On the Relationships Between Notions of Simulation-Based Security*

Anupam Datta[1] Ralf Küsters[2] John C. Mitchell[1] Ajith Ramanathan[1]

[1] *Computer Science Department*
Stanford University
Stanford, CA 94305-9045
{danupam,jcm,ajith}@cs.stanford.edu

[2] *Institut für Informatik*
Christian-Albrechts-Universität zu Kiel
kuesters@ti.informatik.uni-kiel.de

Abstract

Several compositional forms of simulation-based security have been proposed in the literature, including universal composability, black-box simulatability, and variants thereof. These relations between a protocol and an ideal functionality are similar enough that they can be ordered from strongest to weakest according to the logical form of their definitions. However, determining whether two relations are in fact identical depends on some subtle features that have not been brought out in previous studies. We identify the position of a "master process" in the distributed system, and some limitations on transparent message forwarding within computational complexity bounds, as two main factors. Using a general computational framework, called Sequential Probabilistic Process Calculus (SPPC), we clarify the relationships between the simulation-based security conditions. We also prove general composition theorems in SPPC. Many of the proofs are carried out based on a small set of equivalence principles involving processes and distributed systems. This gives us results that carry over to a variety of computational models.

Keywords: simulation-based security, universal composability, reactive simulatability, black-box simulatability, process calculus

1 Introduction

Several current projects use ideal functionality and indistinguishability to state and prove compositional security properties of protocols and related mechanisms. The main projects include work

*An abridged version of this work has been published in TCC 2005 [15]. Our work was partially supported by the DoD University Research Initiative (URI) program administered by the Office of Naval Research under Grant N00014-01-1-0795, by OSD/ONR CIP/SW URI "Trustworthy Infrastructure, Mechanisms, and Experimentation for Diffuse Computing" through ONR Grant N00014-04-1-0725, by NSF CCR-0121403, Computational Logic Tools for Research and Education, and by NSF CyberTrust Grant 0430594, Collaborative research: High-fidelity methods for security protocols. Part of this work was carried out while the second author was at Stanford University supported by the "Deutsche Forschungsgemeinschaft (DFG)".

by Canetti and collaborators on an approach called *universal composability* [9, 11–14] and work by Backes, Pfitzmann, and Waidner on a related approach that also uses *black-box simulatability* [4, 5, 7, 27]. Other projects have used the notion of equivalence in process calculus [17, 23, 24], a well-established formal model of concurrent systems. While some process-calculus-based security studies [1–3] abstract away probability and computational complexity, at least one project [22, 25, 26, 28] has developed a probabilistic polynomial-time process calculus for security purposes. The common theme in each of these approaches is that the security of a real protocol is expressed by comparison with an ideal functionality or ideal protocol. However, there are two main differences between the various approaches: the precise relation between protocol and functionality that is required, and the computational modeling of the entities (protocol, adversary, simulator, and environment). All of the computational models use probabilistic polynomial-time processes, but the ways that processes are combined to model a distributed system vary. We identify two main ways that these computational models vary: one involving the way the next entity to execute is chosen, and the other involving the capacity and computational cost of communication. We then show exactly when the main security notions differ or coincide.

In [9], Canetti introduced universal composability (UC), based on probabilistic polynomial-time interacting Turing machines (PITMs). The UC relation involves a real protocol and ideal functionality to be compared, a real and ideal adversary, and an environment. The real protocol realizes the ideal functionality if, for every attack by a real adversary on the real protocol, there exists an attack by an ideal adversary on the ideal functionality, such that the observable behavior of the real protocol under attack is the same as the observable behavior of the ideal functionality under attack. Each set of observations is performed by the same environment. In other words, the system consisting of the environment, the real adversary, and the real protocol must be indistinguishable from the system consisting of the environment, the ideal adversary, and the ideal functionality. The scheduling of a system of processes (or ITMs) is *sequential* in that only one process is active at a time, completing its computation before another is activated. The default process to be activated, if none is designated by process communication, is the environment. In the present work, we use the term *master process* for the default process in a system that runs when no other process has been activated by explicit communication.

In [27], Pfitzmann and Waidner use a variant of UC and a notion of black-box simulatability (BB) based on probabilistic polynomial-time IO automata (PIOA). In the BB relation between a protocol and ideal functionality, the UC ideal adversary is replaced by the combination of the real adversary and a simulator that must be chosen independently of the real adversary. Communication and scheduling in the PIOA computational model are sequential as in the PITM model. While the environment is the master process in the PITM studies, the adversary is chosen to be the master process in the Pfitzmann-Waidner version of UC. In the Pfitzmann-Waidner version of BB the master process is the adversary or the simulator [27]. In a later version of the PIOA model (see, e.g., [4]), the environment is also allowed to serve as the master process, subject to the restriction that in any given system it is not possible to designate both the adversary/simulator and the environment as the master process. In proofs in cryptography, another variant of BB is often considered in which the simulator may depend on the real adversary or its complexity. We call this variant Weak BB (WBB) and the previous one Strong BB (SBB).

In [22, 26, 28, 29], Mitchell et al. have used a form of process equivalence, where an environment directly interacts with the real and ideal protocol. The computational model in this work is a probabilistic polynomial-time processes calculus (PPC) that allows concurrent (non-sequential)

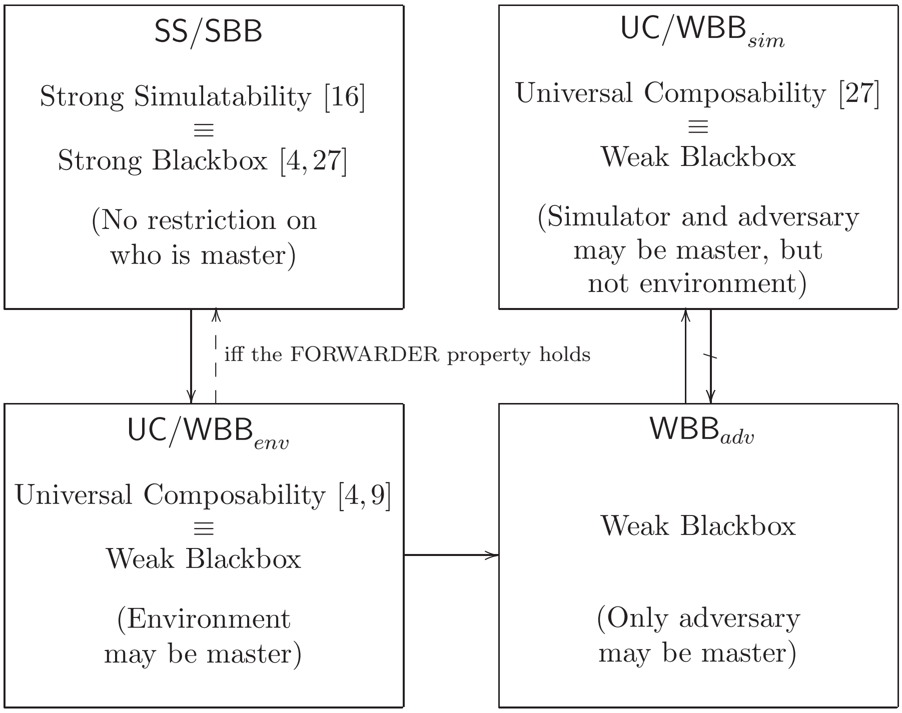

Figure 1: Equivalences and implications between the security notions in SPPC

execution of independent processes. The process equivalence relation gives rise to a relation between protocols and ideal functionalities by allowing a simulator to interact with the ideal functionality, resulting in a relation that we call strong simulatability, SS [16]. The difference between SS and SBB is that in SBB, the environment and the adversary are separated while the SS environment also serves as the adversary.

Contribution of the paper. In this paper, we clarify the relationships between UC, SBB, WBB, SS under different placements of the master process and an additional issue involving the ability to define a "forwarding" process that forwards communication from one process to another. While it seems intuitively reasonable that such a forwarder can be placed between two processes without changing the overall behavior of the system, this may violate complexity bounds if a polynomial-time forwarder must be chosen before the sending or receiving process. If the time bound of the sender, for example, exceeds the time bound of the forwarder, then some sent messages may be lost because the time bound of the forwarder has been exhausted. This is relevant to our study because some equivalence proofs require the existence of forwarders that cannot be exhausted.

Our main results are summarized in Figure 1. Each of the four boxes in this figure stands for a class of equivalent security notions. Specifically, if a real and ideal protocol are related by one notion in this class, then they are also related by all other notions in this class. A solid arrow from one class to another indicates that relations in the first class imply relations in the second class. The implication indicated by the dashed arrow is contingent on whether the aforementioned forwarding property holds for the processes in question.

The proofs of equivalence and implication between security notions are axiomatic, using a relatively small set of clearly stated equivalence principles involving processes and distributed systems. This approach gives us results that carry over to a variety of computational models. Our axiomatic system is proved sound for a specific computational model, a sequential probabilistic polynomial-time process calculus (SPPC), developed for the purpose of this study. SPPC is a sequential model, allowing only one process to run at a time. When one process completes, it sends an output indicating which process will run next. This calculus is close to PIOA and PITM in expressiveness and spirit, while (1) providing a syntax for writing equations between systems of communicating machines and (2) being flexible enough to capture different variants of security notions, including all variants of SS, SBB, WBB, and UC discussed in this paper. Our results about these security notions formulated over SPPC are:

1. Equivalences between security notions.
   (a) The different forms of Strong Simulatability and Strong Blackbox obtained by varying the entity that is the master process are all equivalent. This equivalence class, denoted $\mathsf{SS/SBB}$, is depicted in the top-left box in Figure 1 and includes placements of the master process as considered for Strong Blackbox in [4, 27]
   (b) All variants of Universal Composability and Weak Blackbox in which the environment may be the master process are equivalent. This equivalence class, denoted $\mathsf{UC/WBB}_{env}$, is depicted in the bottom-left box in Figure 1 and includes placements of the master process as considered for Universal Composability in [4, 9].
   (c) All variants of Universal Composability and Weak Blackbox in which the simulator and the adversary may be the master process, but not the environment are equivalent. This equivalence class, denoted $\mathsf{UC/WBB}_{sim}$, is depicted in the top-right box in Figure 1 and includes placements of the master process as considered for Universal Composability in [27].
   (d) All variants of Weak Blackbox where the adversary may be the master process, but neither the environment nor the simulator may play this role are equivalent. This equivalence class, denoted WBB_{adv}, is depicted in the bottom-right box in Figure 1.
2. Implications between the classes.
   (a) $\mathsf{SS/SBB}$ implies $\mathsf{UC/WBB}_{env}$. In particular, Strong Blackbox with placements of the master process as considered in [4, 27] implies Universal Composability with placements of the master process as considered in [4, 9].
   (b) $\mathsf{UC/WBB}_{env}$ implies WBB_{adv}.
   (c) WBB_{adv} implies $\mathsf{UC/WBB}_{sim}$. In particular, Strong Blackbox with placements of the master process as considered in [4, 27] and Universal Composability with placements of the master process as considered in [4, 9] implies Universal Composability with placements of the master process as considered in [27].
3. Separations between the classes.
   (a) The security notions in $\mathsf{UC/WBB}_{env}$ are strictly weaker than those in $\mathsf{SS/SBB}$ in any computational model where the forwarding property (expressed precisely by the FORWARDER axiom) fails. Since this property fails in the PITM model [9] and the buffered

PIOA model [4], it follows that $\mathsf{UC/WBB}_{env}$ does not imply $\mathsf{SS/SBB}$ in these models. This contradicts a theorem claimed in [4]. However, the forwarding property holds in SPPC and the buffer-free PIOA model for most protocols of interest. In these cases, $\mathsf{UC/WBB}_{env}$ implies $\mathsf{SS/SBB}$.

(b) The security notions in $\mathsf{UC/WBB}_{sim}$ are strictly weaker than the notions in WBB_{adv}, and hence, the notions in $\mathsf{UC/WBB}_{env}$ and $\mathsf{SS/SBB}$. In particular, the Universal Composability relation with placements of the master process as considered in [27] does neither imply the Strong Blackbox relations with placements of the master process as considered in [4, 27] nor Universal Composability relations with placements of the master process as considered in [4, 9].

These results all show that the relationship between universal composability and black-box simulatability is more subtle than previously described. One consequence is that when proving compositional security properties by a black-box reduction, care must be taken to make sure that the computational model gives appropriate power to the environment. In particular, the composability theorem of Canetti [9] does not imply that blackbox simulatability is a composable security notion, over any computational model in which the forwarding property (expressed by the FORWARDER axiom) is not satisfied.

Another contribution of this paper is a general composition theorem for SPPC that, similar to the composition theorem in Canetti's model, allows to compose a polynomial number of copies of protocols, while in SPPC the protocols may be combined in a very flexible way. In addition to demonstrating the suitability of SPPC as a formalism for simulation-based security, this result is interesting in its own right.

