern of the UC model by means of our notation. In constrast, we have to prevent ourselves of this relaxation in a full translation from machines into processes. As an example, the first rule for $\mathcal{Z}$ reads: Upon activation in round r, check the condition whether all honest parties $i \in H$ have sent their round $r-1$'s outputs $y_{i,r}$. If it holds, then pick an honest party i, compute his round r's input $x_{i,r}$ and round $r-1$'s messages from corrupt parties $\{m_{j,i,r-1}\}_{j\in C}$. Observe messages $\{m_{j,i,r-1}\}_{j\in H}$ where sent by honest parties in the previous round. Write $x_{i,r}$ on tape inToP_i, and $(\mathit{bgn\_rnd}, i, \{m_{j,i,r-1}\}_{j\in C\cup H})$ on tape msgToP_i. Activate this party by writing P_i on tape *activate*, and switch to control state *waiting*.

This rule is to be executed as long as there are honest parties to be activated with command *beg_rnd*. To do so, we use a variable $\mathit{com}(r) \in \{\mathit{beg\_rnd}, \mathit{end\_rnd}\}$ that indicates which command is to be executed next. In addition, a set I contains the indices of remaining honest parties to be activated.

Note well that on every new activation of $\mathcal{Z}$, the set I, from where an honest party is picked, may have changed, since $\mathcal{Z}$ may decide first to corrupt a party before executing this rule.

5.2 Encoding of Interactive Rules

Figures 4 and 5 define the encodings of the interactive rules for a party P_i and a functionality $\mathcal{F}$, respectively.

5.3 Proof Proposition1

Proof: In UC framework deadlock is prevented because parties are linked only to the environment, and the environment is assumed not to activate two machines at a time. The remaining properties follow by checking the conditions of the interactive rules of each machine. $\diamondsuit$

5.4 Proof Lemma 1

PROOF: Let S be any scheduler over multisets $\mathcal{M}$ of subprocesses of $Z|Q$. Then, given that S is a Markov chain and since data-exchange over public channels occur one at a time, it holds that $S(\mathcal{M}_1, \mathcal{M}_2) = 1$, where

i. $\{\langle v\rangle_c, (x)_c.X\} \subset \mathcal{M}_1$ and there are no other probable transitions in $\mathcal{M}_1$;

ii. $\mathcal{M}_2 = (\mathcal{M}_1 \setminus \{\langle v\rangle_c, (x)_c.X\}) \cup \mathcal{M}_{X_v^x}$ and X_v^x is the process obtained by substituting all free occurrences of x by v in X; and

iii. $c \in \{\mathit{inToP}_i, \mathit{outByP}_i, \mathit{msgToP}_i, \mathit{msgByP}_i, \mathit{corrToP}_i, \mathit{corrByP}_i\}$. ♢

5.5 Proof Lemma 2

PROOF:(Sketch) The proof proceeds by induction on the number of activations in the execution. The key fact is that machines and processes are scheduled in the same order by $\mathcal{Z}$ and its encoding Z. Thus, when machine $\mathcal{E}$ is activated with input value v, its corresponding process E is activated with the same input value and having, by induction hypothesis, the same state ς as $\mathcal{E}$. Then, in virtue of our encoding, process E would initiate a tree of reduction involving solely private data-exchange in order to update its state. Assume the chosen branch in this tree has probability u_S. The branch ends at the point value v' is computed by sending $\langle \mathcal{E}(v, \varsigma)\rangle_{\mathit{tape}_{\mathcal{E},\mathcal{Z}}}$, with probability $p_{v'}$. Finally, the sent value is actually observed with probability one, as it is the unique public data-exchange at this point of the computation (reduction). Therefore, $P(\langle \mathit{tape}_{\mathcal{E},\mathcal{Z}}, v'\rangle \in Tr(Z|Q)_k^S) = p_{v'} \cdot u_S \cdot 1$. ♢

5.6 ppc-Composition Theorem

We provide a prove of the compositional theorem in ppc with respect to definition 8. In what follows we use $\equiv$ to stand for protocol emulation.