We note that the present work concentrates on models for simulation-based security where processes run in polynomial time in the security parameter alone. Recently, several models have been proposed in which the runtime of the processes may depend on the length of their input [8, 18, 21]. Many of the results proved in this work, in particular those involving the issue of placements of the master process, carry over to these models, and they have in fact already influenced design decisions made there.

Outline of the paper. Section 2 defines the sequential polynomial-time process calculus SPPC. In Section 3, we show that every system and every part of a system can be turned into a process expression which exactly mimics a single interactive Turing machine. The security notions are defined in Section 4. The main results, i.e., the relationships between the security notions, are proved in Section 5, with consequences for the PIOA and PITM models developed in Section 6. In Section 7, we briefly consider a less prominent security notion, called *reactive simulatability* in [5] and *security with respect to specialized simulators* in [10], and relate it to the other notions. In Section 8 we show that protocols that satisfy the FORWARDER property preserve this property when they are composed. General composition theorems are then presented in Section 9. We conclude in Section 10. The appendix contains some further details.

2 Sequential Probabilistic Process Calculus (SPPC)

In this section, we introduce Sequential Probabilistic Process Calculus (SPPC) as a language-based computational model for studying security notions. Before we formally define syntax (Section 2.3)

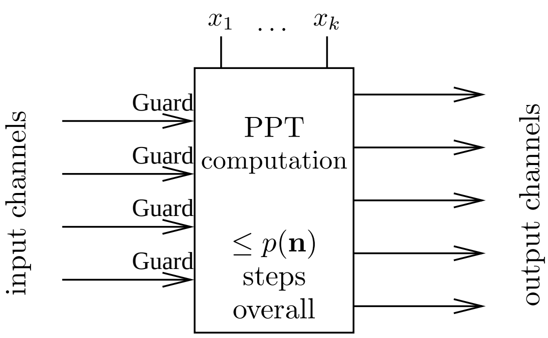

Figure 2: Probabilistic polynomial-time machines in SPPC

and semantics (Section 2.4) of our computational model, we provide an informal description (Section 2.1) and introduce the notion of a probabilistic function (Section 2.2).

2.1 Informal Introduction of SPPC

Let us first note that the driving philosophy behind the design of SPPC is to specify details of the communication model (such as a specific order of activation of entities, insecure, authenticated, secure channels, specific buffers, synchronous communication, broadcasting, and corruption) as part of the protocol specification itself rather than to explicitly encode, and thus fix, these details in the overall computation model. SPPC is expressive enough to encode such details in the protocol specifications. This makes SPPC relatively simple and flexible. In particular, a variety of security notions can easily be formulated in SPPC.

We start by discussing how individual probabilistic polynomial-time machines are modeled in SPPC and then explain how to build and execute systems of interacting machines. Our exposition parallels that of related models [5, 9, 27].

Single probabilistic polynomial-time machines. In SPPC, single machines are of the form as depicted in Figure 2. For the time being, let us ignore the "guards" and the variables $x_1, \ldots, x_k$. Conceptually, a single machine is a black-box with internal state that receives inputs, performs polynomially-bounded computation and then produces outputs. Inputs are received on input channels and outputs are written on output channels. More precisely, single machines are restricted to receiving one input and producing at most one output at a time. While this at first might appear to be a restriction, it is not really a problem since any machine that sends multiple messages can be converted to a machine that stores internally—possibly using internal buffers—the messages it wants to send, and then sends the messages one at a time on request. In fact, this style of communication corresponds exactly to the manner in which communication is defined in other *sequential* models, notably the PIOA and PITM models [9, 27]. Also, just as in these models, the overall runtime of a machine is bounded by a polynomial in the security parameter and does not depend on the number or length of inputs sent to the machine.

The channels of a single machine in SPPC correspond to ports in the PIOA model and to tapes in the PITM model. However, while messages on channels (and ports) are removed when read, this is not the case for tapes. Nevertheless, tapes can be modeled by adding machines, one for each input channel, which simulate the tapes in the obvious way. The "main machine" will then receive its input from the "tape machines". In the PIOA model, buffer machines serve a similar purpose. Note that while in SPPC and the PIOA model, the number of input and output channels/ports is not restricted, in Canetti's PITM model only one pair of input/output and input/output communication tapes is considered.

In SPPC, machines can preprocess their input using *guards* (see Figure 2) which are deterministic polynomial-time machines that are placed on input channels. Given an input on the channel, a guard may accept or reject the input. If rejected, the process does no computation. If accepted, the process receives the output of the guard. This may be different from the input, e.g., a guard can eliminate unnecessary information or transform data. The computation performed by the guard may depend on the current internal state of the process. Its runtime is polynomially-bounded in the security parameter per invocation and is not factored into the overall runtime of the process using the guard. In particular, a guard can be invoked an unbounded number of times. Since guards allow a process to discard messages without incurring a computation cost, attempts to "exhaust" a process by sending many useless messages to the process can be defeated. Additionally, using guards we can simulate an unbounded number of "virtual" channel names by prefixing each message with a session id and/or party name and then stipulating that the guards accept only those messages with the right header information. Such an ability is required for systems with a polynomial number of machines, e.g., multiparty protocols, or with multiple instances of the same protocol. While mechanisms analogous to guards are absent in other models, notably [9, 27], a newer version of PIOA [6] has a length function that, when set to zero, prevents messages from being received by the machine. This corresponds to a guard which rejects all inputs and so can be used to help avoid exhaustion attacks. However, it does not help in the creation of a mechanism analogous to virtual channels.

As mentioned above, guards can be invoked an unbounded number of times without being exhausted and in every invocation their runtime is bounded by a polynomial in the security parameter—the runtime could even depend on the length of the input. Hence, the runtime of a single machine including the guards is polynomially bounded in the security parameter *and* the number of invocations. However, the overall runtime of a single machine excluding the guards is polynomially bounded in the security parameter alone, and hence, such a machine can produce at most polynomially many output messages overall in the security parameter. Now, since guards can only be triggered by messages sent by single machines, it follows that in a system of polynomially many machines guards are only invoked a polynomial number of times in the security parameter. As shown in Section 2.4.3, from this we can conclude that such systems can be simulated by a probabilistic polynomial time Turing machine.

In SPPC, a machine may have auxiliary input, just like auxiliary input can be given to the interacting Turing machines in Canetti's model. This input is written on specific tapes before a (system of) machines is run. If such auxiliary input is used, it results in a non-uniform computational model. The tapes are represented by $x_1, \ldots, x_k$ (see Figure 2). Just like in Canetti's model, we only allow the environment machine to use auxiliary input. However, whether the environment machine is uniform or not does not affect the results presented in this paper (except for those in Section 7).

More formally, in SPPC a single machine is defined by a *process expression* $\mathcal{P}$. Such an expression corresponds to a description of an interacting Turing machine in the PITM model or an I/O automaton in the PIOA model. A process expression is always parameterized by the security parameter $\mathbf{n}$ and possibly so-called free variables $x_1, \ldots, x_k$, which represent the tapes for the auxiliary input mentioned above. Therefore, we sometimes write $\mathcal{P}(x_1, \ldots, x_k)$ instead of $\mathcal{P}$. A process expression with value i chosen for the security parameter and values $\vec{a}$ (the auxiliary inputs) substituted for its free variables $\vec{x}$ yields a *process* $\mathcal{P}(\vec{a})^{\mathbf{n}\leftarrow i}$. A process corresponds to an interacting Turing machine where the security parameter is written on the security parameter tape and the auxiliary input is written on the input tape. Hence, a process can perform computations as soon as it receives input on the input channels. As an expositional convenience, we will use the terms 'process expression' and 'process' interchangeably. A process expression is called *open* if it has free variables, and *closed* otherwise. Hence, open process expressions correspond to non-uniform machines and closed expressions to uniform ones.

Systems of interacting machines. In SPPC, a system of interacting machines is simply a multiset of single machines where an output channel of one machine connects directly to an identically-named input channel of another machine. The manner in which these machines are wired together is uniquely determined by the channel names since we stipulate that no two machines have the same input and output channel names respectively. After a machine M_1 has sent a message on an output channel, the machine waits to receive input on an input channel. The message sent on the output channel is immediately received by the machine M_2 that has an identically-named input channel. If the guards on the input channel of this machine accepts the message, then M_2 may perform some computation and produce one output message. While M_2 now waits for new input on its input channels, the output message (if any) is processed by the next receiving machine, and so on. If there is no receiving machine, or the guard of the receiving machine rejects the message, or no output message is produced, computation would halt since no machine is triggered. To avoid this, in a system of machines, one machine is always declared to be a master machine, also called *master process*, and this machine is triggered if no other machine is.

In SPPC, given process expressions $\mathcal{P}_1, \ldots, \mathcal{P}_n$, each representing a single machine, the combined system of machines is denoted by the process expression $\mathcal{P}_1 \mid \cdots \mid \mathcal{P}_n$. Instead of interpreting $\mathcal{P}_1 \mid \cdots \mid \mathcal{P}_n$ as a system of n single machines, one can consider this system as a single machine (consisting of n sub-machines). This corresponds to the transformation, in the PIOA model, of a system of fixed, finite number of machines into a single machine. However, in SPPC we can apply such transformations to systems containing a polynomial number of machines as well.

With the bounded replication operator $!_{q(\mathbf{n})}\, \mathcal{P}$, where $q(\mathbf{n})$ is some polynomial in the security parameter and $\mathcal{P}$ is a process expression (representing a single machine or a system of machines), systems containing a polynomial number of machines can be described. The process expression $!_{q(\mathbf{n})}\, \mathcal{P}$ stands for a $q(\mathbf{n})$-fold parallel composition $\mathcal{P} \mid \cdots \mid \mathcal{P}$. Note that in such a system, different copies of $\mathcal{P}$ have the same input and output channels. However, as discussed earlier, guards allow us to send messages to (virtual) channels of particular copies of a protocol. Bounded replication can be combined with parallel composition to build bigger systems such as $!_{q_1(\mathbf{n})}\, (\mathcal{P}_1 \mid \mathcal{P}_2 \mid !_{q_3(\mathbf{n})}\, \mathcal{P}_3)$.

As described earlier, since our execution model is sequential, computation may not proceed if currently executing machine produces no output, or a receiving machine rejects an input. In order to ensure that computation proceeds even in this case, we identify a master process by using a special input channel `start`. In case no output is produced by a machine, a fixed value is written on `start` thereby triggering the master process. The master process is also the first machine to be activated when execution starts.

Additionally, in studying security notions, it will be useful to define the output of a system. We do so by writing a bit, the output, onto an output channel named `decision`. The machine containing this channel is called the *decision process*. Given a process expression $\mathcal{R}(\overrightarrow{x})$ with free variables $\overrightarrow{x}$, we denote by $\texttt{Prob}[\mathcal{R}(\overrightarrow{a})^{\mathbf{n}\leftarrow i} \rightsquigarrow 1]$ the probability that $\mathcal{R}$ with security parameter i and substitution of values $\overrightarrow{a}$ for its variables $\overrightarrow{x}$ outputs a 1 on `decision`. Recall that $\mathcal{R}(\overrightarrow{a})^{\mathbf{n}\leftarrow i}$ denotes the process obtained from the process expression $\mathcal{R}$ by replacing the security parameter $\mathbf{n}$ by a value i and replacing the variables $\overrightarrow{x}$ by values $\overrightarrow{a}$. Two process expressions $\mathcal{P}(\overrightarrow{x})$ and $\mathcal{Q}(\overrightarrow{x})$ are called *equivalent* or *indistinguishable*, written $\mathcal{P}(\overrightarrow{x}) \equiv \mathcal{Q}(\overrightarrow{x})$, iff for every polynomial $p(\mathbf{n})$ there exists i_0 such that $|\texttt{Prob}[\mathcal{P}(\overrightarrow{a})^{\mathbf{n}\leftarrow i} \rightsquigarrow 1]-\texttt{Prob}[\mathcal{Q}(\overrightarrow{a})^{\mathbf{n}\leftarrow i} \rightsquigarrow 1]| \leq 1/p(i)$ for every $i \geq i_0$ and every tuple $\overrightarrow{a}$ of bit strings.

We call machines which are neither master nor decision processes *regular*. A machine which is both master and decision is called a *master decision process*. In what follows, by **R**, **M**, **D**, and **MD** we denote the set of all closed regular processes, closed master processes, open or closed decision processes, and open or closed master decision processes, respectively.

2.2 Probabilistic Functions

A *probabilistic function* F from X to Y is a function of the form $X \times Y \rightarrow [0,1]$ which satisfies the following two conditions:

1. The cardinality of the set $\{y \in Y \mid F(x,y) > 0\}$ is finite for every $x \in X$.

2. $\sum_{y\in Y} F(x,y) \leq 1$ for every $x \in X$.