Proposition 2 (Composition) *Let $Q \equiv I$ and $R[I] \equiv J$. Then $R[Q] \equiv J$.*

PROOF: The hypotheses can be rephrased as:

$$\exists\ D[\ ] \quad Q \simeq D[I] \tag{1}$$

$$\exists\ E[\ ] \quad R[I] \simeq E[J] \quad , \tag{2}$$

where $\simeq$ means observational equivalence (cf. Definition 6). We will prove that $D[E[\ ]]$ is a context such that $R[Q] \simeq D[E[J]]$, i.e., for all schedulers and for all contexts $C[\ ]$ it holds that

$$Tr(C[R[Q]]) \overset{c}{\approx} Tr(C[D[E[J]]]) \ .$$

Indeed, since $\overset{c}{\approx}$ is transitive, it follows that

$$
\begin{array}{ll}
 & Tr(C[R[Q]]) \\
\overset{c}{\approx} & \qquad\qquad (\ (1) \text{ with } C[R[\ ]]\ ) \\
 & Tr(C[R[D[I]]]) \\
= & \\
 & Tr(C[D[R[I]]]) \\
\overset{c}{\approx} & \qquad\qquad (\ (2) \text{ with } C[D[\ ]]\ ) \\
 & Tr(C[D[E[J]]])
\end{array}
$$

The second step above holds as long as contexts $R[\ ]$ and $D[\ ]$ are of form $X|[\ ]$ or $(\ )c.[\ ]$ and there is no clash of free-names. We claim, albeit without further evidence here, this kind of composition is sufficient for modelling the interaction of ITM's where linking tapes do not stand for actual communication devices. $\diamondsuit$

Init: $\mathcal{Z}(k, z, \mathfrak{r}_{\mathcal{Z}})$, $P_i(k, \mathfrak{r}_i)$. Set $r := 0$, $C := \{\ \}$, $H \stackrel{\text{def}}{=} [n] - C$, $I := H$.

Environment $\mathcal{Z}$**:**

```
$\forall_{i \in H}\ ?_{outByP_i}\ y_{i,r-1}$
--------------------------------------------
precondition: $comand(r) = beg\_rnd$
precondition: $\{I \supseteq H$ –1st call: =–$\}$
$I := I \setminus C$
$i_{\in I} \leftarrow \mathcal{Z}$
$I := I \setminus \{i\}$
if $I = \{\ \}$ then
    $com(r) := end\_rnd$
    $I := H$
$(\{m_{j,i,r-1}\}_{j \in C}, x_{i,r}) \leftarrow \mathcal{Z}$
$!_{inToP_i}\ (i, x_{i,r})$
$!_{msgToP_i}\ (beg\_rnd, i, \{m_{j,i,r-1}\}_{j \in C \cup H})$
$!_{activate}\ P_i$
waiting


$\forall_{i \in H}\ ?_{msgByP_i}\ \{m_{i,j,r}\}_{j \neq i}$
--------------------------------------------
precondition: $comand(r) = end\_rnd$
precondition: $\{I \supseteq H$ –1st call: =–$\}$
$I := I \setminus C$
$i_{\in I} \leftarrow \mathcal{Z}$
$I := I \setminus \{i\}$
if $I = \{\ \}$ then
    $r := r + 1$
    $com(r) := beg\_rnd$
    $I := H$
$!_{msgToP_i}\ (end\_rnd)$
$!_{activate}\ P_i$
waiting


--------------------------------------------
if $|C| < t$ then
    $i_{\in H} \leftarrow \mathcal{Z}$
    $C := C \cup \{i\}$
    $I := I \setminus C$
    if $I = \{\ \}$ then
        $com(r) := 1 - com(r)$
        $I := H$
    $!_{corrToP_i}\ (corrupt\ i)$
    $!_{activate}\ P_i$
waiting


--------------------------------------------
$b \leftarrow \mathcal{Z}$
$!_{\mathsf{guess}}\ b$
halt
```