We call F a *k-ary probabilistic function* if $X = Z^k = Z \times \cdots \times Z$ for some set Z.
We refer to X as the *domain* of F, to Y as the *codomain* of F, and to the set $\bigcup_{x\in X}\{y \in Y \mid F(x,y) > 0\}$ as the *range* of F.

If $\rightarrow$ is a probabilistic function, then instead of writing $\rightarrow(x,y) = p$ to say that x is mapped to y with probability p, we often write $p = \texttt{Prob}[x \rightarrow y]$ or $x \xrightarrow{p} y$ and say that x is reduced to y (by $\rightarrow$) with probability p.

Let $\rightarrow$ and $\Rightarrow$ be two probabilistic functions such that the range of $\rightarrow$ is a subset of the domain of $\Rightarrow$. Then, the composition $\rightarrow \circ \Rightarrow$ of $\rightarrow$ and $\Rightarrow$ defines the following probabilistic function: $\texttt{Prob}[x \rightarrow \circ \Rightarrow y] = \sum_z \texttt{Prob}[x \rightarrow z \Rightarrow y]$ for all x and y where z ranges over the range of $\rightarrow$ and $\texttt{Prob}[x \rightarrow z \Rightarrow y] = \texttt{Prob}[x \rightarrow z] \cdot \texttt{Prob}[z \Rightarrow y]$.

Let $\rightarrow$ be a probabilistic function such that domain and codomain coincide and let $i > 0$. Then, the probabilistic function $(\rightarrow)^i$ is defined by induction on i as follows: If $i = 0$, then $(\rightarrow)^0$ is the identity function, i.e., $\texttt{Prob}[x(\rightarrow)^0 y] = 1$ if $y = x$ and $\texttt{Prob}[x(\rightarrow)^0 y] = 0$ otherwise. For $i > 0$, we define $\texttt{Prob}[x(\rightarrow)^i y] = \texttt{Prob}[x \rightarrow \circ(\rightarrow)^{i-1} y]$.

Let $\rightarrow$ be a probabilistic function which satisfies the following conditions:

1. Domain and codomain of $\rightarrow$ coincide, say it is the set X.

2. The directed graph induced by $\rightarrow$, i.e., $(X, \{(x,y) \in X \times X \mid \texttt{Prob}[x \rightarrow y] > 0\})$ is cycle free accept for self loops.

3. For every x we have $\texttt{Prob}[x \rightarrow x] \in \{0,1\}$.

Now, the *transitive closure* $(\rightarrow)^+$ of $\rightarrow$ is defined to be the following probabilistic function:

$$\mathtt{Prob}[x(\rightarrow)^+y] = \sum_{i>0} \sum_{\substack{z_1,\dots,z_{i-1} \\ z_j \neq y \texttt{ for all } j}} \mathtt{Prob}[x \rightarrow z_1 \rightarrow \cdots \rightarrow z_{i-1} \rightarrow y]$$

for every x and y. Note that in the sums only a finite number of probabilities are different from 0.

Note that a probabilistic polynomial time Turing machine with k input tapes and l output tapes *realizes a probabilistic function* F of the form $(\{0,1\}^*)^k \times (\{0,1\}^*)^l \rightarrow [0,1]$ where we define the size of the input to be the length of the input written on the *first* tape, i.e., the Turing machine runs in polynomial time in the length of the first component of the function it realizes.

2.3 Syntax of SPPC

We now introduce the syntax of SPPC, in particular, we define process expressions. For this, we first need to introduce terms and channels.

2.3.1 Terms

Let $\mathcal{V}$ be an infinite supply of *variables*. Variables are referred to by x, y, z and decorations thereof. We write $\overrightarrow{x}$ for the sequence $x_1, \dots, x_k$ of variables. Bit strings, i.e., elements of $\{0,1\}^*$, are denoted by a, b, and decorations thereof. The empty bit string is referred to by ε. We write $\overrightarrow{a}$ to denote the sequence $a_1, \dots, a_k$ of bit strings. Let $\mathbf{n}$ be the *security parameter*.

C-terms. A *C-term* $T = T(\overrightarrow{x})$ ("C" being reminiscent of "computation") is some representation of a probabilistic function of the form $(\{0,1\}^* \times (\{0,1\}^*)^k) \times \{0,1\}^* \rightarrow [0,1]$ which can be realized by a probabilistic polynomial time Turing machine where the first component, $\{0,1\}^*$, in the domain $(\{0,1\}^* \times (\{0,1\}^*)^k)$ of this function takes the security parameter $\mathbf{n}$. The variables $\overrightarrow{x}$ are called *input variables* of $T(\overrightarrow{x})$ and the set of input variables of T is denoted by $\mathtt{var}_{\mathtt{in}}(T)$. We write $p = \mathtt{Prob}[T^{\mathbf{n} \leftarrow i}(a_1, \dots, a_k) \downarrow a]$ or simply $p = \mathtt{Prob}[T \downarrow a]$ to say that T outputs (*reduces* to) the bit string a on input $a_1, \dots, a_k$ and security parameter i with probability p. We assume that the class of C-terms is complete in the sense that all probabilistic polynomial time realizable probabilistic functions can be described by some C-term. Obviously we can achieve this by simply taking representations of probabilistic polynomial-time Turing machines to be C-terms.

C-terms will be used in processes of the form $\mathtt{out}(c, T)$ to compute a bit string which is then placed on the channel c.

M-terms. An *M-term* (also called *guard*) $t = t(\overrightarrow{x}; \overrightarrow{y})$ ("M" being reminiscent of "matching") with $\overrightarrow{x} = x_1, \dots, x_k$ and $\overrightarrow{y} = y_1, \dots, y_l$ is some representation of a (*deterministic*) function F of the form $(\{0,1\}^* \times \{0,1\}^* \times (\{0,1\})^k) \times (\{0,1\}^* \times (\{0,1\}^*)^l) \rightarrow \{0,1\}$ that can be realized by a *deterministic* polynomial time Turing machine where the first component of the domain $(\{0,1\}^* \times \{0,1\}^* \times (\{0,1\})^k)$ of this function takes the security parameter $\mathbf{n}$. The variables $\overrightarrow{x}$ are call *input variables* of t and the variables in $\overrightarrow{y}$, which are required to be distinct from the input variables, are called *output variables*. Intuitively, an M-term t works as follows: Given bit strings $\overrightarrow{a}$ for the input variables (the last k components of the domain of t), on receiving a bit string a (the second component of the domain of t) the term t either rejects a or accepts it (i.e., 0 or 1 is returned as the first component of the codomain $(\{0,1\}^* \times (\{0,1\}^*)^l)$ of t). In case t accepts, it produces l bit strings (the last l components of the codomain of t) which are substituted for the output variables $\vec{y}$. In case t rejects, this output is irrelevant. We assume that the class of M-terms is complete in the sense that all deterministic functions F realizable by a deterministic polynomial-time Turing machine can be described by some M-term. Obviously this is possible by taking representations of such Turing machines as M-terms.

For our purposes, M-terms with only one output variable would suffice. However, additional output variables make it more convenient to define certain processes.

M-terms will occur in processes of the form $P = \mathtt{in}(c,t).P'$. If a bit string a is received on channel c, the M-term t allows to parse a (maybe depending on external input) before P actually reads a, and computes substitutions for the output variables in case a is accepted by t. The output variables of t may occur in P'.

Often, t is of the form x which we will interpret as an M-term without input variables and output variable x. This M-term accepts any message and this message is substituted for x. The runtime of such an M-term is determined by the bandwidth of c, which is a polynomial in the security parameter (see below). Conversely, since the computation of an M-term is polynomial bounded in the security parameter, this polynomial could be considered the bandwidth of the channel the M-term operates on. Hence, one could dispense with explicitly assigning bandwidth to channels.

The purpose of M-terms was explained in Section 2.1 where M-terms where referred to as "guards". As explained there, one reason for introducing M-terms is that they allow us to create new virtual channels: An M-term can reject all messages that do not start with a certain bit string b where b stands for a session ID, i.e., it can check whether messages are of the form (b,x). When later different instances of a protocol are modeled, which all have there unique SID, then every instance will only accept a message that is prefixed with the correct SID. Also, every message returned by an instance will be prefixed by the SID of the instance. This will allow us to send/receive messages to/from specific instances of protocols without introducing channel variables as in π-calculus [23,24].

2.3.2 Channels

A *channel* is a tuple c consisting of a *channel name* $\mathtt{name}(c)$, a *bandwidth* $\mathtt{bw}(c)$, and a *priority* $\mathtt{prior}(c)$. The bandwidth $\mathtt{bw}(c)$ is a polynomial in the security parameter and determines the maximum length of messages that can be sent through c. Strictly speaking, as also mentioned above, the bandwidth is not needed since M-terms implicitly bound the length of messages read from a channel. However, making this bound explicit is more convenient. The *priority* $\mathtt{prior}(c)$ of c can take the values high and low, and accordingly, we refer to *high* and *low* channels. A message on a high channel will be scheduled before a message on a low channel. The intuition is that high channels are used to update the internal state of a process, while low channels are used to communicate with external processes or internal subprocesses. In Section 2.1, we only referred to low channels and for the sake of presentation did not mention the internal step to update the internal state that is taken after the output on the low channel has been produced. Channels are usually referred to by c and decorations thereof. The set of channels is denoted by C. We assume that C contains the low channels `start` and `decision`. The channel `start` will be used as input channel of what we will call the master process which is activated via the `start` channel if no further communication is possible. The environment, which is trying to distinguish a protocol from its ideal version, will use the `decision` channel to output its decision.

2.3.3 Sequential Process Expressions and Processes

A *sequential process expression* $\mathcal{P}$ is defined by the following grammar:

$$
\begin{array}{rcll}
\mathcal{P} & ::= & \mathbf{0} & | \hfill \text{(termination)} \\
 & & \mathcal{S} & | \hfill \text{(wait for input on different channels)} \\
 & & (\mathcal{P} \mid\mid \mathcal{P}) & | \hfill \text{(parallel composition)} \\
 & & !_{q(\mathbf{n})}\, \mathcal{P} & | \hfill \text{(bounded replication)} \\
\mathcal{P}_{\mathtt{out}} & ::= & (\mathcal{O}_L \mid\mid \mathcal{O}_H \mid\mid \mathcal{P}) & \hfill \text{(process with initial output)} \\
\mathcal{O}_H & ::= & \mathtt{out}(c_H, T) & | \hfill \text{(output on high channel } c_H\text{)} \\
 & & \mathbf{0} & \\
\mathcal{O}_L & ::= & \mathtt{out}(c_L, T) & | \hfill \text{(output on low channel } c_L\text{)} \\
 & & \mathbf{0} & \\
\mathcal{S} & ::= & \mathtt{in}(c_H, t).\mathcal{P} & | \hfill \text{(wait for input on high channel } c_H\text{)} \\
 & & \mathtt{in}(c_L, t).\mathcal{P}_{\mathtt{out}} & | \hfill \text{(wait for input on low channel } c_L\text{)} \\
 & & (\mathcal{S} + \mathcal{S}) & \hfill \text{(wait for input on different channels)}
\end{array}
$$

where c_L and c_H stand for low and high channels, respectively, and t and T are M- and C-terms, respectively. We will require that high channels occurring in a sequential process expression $\mathcal{P}$ are internal. Internal and external channels are defined below.

Expressions of the form $\mathcal{P}_{\mathtt{out}}$ are called *process expressions with initial output*. The notion *process expression* refers to both sequential process expressions and process expressions with initial output.

The set of channels occurring in a process expression $\mathcal{P}$ is denoted by $\mathsf{C}(\mathcal{P})$. A channel c is called *internal* in $\mathcal{P}$ if it occurs both in an expression of the form $\mathtt{in}(c,t).\mathcal{P}'$ and $\mathtt{out}(c,T)$; otherwise, it is called *external*. The set of internal and external channels of $\mathcal{P}$ is denoted by $\mathsf{C}_{\mathtt{int}}(\mathcal{P})$ and $\mathsf{C}_{\mathtt{ext}}(\mathcal{P})$, respectively. The set $\mathsf{C}_{\mathtt{ext}}(\mathcal{P})$ is further partitioned into *input* ($\mathsf{C}_{\mathtt{in}}(\mathcal{P})$) and *output* channels ($\mathsf{C}_{\mathtt{out}}(\mathcal{P})$) in the obvious way.

Given a non-negative integer i (represented by a bit string), a *process* $P = \mathcal{P}^{\mathbf{n}\leftarrow i}$ of a process expression $\mathcal{P}$ is obtained from $\mathcal{P}$ by replacing every occurrence of $!_{q(i)}\ \mathcal{Q}^{\mathbf{n}\leftarrow i}$ by

$$\overbrace{\mathcal{Q}^{\mathbf{n}\leftarrow i} \mid\mid \cdots \mid\mid \mathcal{Q}^{\mathbf{n}\leftarrow i}}^{q(i)\text{ times}}.$$

We call i the *parameter associated with* P. In P, M-terms and C-terms are evaluated using i as the security parameter.

Note that modulo commutativity and associativity of the parallel composition operator " $||$ ", a process P is of the form P', $\mathtt{out}(c,T) \mid\mid P'$, $\mathtt{out}(c',T') \mid\mid P'$, or $\mathtt{out}(c,T) \mid\mid \mathtt{out}(c',T') \mid\mid P'$ where P' is a process obtained from a sequential process expression (i.e., without initial output), c is a high channel, and c' is a low channel. In other words, at most two messages are currently on channels (at most one on a high channel and at most one on a low channel). In what follows, we refer to processes such as P' by *processes without output*.

Since process expressions and processes are (formal) terms, we sometimes consider them as finite ordered trees with labeled nodes.

Given a process (expression) P, the set of free variables $\mathtt{free}(P)$ of P is the set of input variables of C-terms occurring in P which are not bounded by an input expression. Formally, $\mathtt{free}(P)$ is defined inductively as follows:

- $\mathtt{free}(\mathbf{0}) = \emptyset$,
- $\mathtt{free}(\mathtt{in}(c,t).P) = \mathtt{free}(P) \setminus \mathtt{var}_{\mathtt{out}}(t)$,
- $\mathtt{free}(\mathtt{out}(c,T)) = \mathtt{var}_{\mathtt{in}}(T)$,
- $\mathtt{free}(P \mid\mid Q) = \mathtt{free}(P) \cup \mathtt{free}(Q)$,
- $\mathtt{free}(!_{q(\mathbf{n})}\ P) = \mathtt{free}(P)$,
- $\mathtt{free}(P + Q) = \mathtt{free}(P) \cup \mathtt{free}(Q)$.

We write $P(\vec{x})$ with $\vec{x}= x_1, \ldots, x_k$ to say that $\{x_1, \ldots, x_k\} \subseteq \mathtt{free}(P)$. A process (expression) P is called *closed* if $\mathtt{free}(P) = \emptyset$.

If $P(\vec{x})$ is a process (expression) and $\vec{a}$ is a sequence of bit strings (of the same length as $\vec{x}$), then $P(\vec{a})$ denotes the process (expression) obtained from $P(\vec{x})$ by replacing every free occurrence of x_i by a_i. The M-term and C-terms in $P(\vec{a})$ containing a free occurrence of x_i will be evaluated with x_i replaced by a_i.

Let P be a process and t be an M-term. If t accepts a bit string a, then, as explained, it (deterministically) produces bit strings $a_1, \ldots, a_k$ as output and these bit string are substituted for the output variables $\mathtt{var}_{\mathtt{out}}(t) = \{x_1, \ldots, x_k\}$ of t. We write $[a/t]P$ to denote the process obtained from P by substituting every free occurrence of x_i in P by a_i.

The *communication size* $\mathtt{comsize}(P)$ of a process P is the number of occurrences of input and output processes in P. For instance, $\mathtt{comsize}(\mathtt{out}(c,T) \mid\mid \mathtt{out}(c,T) \mid\mid \mathtt{in}(c,t).\mathtt{in}(c,t)) = 4$.

The *communication size* $\mathtt{comsize}(\mathcal{P})(\mathbf{n})$ of a process expression $\mathcal{P}$ is a polynomial $q(\mathbf{n})$ such that $q(i) = \mathtt{comsize}(\mathcal{P}^{\mathbf{n}\leftarrow i})$ for every i. Clearly, such a polynomial exists.

2.3.4 Contexts

A *context* $C[\,]$ is a process where exactly one leave is labeled with $[\,]$.

The process $C[P]$ is obtained from the context $C[\,]$ by plugging the process P into the hole of $C[\,]$. This notation is used to refer to some subprocess P of a given process $P' = C[P]$.

Given a process P, we call a context $C[\,]$ an *input context* for P, if there exists a process of the form $\mathtt{in}(c,t).Q$, called the *input process associated with* $C[\,]$, such that $C[\mathtt{in}(c,t).Q] = P$ and on the path from the root of $C[\,]$ to its hole all nodes are labeled with ‘ $\mid\mid$ ’ or ‘$+$’. This implies that P is ready to receive input on channel c (with M-term t).

2.4 Semantics of SPPC

Processes have an interleaving semantics which is defined in terms of reductions. Roughly speaking, given a process P, the reduction is carried out by iteratively performing the following steps until nothing changes:

1. **Reduction.** All C-terms occurring in P where all the variables are replaced by bit strings are reduced. After this step, there is at most one message on a high channel and at most one message on a low channel in P, i.e., modulo commutativity and associativity of “ $\mid\mid$ ”, P is of the form P', $\mathtt{out}(c,a) \mid\mid P'$, $\mathtt{out}(c',a') \mid\mid P'$, or $\mathtt{out}(c,a) \mid\mid \mathtt{out}(c',a') \mid\mid P'$ where P' is a process without output, c is a high channel, and c' is a low channel.

2. **Communication.** In the first case (where no message is on any channel), the empty bit string ε is put on `start` which is then read by the process. In case there is only one message on a (high or low) channel, then this message is read. In the last case (where there is one message on a high channel and one on a low channel), the message a on the high channel c is read by the process. (Note that, by definition of process expressions, after reading a on c, the process does not produce new output and in the next iteration step, the message on the low channel will be read.)

The intuition is that high channels are used to update the current (internal) state of a process (and therefore have priority over low channels) while low channels are intended for communication with external processes or internal subprocesses.

In what follows, every single step is defined formally and the steps are put together in Section 2.4.3.

2.4.1 C-term Reduction

The reduction of C-terms occurring in a process is defined by the probabilistic function $\rightharpoonup$ on *closed* processes. For open processes $P(\vec{x})$, the free variables $\vec{x}$ are first substituted by bit strings.

Formally, $P \overset{p}{\rightharpoonup} Q$ is defined by structural induction on P where we assume that the security parameter associated with P is i:

- $\mathbf{0} \overset{1}{\rightharpoonup} \mathbf{0}$,
- $\mathtt{in}(c,t).P \overset{1}{\rightharpoonup} \mathtt{in}(c,t)P$,
- $\mathtt{out}(c,T) \overset{p}{\rightharpoonup} \mathtt{out}(c,a)$ if $p = \sum_{b \equiv a \bmod 2^{\mathtt{bw}(c)(i)}-1} \mathtt{Prob}[T \downarrow b]$.
- $P \mid\mid Q \overset{p}{\rightharpoonup} P' \mid\mid Q'$ if $P \overset{q}{\rightharpoonup} P'$, $Q \overset{q'}{\rightharpoonup} Q'$, and $p = q \cdot q'$,
- $P + Q \overset{1}{\rightharpoonup} P + Q$.

For the cases not covered above we define $P \overset{0}{\rightharpoonup} Q$. Note that since we assume P to be closed, all input variables of the C-term T have been substituted by bit strings.

2.4.2 Communication

To define the communication step, we first introduce a probabilistic function $\rightarrow_{(c,a)}$ on closed processes which describes how a message a on channel c is read by a process.

Formally, we define $P \rightarrow^{p}_{(c,a)} Q$ iff the following is true: Let N be the number of different input contexts $C[\ ]$ of P with associated input processes of the form $\mathtt{in}(c,t).P'$ such that t accepts a. In other words, N is the number of input expressions in P ready to receive input a on channel c. Then,

1. If $N = 0$, then $P = Q$ and $p = 1$, or $P \neq Q$ and $p = 0$.
2. If $N \neq 0$, then $Q = C[[a/t]P']$ and $p = 1/N$ for some $C[\ ]$ and $\mathtt{in}(c,t).P'$ as above, and otherwise $p = 0$.

Now, we are ready to define a *communication step* $\to$ on closed processes. We set $P \xrightarrow{p} Q$ if the following is true:

1. If $P = \mathtt{out}(\mathtt{decision}, a) \mid\mid P'$ for some process P', then $P = Q$ and $p = 1$, or $P \neq Q$ and $p = 0$. (Since in P output was written on the channel $\mathtt{decision}$, no further step shall be taken.) Otherwise:

2. If P is a process without output, then $p = \mathtt{Prob}[P \to_{(\mathtt{start},\varepsilon)} Q]$.

3. If $P = \mathtt{out}(c, a) \mid\mid P'$ for some process P' without output, then $p = \mathtt{Prob}[P' \to_{(c,a)} Q]$.

4. If $P = \mathtt{out}(c, a) \mid\mid \mathtt{out}(c', a') \mid\mid P'$ where P' is a process without output and c is a high channel, then $p = \mathtt{Prob}[\mathtt{out}(c', a') \mid\mid P' \to_{(c,a)} Q]$.

In all other cases, we define $P \xrightarrow{0} Q$.

2.4.3 The Complete Reduction of Processes

The probabilistic function $\Rightarrow$ defines the complete reduction of processes.

For all processes P, Q, and probabilities p we define

$$P \overset{p}{\Rightarrow} Q \text{ iff } p = \mathtt{Prob}[P \rightharpoonup \circ (\to \circ \rightharpoonup)^{+} Q].$$

The probability p that the decision returned by a closed process P is 1 is written $\mathtt{Prob}[P \rightsquigarrow 1]$ or $P \overset{p}{\rightsquigarrow} 1$ and is defined as follows:

$$\mathtt{Prob}[P \rightsquigarrow 1] = \sum_{Q} \mathtt{Prob}[P \Rightarrow Q]$$

where Q ranges over all processes of the form $\mathtt{out}(\mathtt{decision}, 1) \mid\mid Q'$ (modulo commutativity and associativity of '$\mid\mid$').

The following theorem tells us that the computation of process expressions can be simulated by a probabilistic polynomial time Turing machine.

Theorem 1. *Let $\mathcal{P}(x_1, \ldots, x_k)$ be a process expression. There exists a probabilistic polynomial-time Turing machine that for all processes Q (modulo associativity of '$+$' and '$\mid\mid$') returns Q on input $i, \mathcal{P}(x_1, \ldots, x_k), a_1, \ldots, a_k$ (given on separate input tapes) with probability p iff $\mathcal{P}(a_1, \ldots, a_k)^{\mathbf{n} \leftarrow i} \overset{p}{\Rightarrow} Q$. The Turing machine runs in polynomial time in the security parameter i.*

3 Single Machine Normal Form

In this section, we show that sequential process expressions can be turned into what we call single machine normal form.

As explained informally in Section 2.1, these normal forms correspond to probabilistic polynomial time IO automata (PIOAs) or Interactive Turing Machines (ITMs) which in every step read exactly one message (on some external channel) as input and produce at most one message as output, and which, in addition, have guards (M-terms) which allow them to reject or accept their input. If they reject the input, then the message sent is dropped and no further computation is carried out. (In this case, the master process is triggered by reading a message on $\mathtt{start}$). The guards are deterministic Turing machines which run in polynomial time in the security parameter. If a process (in SMNF) is open, i.e., has free variables, then this corresponds to a non-uniform PIOA/ITM (with guards), and otherwise to a uniform one.

Guards add additional power to processes (or PIOA and ITM extended by guards). The machine itself could inspect the input and decide whether to accept or reject it, and in the latter case, would simply produce no output. However, the overall run time of machines is bounded by a polynomial in the security parameter and is independent of the number of invocations to a machine. As a consequence, inspecting the input consumes resources. The idea behind the guards is that they are invoked anytime a message is sent on an input channel and that their run time is not added to the run time of the "main machine". Consequently, using guards, inspecting the input does not consume resources. In other words, guards of processes (or PIOA and ITM extended by guards) are devices of a machine whose run time may depend on the number of invocations (where within one invocation the run time is polynomial in the security parameter).

We will sometimes consider processes in what we call *simple* single machine normal form (SSMNF) where the guards accept all messages. We refer to the fragment of SPPC where all process are parallel compositions of processes in SSMNF the *guard-free fragment of SPPC*.

A process expression in single machine normal form works as follows: Before processing a message on an input channel (a low channel), an M-term (the "guard") on this channel decides whether to accept or reject this message. The message is only processed further if the M-term accepts the message and otherwise the message is dropped. In the former case, the internal state is updated (via a high channel) depending on the current internal state and the current message, and next at most one message is written on an output channel (a low channel). Further intuition is given below. Formally, the single machine normal form is defined as follows.