Party P_i**:**

```
$?_{inToP_i}\ (i, x_{i,r})$
$?_{msgToP_i}\ (beg\_rnd, i, \{m_{j,i,r-1}\}_{j \in C \cup H})$
--------------------------------------------
    $(\{m_{i,j,r}\}, y_{i,r}) = P_i(\{m_{j,i,r-1}\}, x_{i,r})$
    $!_{msgByP_i}\ \{m_{i,j,r}\}_{j \neq i}$
    waiting


$?_{msgToP_i}\ (end\_rnd)$
--------------------------------------------
    $!_{outByP_i}\ y_{i,r}$
    $!_{activate}\ \mathcal{Z}$
    waiting


$?_{corrToP_i}\ (corrupt\ i)$
--------------------------------------------
    $!_{corrByP_i}\ \mathfrak{r}_i$
    $!_{activate}\ \mathcal{Z}$
    waiting
```

Figure 2: Order of activations in $\varnothing$-hybrid model

Init: $\mathcal{Z}(k, z, \mathfrak{r}_{\mathcal{Z}})$, $\mathcal{T}(k, \mathfrak{r}_{\mathcal{T}})$, $\mathcal{F}(k, \mathfrak{r}_{\mathcal{F}})$. Set $r := 0$, $C := \{\ \}$, $H \overset{\text{def}}{=} [n] - C$, $I_{\mathcal{T}} := H$, $I_{\mathcal{F}} := H$.

Interface $\mathcal{T}$:

```
$?_{msgToP_i}\ (beg\_rnd, i, \{m_{j,i,r-1}\}_{j \in C \cup H})$
----------------------------------------
precondition:$\{I_{\mathcal{T}} \supseteq H$ –1st call: =–$\}$
$I_{\mathcal{T}} := I_{\mathcal{T}} \setminus \{i\}$
$!_{activate}\ \mathcal{F}$
    $?_{leaked}\ v^{inleak}_{\mathcal{F},i}$
    ------------------------------------
    $(\{m_{i,j,r}\}_{j \in [n]})$
    $\leftarrow \mathcal{T}(k, \{m_{j,i,r-1}\}_{j \in C, i \in H}, v^{inleak}_{\mathcal{F},i}; \mathfrak{r}_{\mathcal{T}})$
    if $I_{\mathcal{T}} = \{\ \}$ then
        $!_{probe}\ (beg\_rnd, v')$
        $!_{activate}\ \mathcal{F}$
            $true$
        ------------
        $I_{\mathcal{T}} := H$
    $!_{msgByP_i}\ \{m_{i,j,r}\}_{j \neq i}$
    $!_{activate}\ \mathcal{Z}$
    $waiting$
```

```
$?_{msgToP_i}\ (end\_rnd)$
------------------------
$!_{probe}\ (end\_rnd, i)$
$!_{activate}\ \mathcal{F}$
    $true$
------------
$!_{activate}\ \mathcal{Z}$
$waiting$
```

```
$?_{corrToP_i}\ (corrupt\ i)$
------------------------
$I_{\mathcal{T}} := I_{\mathcal{T}} \setminus \{i\}$
if $I_{\mathcal{T}} = \{\ \}$ then
    $!_{probe}\ v'$
    $!_{activate}\ \mathcal{F}$
        $true$
    ------------
    $I_{\mathcal{T}} := H$
$!_{probe}\ (corrupt\ i)$
$!_{activate}\ \mathcal{F}$
    $?_{leaked}\ v^{corrleak}_{\mathcal{F},i}$
    ------------------------
    $\mathfrak{r}_i \leftarrow \mathcal{T}$
    $!_{corrByP_i}\ \mathfrak{r}_i$
    $!_{activate}\ \mathcal{Z}$
    $waiting$
```

Functionality $\mathcal{F}$:

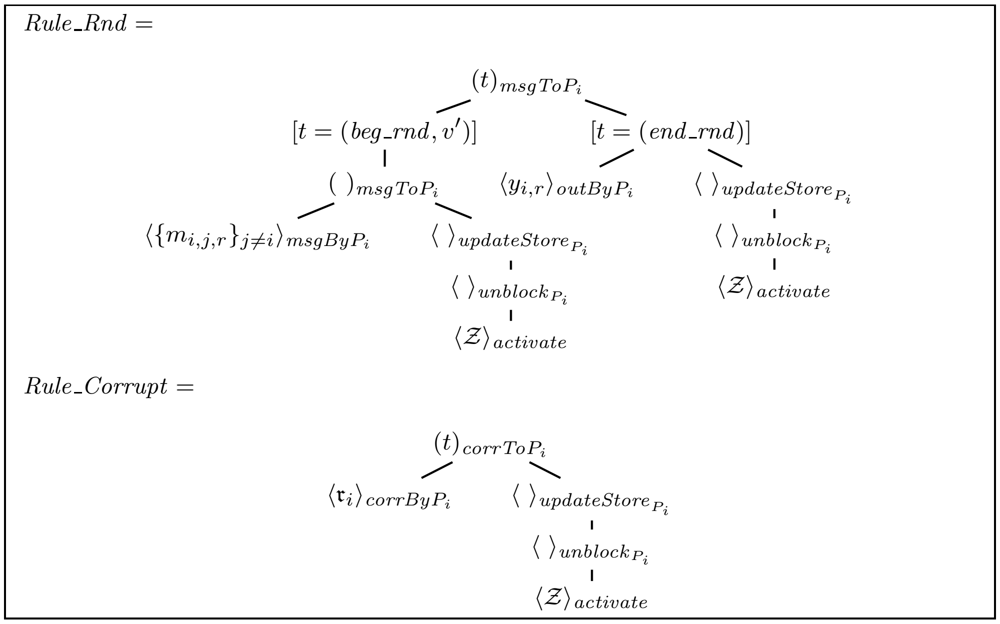
```
$?_{inToP_i}\ (i, x_{i,r})$
------------------------
$v^{inleak}_{\mathcal{F},i} \leftarrow \mathcal{F}(k, x_{i,r}; \mathfrak{r}_{\mathcal{F}})$
$!_{leaked}\ v^{inleak}_{\mathcal{F},i}$
$!_{activate}\ \mathcal{T}$
$waiting$
```

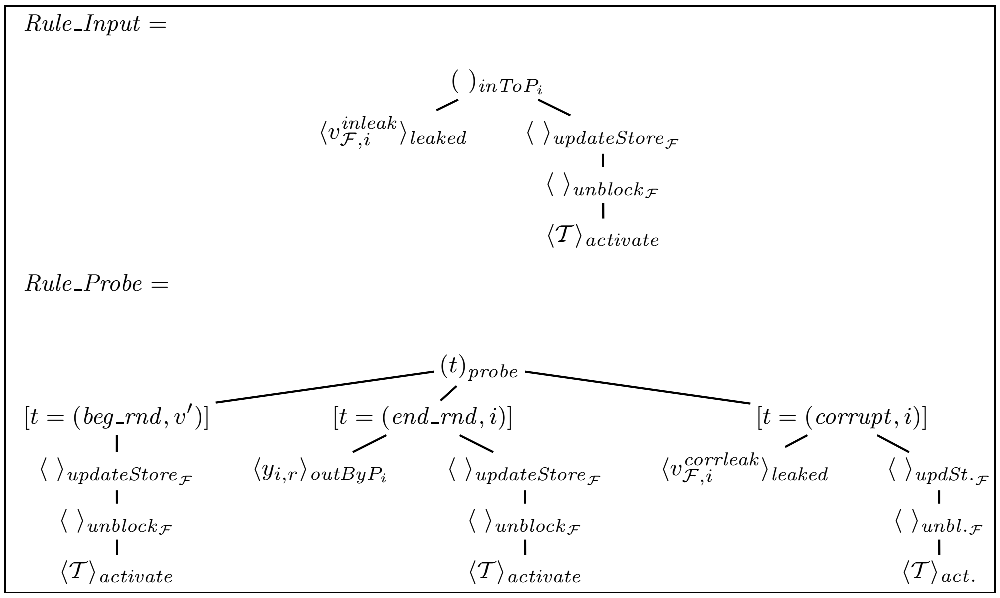
```
$?_{probe}\ (beg\_rnd, v')$
------------------------
$(\{y_{i,r}\}_{i \in H}, v^{outleak}_{\mathcal{F}})$
$\leftarrow \mathcal{F}(k, \{x_{i,r}\}_{i \in H}, v'; \mathfrak{r}_{\mathcal{F}})$
$!_{activate}\ \mathcal{T}$
$waiting$
```

```
$?_{probe}\ (end\_rnd, i)$
------------------------
precondition:$\{I_{\mathcal{F}} \supseteq H$ –1st call: =–$\}$
$I_{\mathcal{F}} := I_{\mathcal{F}} \setminus \{i\}$
$!_{outByP_i}\ y_{i,r}$
if $I_{\mathcal{F}} = \{\ \}$ then
    $!_{leaked}\ v^{outleak}_{\mathcal{F}}$
    $I_{\mathcal{F}} := H$
$!_{activate}\ \mathcal{T}$
$waiting$
```

```
$?_{probe}\ (corrupt\ i)$
------------------------
$I_{\mathcal{F}} := I_{\mathcal{F}} \setminus \{i\}$
if $I_{\mathcal{F}} = \{\ \}$ then
    $!_{leaked}\ v^{outleak}_{\mathcal{F}}$
    $I_{\mathcal{F}} := H$
$v^{corrleak}_{\mathcal{F},i} \leftarrow \mathcal{F}(k, i; \mathfrak{r}_{\mathcal{F}})$
$!_{leaked}\ v^{corrleak}_{\mathcal{F},i}$
$!_{activate}\ \mathcal{T}$
$waiting$
```