Definition 2. *We say that a sequential process expression $\mathcal{P}(\vec{z})$ is in* single machine normal form (SMNF) *if it is of the following form:*

$$\mathcal{S}_{\mathtt{init}}(\vec{z}) \mid\mid !_{q(\mathbf{n})}\ \mathcal{S}(\vec{z}) \tag{1}$$

where

$$\begin{aligned}\mathcal{S}(\vec{z}) \quad = \quad & \mathtt{in}(c_{\mathtt{s}}, z). \sum_{c \in \mathsf{C}_{\mathtt{in}}(\mathcal{P}(\vec{z}))} \mathtt{in}(c, t_{\mathtt{in}}(c, z, \vec{z}; x)).\Bigg(\mathtt{out}(c_{\mathtt{ns}}, T_{\mathtt{ns}}(c, x, z, \vec{z})) \mid\mid \\ & \Bigg(\sum_{c' \in \mathsf{C}_{\mathtt{out}}(\mathcal{P}(\vec{z}))} \mathtt{in}(c_{\mathtt{ns}}, t_{\mathtt{out}}(c'; y, v)).\Big(\mathtt{out}(c_{\mathtt{s}}, y) \mid\mid \mathtt{out}(c', v)\Big) + \\ & \mathtt{in}(c_{\mathtt{ns}}, t_{\mathtt{empty}}(y)).\mathtt{out}(c_{\mathtt{s}}, y)\Bigg)\Bigg)\end{aligned}$$

and $\mathcal{S}_{\mathtt{init}}(\vec{z})$ is defined in the same way except that $\mathtt{in}(c_{\mathtt{s}}, z)$ is removed and every occurrence of z is replaced by ε (representing the initial state of $\mathcal{P}$).

The internal channels $c_{\mathtt{s}}$ and $c_{\mathtt{ns}}$ (which carry the current and the updated state, respectively) are defined to be high channels while all other channels are (external, and thus) low channels.

We say that a sequential process expression is in simple single machine normal form (SSMNF) *if it is in SMNF and the M-term $t_{\mathtt{in}}(c, z, \vec{z}; x)$ accepts all incoming messages. We call the fragment of*

SPPC where all processes are parallel compositions of processes in SSMNF the guard-free fragment of SPPC.

Intuitively, the variable z stores the current state of $\mathcal{P}$. The M-term $t_{\mathtt{in}}(c, z, \vec{z}; x)$ is the guard which is used to decide whether $\mathcal{P}$ accepts or rejects the input on c. This term takes as input the name of the channel c from which input shall be read, the current state z, and the external inputs $\vec{z}$. Its output is written into x. The C-term $T_{\mathtt{ns}}(c, x, z, \vec{z})$ computes the new state depending on the current state z, the output x of $t_{\mathtt{in}}(c, z, \vec{z}; x)$, and the free variables $\vec{z}$. The M-term $t_{\mathtt{out}}(c'; y, v)$ intuitively accepts an input a (which is the current state) if a encodes that the next output, say b, shall be written on c'. If $t_{\mathtt{out}}(c'; y, v)$ accepts a, then y is substituted by a and v by b. The M-term $t_{\mathtt{empty}}(y)$ works similarly. It only accepts a if a encodes that nothing shall be written on an output channel.

The following lemma tells us that every sequential process expression can be turned into single machine normal form. The proof of the following lemma is rather simple. Intuitively, the sequential process expressions $\mathcal{P}_1$ and $\mathcal{P}_2$ in this lemma are independent processes that may communicate (see also Section 2.1 and 4.1).

Lemma 3. *Let $\mathcal{P}_1 = \mathcal{P}_1(\vec{z})$ and $\mathcal{P}_2 = \mathcal{P}_2(\vec{z})$ be (possibly open) sequential process expressions such that $\mathsf{C}(\mathcal{P}_1) \cap \mathsf{C}_{\mathtt{int}}(\mathcal{P}_2) = \emptyset$, $\mathsf{C}_{\mathtt{int}}(\mathcal{P}_1) \cap \mathsf{C}(\mathcal{P}_2) = \emptyset$, $\mathsf{C}_{\mathtt{in}}(\mathcal{P}_1) \cap \mathsf{C}_{\mathtt{in}}(\mathcal{P}_2) = \emptyset$, and $\mathsf{C}_{\mathtt{out}}(\mathcal{P}_1) \cap \mathsf{C}_{\mathtt{out}}(\mathcal{P}_2) = \emptyset$. Then, $\mathcal{P}_1$ and $\mathcal{P}_2$ can be turned into single machine normal forms $\mathcal{P}'_1$ and $\mathcal{P}'_2$, respectively, such that*

$$
\begin{aligned}
\mathtt{Prob}[(\mathcal{P}_1(\vec{a}) \mid\mid \mathcal{P}_2(\vec{a}))^{\mathbf{n}\leftarrow i} \rightsquigarrow 1] &= \mathtt{Prob}[(\mathcal{P}'_1(\vec{a}) \mid\mid \mathcal{P}_2(\vec{a}))^{\mathbf{n}\leftarrow i} \rightsquigarrow 1] \\
&= \mathtt{Prob}[(\mathcal{P}_1(\vec{a}) \mid\mid \mathcal{P}'_2(\vec{a}))^{\mathbf{n}\leftarrow i} \rightsquigarrow 1] \\
&= \mathtt{Prob}[(\mathcal{P}'_1(\vec{a}) \mid\mid \mathcal{P}'_2(\vec{a}))^{\mathbf{n}\leftarrow i} \rightsquigarrow 1]
\end{aligned}
$$

for every i and tuple $\vec{a}$. In particular,

$$\mathcal{P}_1 \mid\mid \mathcal{P}_2 \equiv \mathcal{P}'_1 \mid\mid \mathcal{P}_2 \equiv \mathcal{P}_1 \mid\mid \mathcal{P}'_2 \equiv \mathcal{P}'_1 \mid\mid \mathcal{P}'_2.$$

PROOF. We prove that $\mathcal{P}_1$ can be turned into single machine normal form $\mathcal{P}'_1$ such that $\mathtt{Prob}[(\mathcal{P}_1(\vec{a}) \mid\mid \mathcal{P}_2(\vec{a}))^{\mathbf{n}\leftarrow i} \rightsquigarrow 1] = \mathtt{Prob}[(\mathcal{P}'_1(\vec{a}) \mid\mid \mathcal{P}_2(\vec{a}))^{\mathbf{n}\leftarrow i} \rightsquigarrow 1]$ for every i and $\vec{a}$. Turning $\mathcal{P}_2$ into single machine normal form is done in the same way.

To define $\mathcal{P}'_1(\vec{z})$, we need to specify the C-term $T_{\mathtt{ns}}(c, x, z, \vec{z})$, the M-terms $t_{\mathtt{in}}(c, z, \vec{z}; x)$, $t_{\mathtt{out}}(c'; y, v)$, and $t_{\mathtt{empty}}(y)$, and the polynomial $q(\mathbf{n})$ in (1).

We start with the definition of $t_{\mathtt{in}}(c, z, \vec{z}; x)$. The variable z will be substituted by some representation of a process P_1. From P_1 together with $\vec{z}$, $t_{\mathtt{in}}(c, z, \vec{z}; x)$ can determine whether P_1 would accept or reject a given input a on c as follows: First, $t_{\mathtt{in}}(c, z, \vec{z}; x)$ determines from P_1 the set $t_1, \ldots, t_l$ of M-terms occurring in input processes $\mathtt{in}(c, t_i).P'_1$ associated with input contexts of P_1. Note that l is polynomially bounded in the security parameter. If $l = 0$, then this means that P_1 does not accept any input on c, and therefore, $t_{\mathtt{in}}(c, z, \vec{z}; x)$ will reject every input on c. Otherwise, given an input a, $t_{\mathtt{in}}(c, z, \vec{z}; x)$ will apply every t_i to a. If every t_i rejects a, then $t_{\mathtt{in}}(c, z, \vec{z}; x)$ rejects a as well. Otherwise, $t_{\mathtt{in}}(c, z, \vec{z}; x)$ copies a into the output x.

The C-term $T_{\mathtt{ns}}(c, x, z, \vec{z})$ evaluates $z = P_1$ based on x and $\vec{z}$. More precisely, if $z = \varepsilon$ and the security parameter is i, then $T_{\mathtt{ns}}(c, a, \varepsilon, \vec{a})$ does the following: $T_{\mathtt{ns}}(c, a, \varepsilon, \vec{a})$ computes and outputs a process Q by first applying $\rightarrow_{(c,a)}$ to $\mathcal{P}_1(\vec{a})^{\mathbf{n}\leftarrow i}$ and then, to the result, alternatively applying $\rightharpoonup$ and $\rightarrow$ until for the current process P_1 one of the following is true: a) P_1 is a process without output, b) $P_1 = \mathtt{out}(c,a) \mid\mid P_1'$ for some process P_1' without output and an external (i.e., low) channel c, or c) P_1 occurred before. (Note that if P_1 is a process with output on a high and a low channel, then $T_{\mathtt{ns}}(c,a,\varepsilon,\vec{a})$ further simulates the computation of P_1, i.e., have P_1 read the message on the high channel. This results in a process with output only on the low channel.) Analogously to Theorem 1, it is easy to see that this reduction of $\mathcal{P}_1(\vec{a})^{\mathbf{n}\leftarrow i}$ can be carried out by a probabilistic polynomial time algorithm. If $z \neq \varepsilon$, then z is a process P_1. In this case, $T_{\mathtt{ns}}(c,x,z,\vec{z})$ simulates the reduction of this process in the same way as described above.

The M-term $t_{\mathtt{out}}(c';y,v)$ receives as input a process, say P_1. If P_1 is a process without output, $t_{\mathtt{out}}(c';y,v)$ rejects P_1. Otherwise, P_1 is of the form $\mathtt{out}(c,a) \mid\mid P_1'$ for some process P_1' without output and an external channel c. If $c \neq c'$, then again, $t_{\mathtt{out}}(c';y,v)$ rejects P_1. Otherwise, y is substituted by P_1' and v by a.

The behavior of the M-term $t_{\mathtt{empty}}(y)$ is analogous to the one of $t_{\mathtt{out}}(c';y,v)$. It accepts P_1 iff it is a process without output, and in this case, y is substituted with P_1.

We define the polynomial $q(\mathbf{n})$ to be the communication size $\mathtt{comsize}(\mathcal{P}_1(\vec{z}))$ of $\mathcal{P}_1(\vec{z})$. Note that due to the M-term $t_{\mathtt{in}}(c,z,\vec{z};x)$, the single machine normal form of $\mathcal{P}_1$ reads an input a on a channel c iff $\mathcal{P}_1$ would read a on c. In particular, if $\mathcal{P}_1$ currently can not read a on c (because there is no input process for c or all M-terms reject a), then the single machine normal form would not read a on c as well, and thus would not be activated in the first place. This guarantees that the single machine normal form is activated by receiving external messages exactly as often as $\mathcal{P}_1$, and therefore, it suffices to define $q(\mathbf{n})$ as done above. Without $t_{\mathtt{in}}(c,z,\vec{z};x)$, the decision whether a can be read by $\mathcal{P}_1$ would only be made when evaluating $T_{\mathtt{ns}}(c,x,z,\vec{z})$. Hence, the single machine normal form would always be activated even if $\mathcal{P}_1$ would reject the input. In this case, the number of activations of the single machine normal form could not be bounded by $q(\mathbf{n})$.

Now, $\mathtt{Prob}[(\mathcal{P}_1(\vec{a}) \mid\mid \mathcal{P}_2(\vec{a})^{\mathbf{n}\leftarrow i} \rightsquigarrow 1] = \mathtt{Prob}[(\mathcal{P}_1'(\vec{a}) \mid\mid \mathcal{P}_2(\vec{a})^{\mathbf{n}\leftarrow i} \rightsquigarrow 1]$ is easy to verify. □

We note that Lemma 3 does not hold for SSMNFs since a process in SSMNF has to process all input messages. For instance, there is no SSMNF equivalent to the process expression $\mathcal{P} = \mathtt{in}(c_0,x).\mathtt{in}(c_1,y).\mathtt{out}(c_2,y)$: Assume that there is a process $\mathcal{Q}$ in SSMNF equivalent to $\mathcal{P}$. Let $q(\mathbf{n})$ be the communication size of this machine. An environment $\mathcal{E}$ could send $q(\mathbf{n})+1$ random messages on c_1 before sending a message on c_0, and then a random message m on c_1. The environment $\mathcal{E}$ will output 1 on $\mathtt{decision}$ if m is returned on c_{out}, and otherwise, if $\mathcal{E}$ is triggered on $\mathtt{start}$, it will output 0 on $\mathtt{decision}$. When interacting with $\mathcal{P}$, the message m will be returned to $\mathcal{E}$ on c_2. However, when interacting with $\mathcal{Q}$, no message will be returned since $\mathcal{Q}$ has already terminated as it can only process $q(\mathbf{n})$ messages. Thus, $\mathcal{E} \upharpoonright \mathcal{P} \not\equiv \mathcal{E} \upharpoonright \mathcal{Q}$.

Remark 4. *Lemma 3 does not hold, in general, when restricted to SSMNFs.*

4 Definition of Security Notions

In this section, we introduce the security notions strong simulatability, strong and weak blackbox simulatability, and universal composability. We first need some more notation.

4.1 Channel Configurations

To define the security notions, we need to specify how different processes (describing the environment, the real/ideal adversary, the simulator, the real/ideal protocol) can be connected via channels. At the end of this section, we provide an example to illustrate the definitions.

Recall from Section 2 that $\mathsf{C}(\mathcal{P})$ denotes the set of channels of $\mathcal{P}$ and $\mathsf{C}_{\mathtt{int}}(\mathcal{P})$, $\mathsf{C}_{\mathtt{ext}}(\mathcal{P})$, $\mathsf{C}_{\mathtt{in}}(\mathcal{P})$, and $\mathsf{C}_{\mathtt{out}}(\mathcal{P})$ are the sets of internal, external, input, and output channels of $\mathcal{P}$, respectively.

The set of external channels of a process expression $\mathcal{P}$ is further partitioned into the set of *IO channels* $\mathsf{C}^{\mathtt{io}}_{\mathtt{ext}}(\mathcal{P})$ and the set of *network channels* $\mathsf{C}^{\mathtt{net}}_{\mathtt{ext}}(\mathcal{P})$. Thus, the set of external channels of P is partitioned into the set of input IO channels $\mathsf{C}^{\mathtt{io}}_{\mathtt{in}}(\mathcal{P})$, output IO channels $\mathsf{C}^{\mathtt{io}}_{\mathtt{out}}(\mathcal{P})$, input network channels $\mathsf{C}^{\mathtt{net}}_{\mathtt{in}}(\mathcal{P})$, and output network channels $\mathsf{C}^{\mathtt{net}}_{\mathtt{out}}(\mathcal{P})$. We require that if $\mathtt{decision} \in \mathsf{C}(\mathcal{P})$, then $\mathtt{decision} \in \mathsf{C}^{\mathtt{io}}_{\mathtt{out}}(\mathcal{P})$. Also, if $\mathtt{start} \in \mathsf{C}(\mathcal{P})$, then $\mathtt{start} \in \mathsf{C}^{\mathtt{io}}_{\mathtt{in}}(\mathcal{P})$.

We say that two process expressions $\mathcal{P}$ and $\mathcal{Q}$ are *compatible* iff they have the same set of external channels and these channels are of the same type, i.e., $\mathsf{C}^{\mathtt{net}}_{\mathtt{in}}(\mathcal{P}) = \mathsf{C}^{\mathtt{net}}_{\mathtt{in}}(\mathcal{Q})$, $\mathsf{C}^{\mathtt{io}}_{\mathtt{in}}(\mathcal{P}) = \mathsf{C}^{\mathtt{io}}_{\mathtt{in}}(\mathcal{Q})$, $\mathsf{C}^{\mathtt{net}}_{\mathtt{out}}(\mathcal{P}) = \mathsf{C}^{\mathtt{net}}_{\mathtt{out}}(\mathcal{Q})$, and $\mathsf{C}^{\mathtt{io}}_{\mathtt{out}}(\mathcal{P}) = \mathsf{C}^{\mathtt{io}}_{\mathtt{out}}(\mathcal{Q})$. They are *IO-compatible* iff they have the same set of IO channels and disjoint sets of network channels, i.e., $\mathsf{C}^{\mathtt{net}}_{\mathtt{ext}}(\mathcal{P}) \cap \mathsf{C}^{\mathtt{net}}_{\mathtt{ext}}(\mathcal{Q}) = \emptyset$, $\mathsf{C}^{\mathtt{io}}_{\mathtt{in}}(\mathcal{P}) = \mathsf{C}^{\mathtt{io}}_{\mathtt{in}}(\mathcal{Q})$, and $\mathsf{C}^{\mathtt{io}}_{\mathtt{out}}(\mathcal{P}) = \mathsf{C}^{\mathtt{io}}_{\mathtt{out}}(\mathcal{Q})$.

Given sequential process expressions $\mathcal{P}$ and $\mathcal{Q}$, by $\mathcal{P} \upharpoonright \mathcal{Q}$ we denote the parallel composition $\mathcal{P}' \mid\mid \mathcal{Q}'$ where $\mathcal{P}'$ and $\mathcal{Q}'$ are obtained from $\mathcal{P}$ and $\mathcal{Q}$ by renaming the internal channels of $\mathcal{P}$ and $\mathcal{Q}$, respectively, in such a way that $\mathsf{C}(\mathcal{P}') \cap \mathsf{C}_{\mathtt{int}}(\mathcal{Q}') = \emptyset$ and $\mathsf{C}_{\mathtt{int}}(\mathcal{P}') \cap \mathsf{C}(\mathcal{Q}') = \emptyset$. The intuition is that $\mathcal{P}$ and $\mathcal{Q}$ are different processes (machines) which communicate via their external channels only as explained in Section 2.1 and 3. They should not interfere on their internal channels. It may help to think of $\mathcal{P}$ and $\mathcal{Q}$ to be in SMNF (with different high channels). By Lemma 3 this is w.l.o.g. The definition of $\upharpoonright$ is generalized to sequential process expressions $\mathcal{P}_1, \ldots, \mathcal{P}_n$ in the obvious way. To really match the intuition that different processes communicate, we introduce what we call valid process expressions.

A sequential process expression $\mathcal{P}$ is *valid* for a sequential process expression $\mathcal{Q}$ if

$$\mathsf{C}_{\mathtt{ext}}(\mathcal{P}) \cap \mathsf{C}_{\mathtt{ext}}(\mathcal{Q}) \subseteq \bigcup_{x \in \{\mathtt{in},\mathtt{out}\}, y \in \{\mathtt{net},\mathtt{io}\}} (\mathsf{C}^{y}_{x}(\mathcal{P}) \cap \mathsf{C}^{y}_{\overline{x}}(\mathcal{Q}))$$

where $\overline{x} = \mathtt{in}$ if $x = \mathtt{out}$, and vice versa. That is, external channels used both in $\mathcal{P}$ and $\mathcal{Q}$ are of the same "type" w.r.t. being network or IO channels and are of opposite "types" w.r.t. being input or output channels. Note that being valid is a symmetric relation. Given a set $\mathbf{Z}$ of sequential process expressions, we denote by $\mathbf{Z}$-Valid($\mathcal{Q}$) the set of all process expressions in $\mathbf{Z}$ valid for $\mathcal{Q}$.

We call $\mathcal{P}$ an *adversarial (adversarially valid) process expression* for $\mathcal{Q}$ if $\mathcal{P}$ is valid for $\mathcal{Q}$ and $\mathsf{C}_{\mathtt{ext}}(\mathcal{P}) \cap \mathsf{C}^{\mathtt{io}}_{\mathtt{ext}}(\mathcal{Q}) = \emptyset$. In other words, an adversarial process expression never connects to another process expression via the IO channels. By $\mathbf{Z}$-Adv($\mathcal{Q}$) we denote the set of sequential process expressions in $\mathbf{Z}$ adversarially valid for $\mathcal{Q}$. We write $\mathbf{Z}$-Adv$_{\mathcal{P}}$($\mathcal{Q}$) to denote the set of all process expressions $\mathcal{P}' \in \mathbf{Z}$-Adv($\mathcal{Q}$) such that $\mathcal{P}' \upharpoonright \mathcal{Q}$ and $\mathcal{P}$ are compatible.

We say that $\mathcal{P}$ is an *environmental (environmentally valid) process expression* for $\mathcal{Q}$ if $\mathcal{P}$ is valid for $\mathcal{Q}$ and $\mathsf{C}_{\mathtt{ext}}(\mathcal{P}) \cap \mathsf{C}^{\mathtt{net}}_{\mathtt{ext}}(\mathcal{Q}) = \emptyset$. That is, an environmental process expression never connects to the network channels of a process expression. By $\mathbf{Z}$-Env($\mathcal{Q}$) we denote the set of all process expressions in $\mathbf{Z}$ environmentally valid for $\mathcal{Q}$.

We call a sequential process expression $\mathcal{P}$

- *regular* if $\mathtt{start}, \mathtt{decision} \notin \mathsf{C}(\mathcal{P})$,

- *master* if $\texttt{decision} \notin \mathsf{C}(\mathcal{P})$,
- *decision* if $\texttt{start} \notin \mathsf{C}(\mathcal{P})$, and
- *master decision* otherwise.

In what follows, by **R**, **M**, **D**, and **MD** we denote the set of all closed regular process expressions, closed master process expressions, (open/closed) decision process expressions, and (open/closed) master decision process expressions, respectively. In those cases where it is relevant to distinguish between open and closed process expressions, we explicitly specify **D** and **MD**.

Example: To illustrate the above, let us look at the following typical configuration: $\mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P}$ where $\mathcal{A}$ is adversarially valid for $\mathcal{P}$ and $\mathcal{E}$ is environmentally valid for $\mathcal{A} \upharpoonright \mathcal{P}$. That is, $\mathcal{A}$ only connects to (not necessarily all) network channels of $\mathcal{P}$ and $\mathcal{E}$ only connects to (not necessarily all) IO channels of $\mathcal{P}$ and $\mathcal{A}$. Intuitively $\mathcal{E}$ is the environment process, $\mathcal{A}$ is the adversarial process, and $\mathcal{P}$ is the real protocol. Moreover, let us assume that $\mathcal{E} \in \mathbf{MD}$, i.e., $\mathcal{E}$ is a master decision process. It is helpful to think of $\mathcal{E}$ and $\mathcal{A}$ to be in SMNF. Often, $\mathcal{P}$ is the parallel composition $\mathcal{P}_1 \upharpoonright \cdots \upharpoonright \mathcal{P}_n$ describing n parties (or machines) running a particular protocol.[1] Again, one can think of every $\mathcal{P}_i$ to be in SMNF. (One could as well consider $\mathcal{P}$ to be in SMNF.)

Let us look at a run of such a system. As $\mathcal{E}$ is the master process and initially no other communication is possible (since no other process has produced output yet), $\mathcal{E}$ can read a message on the start channel.[2] After some computation, $\mathcal{E}$ will typically write a message on one of its external channels c (and thus, IO channels[3]) to $\mathcal{A}$ or $\mathcal{P}$, say to $\mathcal{P}$. Then, $\mathcal{E}$ will have to wait for new input and by reading the message on c, $\mathcal{P}$ will be activated next, in case the guard on c of $\mathcal{P}$, more precisely one of the subprocesses $\mathcal{P}_i$ of $\mathcal{P}$, accepts the message. (Otherwise, $\mathcal{E}$ as the master process will be activated again via the `start` channel.) After some computation, $\mathcal{P}$ will typically send a message on some external channel say on a network channel to $\mathcal{A}$, which activates $\mathcal{A}$, and $\mathcal{P}$ has to wait for new input, and so on.

4.2 Indistinguishability of Process Expressions

The indistinguishability of process expressions is defined as expected:

Definition 5. *Two sequential process expressions $\mathcal{P}(\vec{x})$ and $\mathcal{Q}(\vec{x})$ are called* equivalent *or* indistinguishable *($\mathcal{P}(\vec{x}) \equiv \mathcal{Q}(\vec{x})$) iff for every polynomial $p(\mathbf{n})$ there exists i_0 such that for every $i \geq i_0$ and every tuple $\vec{a}$ of bit strings we have that*

$$|\texttt{Prob}[\mathcal{P}(\vec{a})^{\mathbf{n}\leftarrow i} \rightsquigarrow 1] - \texttt{Prob}[\mathcal{Q}(\vec{a})^{\mathbf{n}\leftarrow i} \rightsquigarrow 1]| \leq 1/p(i).$$

Obviously, $\equiv$ is an equivalence relation on sequential process expressions.

[1] As mentioned, by using bounded replication it is also possible to specify protocols with a polynomial number of parties.

[2] In case $\mathcal{E}$ does not read a message on this channel or the guard rejects the message, nothing will happen, and the run is completed.

[3] $\mathcal{E}$ could as well write a message on a channel that is declared to be a network channel. However, this channel will not be connected to $\mathcal{A}$ or $\mathcal{P}$ since $\mathcal{E}$ may only connect to IO channels of these processes.

4.3 Security Notions

We now define the various security notions. For strong simulatability as well as strong and weak blackbox simulatability we define two variants depending on whether or not the simulator may play the role of the master process.

The first definition will be used if the simulator is a regular process, and thus, not a master.