Figure 3: Order of activations in ideal process


$Rule\_Rnd =$
$(t)_{msgToP_i}$
$[t = (beg\_rnd, v')]$
$[t = (end\_rnd)]$
$(\ )_{msgToP_i}$
$\langle y_{i,r} \rangle_{outByP_i}$
$\langle\ \rangle_{updateStore_{P_i}}$
$\langle \{m_{i,j,r}\}_{j \neq i} \rangle_{msgByP_i}$
$\langle\ \rangle_{updateStore_{P_i}}$
$\langle\ \rangle_{unblock_{P_i}}$
$\langle\ \rangle_{unblock_{P_i}}$
$\langle \mathcal{Z} \rangle_{activate}$
$\langle \mathcal{Z} \rangle_{activate}$
$Rule\_Corrupt =$
$(t)_{corrToP_i}$
$\langle \mathfrak{r}_i \rangle_{corrByP_i}$
$\langle\ \rangle_{updateStore_{P_i}}$
$\langle\ \rangle_{unblock_{P_i}}$
$\langle \mathcal{Z} \rangle_{activate}$
Figure 4: Enconding of P_i's Interactive Rules


$Rule\_Input =$
$(\ )_{inToP_i}$
$\langle v^{inleak}_{\mathcal{F},i} \rangle_{leaked}$
$\langle\ \rangle_{updateStore_{\mathcal{F}}}$
$\langle\ \rangle_{unblock_{\mathcal{F}}}$
$\langle \mathcal{T} \rangle_{activate}$
$Rule\_Probe =$
$(t)_{probe}$
$[t = (beg\_rnd, v')]$
$[t = (end\_rnd, i)]$
$[t = (corrupt, i)]$
$\langle\ \rangle_{updateStore_{\mathcal{F}}}$
$\langle y_{i,r} \rangle_{outByP_i}$
$\langle\ \rangle_{updateStore_{\mathcal{F}}}$
$\langle v^{corrleak}_{\mathcal{F},i} \rangle_{leaked}$
$\langle\ \rangle_{updSt._{\mathcal{F}}}$
$\langle\ \rangle_{unblock_{\mathcal{F}}}$
$\langle\ \rangle_{unblock_{\mathcal{F}}}$
$\langle\ \rangle_{unbl._{\mathcal{F}}}$
$\langle \mathcal{T} \rangle_{activate}$
$\langle \mathcal{T} \rangle_{activate}$
$\langle \mathcal{T} \rangle_{act.}$
Figure 5: Enconding of $\mathcal{F}$'s Interactive Rules