Definition 6. *Let* $\mathbf{A}$ *(adversaries),* $\mathbf{I}$ *(ideal adversaries),* $\mathbf{E}$ *(environments), and* $\mathbf{S}$ *(simulators) be sets of sequential process expressions, and* $\mathcal{P}$ *(the real protocol) and* $\mathcal{F}$ *(the ideal functionality) be IO-compatible sequential process expressions.*

Strong Simulatability: $\texttt{SS}_{(\mathbf{S},\mathbf{E})}(\mathcal{P},\mathcal{F})$ *iff* $\mathcal{P}$ *and* $\mathcal{F}$ *are IO-compatible and there exists* $\mathcal{S} \in \mathbf{S}\text{-}Adv_{\mathcal{P}}(\mathcal{F})$ *(called* simulator*) such that* $\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}$ *for every* $\mathcal{E} \in \mathbf{E}\text{-}Valid(\mathcal{P})$ *(called* environment*).*

Strong Blackbox Simulatability: $\texttt{SBB}_{(\mathbf{A},\mathbf{S},\mathbf{E})}(\mathcal{P},\mathcal{F})$ *iff* $\mathcal{P}$ *and* $\mathcal{F}$ *are IO-compatible and there exists* $\mathcal{S} \in \mathbf{S}\text{-}Adv_{\mathcal{P}}(\mathcal{F})$ *(called* simulator*) such that* $\mathcal{E} \mid \mathcal{A} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{A} \mid \mathcal{S} \mid \mathcal{F}$ *for every* $\mathcal{A} \in \mathbf{A}\text{-}Adv(\mathcal{P})$ *(called* adversary*) and* $\mathcal{E} \in \mathbf{E}\text{-}Env(\mathcal{A} \mid \mathcal{P})$ *(called* environment*).*

Weak Blackbox Simulatability: $\texttt{WBB}_{(\mathbf{A},\mathbf{S},\mathbf{E})}(\mathcal{P},\mathcal{F})$ *iff* $\mathcal{P}$ *and* $\mathcal{F}$ *are IO-compatible and for every* $\mathcal{A} \in \mathbf{A}\text{-}Adv(\mathcal{P})$ *(called* adversary*) there exists* $\mathcal{S} \in \mathbf{S}\text{-}Adv_{\mathcal{P}}(\mathcal{F})$ *(called* simulator*) such that* $\mathcal{E} \mid \mathcal{A} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{A} \mid \mathcal{S} \mid \mathcal{F}$ *for every* $\mathcal{E} \in \mathbf{E}\text{-}Env(\mathcal{A} \mid \mathcal{P})$ *(called* environment*). As we will see, equivalently we can require that the simulator* $\mathcal{S}$ *may only depend on the communication size of* $\mathcal{A}$ *instead of* $\mathcal{A}$ *itself. In particular, all our results hold for both variants, and we therefore do not distinguish between them explicitly. Necessary adjustments in proofs will be pointed out.*

Universal Composability: $\texttt{UC}_{(\mathbf{A},\mathbf{I},\mathbf{E})}(\mathcal{P},\mathcal{F})$ *iff* $\mathcal{P}$ *and* $\mathcal{F}$ *are IO-compatible and for every* $\mathcal{A} \in \mathbf{A}\text{-}Adv(\mathcal{P})$ *(called* real adversary*) there exists* $\mathcal{I} \in \mathbf{I}\text{-}Adv_{\mathcal{A} \mid \mathcal{P}}(\mathcal{F})$ *(called* ideal adversary*) such that* $\mathcal{E} \mid \mathcal{A} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{I} \mid \mathcal{F}$ *for every* $\mathcal{E} \in \mathbf{E}\text{-}Env(\mathcal{A} \mid \mathcal{P})$ *(called* environment*).*

In Section 5, we will consider a variety of different instances of the above security notions by defining the sets **A**, **I**, **S**, and **E** to be one of the sets **R**, **M**, **D**, and **MD**. One such instance is $\texttt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$: Note that with $\mathcal{A} \in \mathbf{M}\text{-Adv}(\mathcal{P})$, $\mathcal{I} \in \mathbf{M}\text{-Adv}_{\mathcal{A} \mid \mathcal{P}}(\mathcal{F})$, and $\mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{A} \mid \mathcal{P})$ as required by $\texttt{UC}_{(\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ it follows that if $\mathcal{A}$ is a master process expression, i.e., contains `start`, then $\mathcal{E}$ is not allowed to contain `start`, i.e., $\mathcal{E}$ is just a decision process expression, since otherwise $\mathcal{E}$ would not be environmentally valid for $\mathcal{A}$. Conversely, if $\mathcal{A}$ is regular, i.e., does not contain `start`, then $\mathcal{E}$ may contain `start`. In other words, it is guaranteed that not both, the adversary and the environment, are master process expressions at the same time. Only at most one of them has this role.

We now define versions of strong simulatability as well as strong and weak blackbox simulatability for the case that the simulator may play the role of the master process. These definitions also apply for the case where the simulator is restricted to be a regular process, and in this case they coincide with the definitions given above. Hence, it would have been enough to provide only one definition. However, since the previous definitions are simpler, we separate the two cases.

Definition 7. ***Strong Simulatability:*** $\texttt{SSsim}_{(\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff the following conditions are satisfied:*

- $\mathcal{P}$ *and* $\mathcal{F}$ *are IO-compatible,*
- *there exists* $\mathcal{S} \in \mathbf{M}$ *adversarially valid for* $\mathcal{F}$ *and such that* $\mathcal{P}$ *and* $\mathcal{S} \upharpoonright \mathcal{F}$ *are compatible except that, in addition to the external channels in* $\mathcal{P}$, $\mathcal{S}$ *may contain* `start` *as input IO channel and if* `start` *occurs in* $\mathcal{S}$, *then* `start`$'$ *may also occur in* $\mathcal{S}$ *as a new output IO channel, and*
- $\mathcal{E} \upharpoonright \mathcal{P} \equiv \mathcal{E}' \upharpoonright \mathcal{S} \upharpoonright \mathcal{F}$ *for every* $\mathcal{E} \in \mathbf{MD}$-*Valid*$(\mathcal{P})$ *where* $\mathcal{E}' = \mathcal{E}$ *if* `start` *does not occur in* $\mathcal{S}$ *and where* $\mathcal{E}'$ *is obtained from* $\mathcal{E}$ *by replacing every occurrence of* `start` *by* `start`$'$ *otherwise.*

Strong Blackbox Simulatability: $\mathtt{SBBsim}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff the following conditions are satisfied:*

- $\mathcal{P}$ *and* $\mathcal{F}$ *are IO-compatible,*
- *there exists* $\mathcal{S} \in \mathbf{M}$ *adversarially valid for* $\mathcal{F}$ *and such that* $\mathcal{P}$ *and* $\mathcal{S} \upharpoonright \mathcal{F}$ *are compatible except that, in addition to the external channels in* $\mathcal{P}$, $\mathcal{S}$ *may contain* `start` *as input IO channel and if* `start` *occurs in* $\mathcal{S}$, *then* `start`$'$ *may also occur in* $\mathcal{S}$ *as a new output IO channel, and*
- $\mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E}' \upharpoonright \mathcal{A}' \upharpoonright \mathcal{S} \upharpoonright \mathcal{F}$ *for every* $\mathcal{A} \in \mathbf{M}$-*Adv*$(\mathcal{P})$ *and* $\mathcal{E} \in \mathbf{MD}$-*Env*$(\mathcal{A} \upharpoonright \mathcal{P})$ *where* $\mathcal{E}' = \mathcal{E}$ *and* $\mathcal{A}' = \mathcal{A}$ *if* `start` *does not occur in* $\mathcal{S}$ *and where* $\mathcal{E}'$ *and* $\mathcal{A}'$ *are obtained from* $\mathcal{E}$ *and* $\mathcal{A}$, *respectively, by replacing every occurrence of* `start` *by* `start`$'$ *otherwise.*[4]

Weak Blackbox Simulatability: $\mathtt{WBBsim}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff the following conditions are satisfied:*

- $\mathcal{P}$ *and* $\mathcal{F}$ *are IO-compatible,*
- *for every* $\mathcal{A} \in \mathbf{M}$-*Adv*$(\mathcal{P})$ *there exists* $\mathcal{S} \in \mathbf{M}$ *adversarially valid for* $\mathcal{F}$ *and such that* $\mathcal{P}$ *and* $\mathcal{S} \upharpoonright \mathcal{F}$ *are compatible except that, in addition to the external channels in* $\mathcal{P}$, $\mathcal{S}$ *may contain* `start` *as input IO channel and if* `start` *occurs in* $\mathcal{S}$, *then* `start`$'$ *may also occur in* $\mathcal{S}$ *as a new output IO channel, and*
- $\mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E}' \upharpoonright \mathcal{A}' \upharpoonright \mathcal{S} \upharpoonright \mathcal{F}$ *for every* $\mathcal{E} \in \mathbf{MD}$-*Env*$(\mathcal{A} \upharpoonright \mathcal{P})$ *where* $\mathcal{E}' = \mathcal{E}$ *and* $\mathcal{A}' = \mathcal{A}$ *if* `start` *does not occur in* $\mathcal{S}$ *and where* $\mathcal{E}'$ *and* $\mathcal{A}'$ *are obtained from* $\mathcal{E}$ *and* $\mathcal{A}$, *respectively, by replacing every occurrence of* `start` *by* `start`$'$ *otherwise.*

In the above definition of WBBsim, we could also restrict $\mathcal{S}$ *to be master only if* $\mathcal{A}$ *is master, i.e.,* $\mathcal{S}$ *may contain* `start` *only if* $\mathcal{A}$ *does. In this case, we can always replace* $\mathcal{E}'$ *by* $\mathcal{E}$. *Also, just as for WBB, we can consider a version of WBBsim where the simulator may only depend on the communication size of* $\mathcal{A}$. *As we will see, all (four) variants are equivalent, and we will not distinguish between them explicitly. However, we point out necessary modifications in proofs.*

Variants of SSsim, SBBsim, and WBBsim such as $\mathtt{SSsim}_{(\mathbf{M}\setminus\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$, $\mathtt{SBBsim}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$, *and* $\mathtt{WBBsim}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *are defined in the obvious way.*

[4] Note that `start` can not occur both in $\mathcal{E}$ and $\mathcal{A}$ since otherwise $\mathcal{E}$ would not be environmentally valid for $\mathcal{A} \upharpoonright \mathcal{P}$.

5 Relationships between the Security Notions

In this section, we examine the relationships between the security notions introduced in the previous section. We prove some expected equivalences between the security notions and observe some surprising differences which would not be apparent without detailed analysis. The proofs are carried out axiomatically. The axiom system is introduced in Section 5.1. We show that all of the axioms are sound, i.e., are satisfied in SPPC, except for one axiom, which is called FORWARDER. In Section 5.2, we show that FORWARDER is a *necessary* condition on protocols in order for universal composability to imply black-box simulatability or strong simulatability, i.e., if this axiom does not hold, then universal composability does not imply black-box simulatability (strong simulatability). In Section 5.3, we compare the security notions for the cases where the environment may play the role of the master decision process, and in Section 5.4 we restrict the environment to be a decision process. In Section 5.5, we study variants of strong and blackbox simulatability where the simulators may be master processes. All results are summarized in Section 5.6. Based on the FORWARDER axiom, we obtain a complete characterization of the conditions under which universal composability and blackbox simulatability are equivalent. Since the proofs are carried out axiomatically, this enables us to carry over our results to other computational models [9,27] (Section 6). These axioms can also serve as an abstract specification for a "reasonable" computational model for simulation-based security. Furthermore, the counter-examples used to demonstrate that certain notions are strictly stronger than others are quite simple and easily translate into the related models (Section 6).

5.1 The Axiom System

To define the axioms (equational principles), we first introduce four variants of so-called *dummy adversaries*, which simply forward messages on network channels of protocols and are used to rename network channels and turn network channels into IO channels.

The first dummy adversary, called regular network dummy adversary, simply forwards messages received on channel $c \in \mathsf{C}_{\mathtt{out}}$ to a copy c' of this channel and messages received on channel c' for a copy of a channel $c \in \mathsf{C}_{\mathtt{in}}$ to channel c. The number of messages this dummy can forward is bounded by a polynomial in the security parameter. Formally, a *regular network dummy adversary* is defined as follows:

$$\begin{aligned} \mathcal{D}_R^{\mathtt{net}}(\mathsf{C}_{\mathtt{in}}, \mathsf{C}_{\mathtt{out}}, q(\mathbf{n})) &= \mathcal{D}_R^{\mathtt{net}} \\ &= !_{q(\mathbf{n})} \left(\sum_{c \in \mathsf{C}_{\mathtt{out}}} \mathtt{in}(c,x).\mathtt{out}(c',x) + \right. \\ &\qquad \left. \sum_{c \in \mathsf{C}_{\mathtt{in}}} \mathtt{in}(c',x).\mathtt{out}(c,x) \right) \end{aligned} \tag{2}$$

where $\mathsf{C}_{\mathtt{in}}$ and $\mathsf{C}_{\mathtt{out}}$ are disjoint and finite sets of channel names, c' is a new copy of c, i.e., it has a new name, and $q(\mathbf{n})$ is a polynomial in $\mathbf{n}$. All channels in $\mathcal{D}_R^{\mathtt{net}}$ are considered network channels. Note that $\mathcal{D}_R^{\mathtt{net}} \in \mathbf{R}$.

The *regular IO dummy adversary* $\mathcal{D}_R^{\mathtt{io}}(\mathsf{C}_{\mathtt{in}}, \mathsf{C}_{\mathtt{out}}, q(\mathbf{n}))$ is defined just as $\mathcal{D}_R^{\mathtt{net}}(\mathsf{C}_{\mathtt{in}}, \mathsf{C}_{\mathtt{out}}, q(\mathbf{n}))$ except that the channels c' are declared to be IO channels. Again, we have that $\mathcal{D}_R^{\mathtt{io}}(\mathcal{P}) \in \mathbf{R}$.

The following two dummy adversaries are master process expressions. The first one is called *master network dummy adversary* and it works just as the regular network dummy adversary except that it also forwards messages received on $\mathtt{start}$ to $\mathtt{start}'$. More formally, a master network dummy adversary is defined as follows:

$$\begin{aligned}
\mathcal{D}_M^{\mathtt{net}}(\mathsf{C}_{\mathtt{in}}, \mathsf{C}_{\mathtt{out}}, q(\mathbf{n})) &= \mathcal{D}_M^{\mathtt{net}} \\
&= !_{q(\mathbf{n})} \Bigg(\sum_{c \in \mathsf{C}_{\mathtt{out}}} \mathtt{in}(c, x).\mathtt{out}(c', x) + \\
&\qquad \sum_{c \in \mathsf{C}_{\mathtt{in}}} \mathtt{in}(c', x).\mathtt{out}(c, x) + \\
&\qquad \mathtt{in}(\mathtt{start}, x).\mathtt{out}(\mathtt{start}', x) \Bigg)
\end{aligned} \tag{3}$$

where $\mathsf{C}_{\mathtt{in}}$, $\mathsf{C}_{\mathtt{out}}$, $q(\mathbf{n})$, and the channels c' are defined just as in $\mathcal{D}_R^{\mathtt{net}}(\mathsf{C}_{\mathtt{in}}, \mathsf{C}_{\mathtt{out}}, q(\mathbf{n}))$. The channel $\mathtt{start}'$ is a new channel and declared to be an IO channel.

The *master IO dummy adversary* $\mathcal{D}_M^{\mathtt{io}}(\mathsf{C}_{\mathtt{in}}, \mathsf{C}_{\mathtt{out}}, q(\mathbf{n}))$ is defined just as $\mathcal{D}_M^{\mathtt{net}}(\mathsf{C}_{\mathtt{in}}, \mathsf{C}_{\mathtt{out}}, q(\mathbf{n}))$ except that the channels c' are declared to be IO channels.

Now we are ready to state the axioms and equational principles we use. Further explanations follow below.

COM. For all sequential process expressions $\mathcal{P}$ and $\mathcal{Q}$:

$$\mathcal{P} \mid \mathcal{Q} \equiv \mathcal{Q} \mid \mathcal{P}.$$

ASC. For all sequential process expressions $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{R}$:

$$\mathcal{P} \mid (\mathcal{Q} \mid \mathcal{R}) \equiv (\mathcal{P} \mid \mathcal{Q}) \mid \mathcal{R}.$$

TRN. For all sequential process expression $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{R}$:

$$\mathcal{P} \equiv \mathcal{Q}, \mathcal{Q} \equiv \mathcal{R} \implies \mathcal{P} \equiv \mathcal{R}.$$

SYM. For all sequential process expressions $\mathcal{P}$ and $\mathcal{Q}$:

$$\mathcal{P} \equiv \mathcal{Q} \implies \mathcal{Q} \equiv \mathcal{P}.$$

RENAME. For sequential process expressions $\mathcal{P}_1, \ldots, \mathcal{P}_k$ such that $\mathcal{P}_i$ is valid for $\mathcal{P}_{i+1} \mid \ldots \mid \mathcal{P}_k$ for every i:

$$\mathcal{P}_1 \mid \ldots \mid \mathcal{P}_k \equiv \mathcal{P}_1' \mid \ldots \mid \mathcal{P}_k'$$

where the $\mathcal{P}_i'$ are derived from $\mathcal{P}_i$ by consistently (w.r.t. the other $\mathcal{P}_j'$) renaming external channels ($\mathtt{start}$ and $\mathtt{decision}$ may not be renamed) and changing network channels to IO channels and vice versa.

RENAME-START

$$\mathcal{E} \mid \mathcal{A} \equiv \mathcal{E} \mid !_{q(\mathbf{n})}\ \mathtt{in}(\mathtt{start}, \varepsilon).\mathtt{out}(\mathtt{start}', \varepsilon) \mid \mathcal{A}'$$

for every $\mathcal{A} \in \mathbf{M}$, $\mathcal{E} \in \mathbf{D}\text{-Valid}(\mathcal{A})$, and $q(\mathbf{n}) \geq \mathtt{comsize}(\mathcal{A})(\mathbf{n})$ where $\mathcal{A}'$ is obtained from $\mathcal{A}$ by replacing every occurrence of $\mathtt{start}$ by the new channel $\mathtt{start}'$.

REG-S-FORWARDER.

$$\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E}' \mid \mathcal{D}_R^{\mathtt{net}} \mid \mathcal{P}$$

for every $\mathcal{P} \in \mathbf{R}$, $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$, and $q(\mathbf{n}) \geq \mathtt{comsize}(\mathcal{P})(\mathbf{n}) + \mathtt{comsize}(\mathcal{E})(\mathbf{n})$ such that $\mathcal{D}_R^{\mathtt{net}} = \mathcal{D}_R^{\mathtt{net}}(\mathsf{C}_{\mathtt{in}}^{\mathtt{net}}(\mathcal{P}), \mathsf{C}_{\mathtt{out}}^{\mathtt{net}}(\mathcal{P}), q(\mathbf{n}))$ and $\mathcal{E}'$ is obtained from $\mathcal{E}$ by replacing every occurrence of network channels c of $\mathcal{P}$ by c' as in the definition of $\mathcal{D}_R^{\mathtt{net}}$.

REG-ADV-FORWARDER.

$$\mathcal{E} \mid \mathcal{A} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{A}' \mid \mathcal{D}_R^{\mathtt{net}} \mid \mathcal{P}$$

for every $\mathcal{P} \in \mathbf{R}$, $\mathcal{A} \in \mathbf{M}\text{-Adv}(\mathcal{P})$, $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{A} \mid \mathcal{P})$, and $q(\mathbf{n}) \geq \mathtt{comsize}(\mathcal{P})(\mathbf{n}) + \mathtt{comsize}(\mathcal{A})(\mathbf{n})$ such that $\mathcal{D}_R^{\mathtt{net}} = \mathcal{D}_R^{\mathtt{net}}(\mathsf{C}_{\mathtt{in}}^{\mathtt{net}}(\mathcal{P}), \mathsf{C}_{\mathtt{out}}^{\mathtt{net}}(\mathcal{P}), q(\mathbf{n}))$ and $\mathcal{A}'$ is obtained from $\mathcal{A}$ by replacing every occurrence of network channels c of $\mathcal{P}$ by c' as in the definition of $\mathcal{D}_R^{\mathtt{net}}$.

MASTER-S-FORWARDER.

$$\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E}'_M \mid \mathcal{D}_M^{\mathtt{net}} \mid \mathcal{P}$$

for every $\mathcal{P} \in \mathbf{R}$, $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$, and $q(\mathbf{n}) > \mathtt{comsize}(\mathcal{P})(\mathbf{n}) + \mathtt{comsize}(\mathcal{E})(\mathbf{n})$ such that $\mathcal{D}_M^{\mathtt{net}} = \mathcal{D}_M^{\mathtt{net}}(\mathsf{C}_{\mathtt{in}}^{\mathtt{net}}(\mathcal{P}), \mathsf{C}_{\mathtt{out}}^{\mathtt{net}}(\mathcal{P}), q(\mathbf{n}))$ and $\mathcal{E}'_M$ is obtained from $\mathcal{E}$ by replacing every occurrence of network channels c of $\mathcal{P}$ by c' as in the definition of $\mathcal{D}_R^{\mathtt{net}}$ and replacing every occurrence of $\mathtt{start}$ in $\mathcal{E}$ (if any) by $\mathtt{start}'$.

MASTER-ADV-FORWARDER.

$$\mathcal{E} \mid \mathcal{A} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{A}'_M \mid \mathcal{D}_M^{\mathtt{net}} \mid \mathcal{P}$$

for every $\mathcal{P} \in \mathbf{R}$, $\mathcal{A} \in \mathbf{M}\text{-Adv}(\mathcal{P})$, $\mathcal{E} \in \mathbf{D}\text{-Valid}(\mathcal{A} \mid \mathcal{P})$, and $q(\mathbf{n}) > \mathtt{comsize}(\mathcal{P})(\mathbf{n}) + \mathtt{comsize}(\mathcal{A})(\mathbf{n})$ such that $\mathcal{D}_M^{\mathtt{net}} = \mathcal{D}_M^{\mathtt{net}}(\mathsf{C}_{\mathtt{in}}^{\mathtt{net}}(\mathcal{P}), \mathsf{C}_{\mathtt{out}}^{\mathtt{net}}(\mathcal{P}), q(\mathbf{n}))$ and $\mathcal{A}'_M$ is obtained from $\mathcal{A}$ by replacing every occurrence of network channels c of $\mathcal{P}$ by c' as in the definition of $\mathcal{D}_R^{\mathtt{net}}$ and replacing every occurrence of $\mathtt{start}$ in $\mathcal{A}$ (if any) by $\mathtt{start}'$.

FORWARDER($\mathcal{P}$). There exists $\mathcal{D} \in \mathbf{R}\text{-Adv}(\mathcal{P})$ such that $\mathsf{C}_{\mathtt{ext}}^{\mathtt{io}}(\mathcal{D}) = \emptyset$, $\mathsf{C}_{\mathtt{in}}^{\mathtt{net}}(\mathcal{D}) = \mathsf{C}_{\mathtt{out}}^{\mathtt{net}}(\mathcal{P}) \cup \mathsf{C}_{\mathtt{in}}^{\mathtt{net}}(\mathcal{P})'$, $\mathsf{C}_{\mathtt{out}}^{\mathtt{net}}(\mathcal{D}) = \mathsf{C}_{\mathtt{in}}^{\mathtt{net}}(\mathcal{P}) \cup \mathsf{C}_{\mathtt{out}}^{\mathtt{net}}(\mathcal{P})'$, and for every $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$:

$$\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E}' \mid \mathcal{D} \mid \mathcal{P},$$

where $\mathsf{C}_{\mathtt{in}}^{\mathtt{net}}(\mathcal{P})'$ consists of new copies c' of the channels in $\mathsf{C}_{\mathtt{in}}^{\mathtt{net}}(\mathcal{P})$, $\mathsf{C}_{\mathtt{out}}^{\mathtt{net}}(\mathcal{P})'$ consists of new copies c' of the channels in $\mathsf{C}_{\mathtt{out}}^{\mathtt{net}}(\mathcal{P})$, and $\mathcal{E}'$ is obtained from $\mathcal{E}$ by replacing every occurrence of network channels c of $\mathcal{P}$ by c'.

MMD-INCLUSION. For every $\mathcal{P} \in \mathbf{R}$:

$$\forall \mathcal{A} \in \mathbf{M}\text{-Adv}(\mathcal{P}). \forall \mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{A} \mid \mathcal{P}) : \mathcal{E} \mid \mathcal{A} \in \mathbf{MD}\text{-Valid}(\mathcal{P}).$$

MD-INCLUSION. For every $\mathcal{P} \in \mathbf{R}$:

$$\forall \mathcal{A} \in \mathbf{M}\text{-Adv}(\mathcal{P}). \forall \mathcal{E} \in \mathbf{D}\text{-Env}(\mathcal{A} \mid \mathcal{P}) : \mathcal{E} \mid \mathcal{A} \in \mathbf{MD}\text{-Valid}(\mathcal{P}).$$

The letters 'S' in the axioms REG-S-FORWARDER and MASTER-S-FORWARDER are reminiscent of "simple". All axioms, except for the last two, are equational principals on process expressions. The last two axioms allow us to combine a environment which may only access the IO channels of a protocol and an adversary, which may only access the network channels of a protocol, into an environment that may access both the IO and network channels of a protocol.

The following lemma states that all axioms, except for FORWARDER, are sound, i.e., are true in SPPC. These axioms are called *basic axioms* as they should be satisfied in most computational models for simulation-based security notions (see Section 6). As we will see, most of the relationships between the security notions only require the basic axioms to hold. We will only need FORWARDER to show that universal composability implies black-box simulatability (or equivalently, strong simulatability). The axiom FORWARDER is further discussed in Section 5.2.

Lemma 8. *All axioms mentioned above, except for FORWARDER, are sound, i.e., hold for SPPC. This is also true for the guard-free fragment of SPPC.*

Proof. The proofs of the properties COM, ASC, TRN, SYM are trivial. The property RENAME is also obvious as the semantics of process expressions does not depend on the names of channels and whether they are network or IO channels.

In RENAME-START, the additional process expression on the right hand-side simply forwards the message on $\mathtt{start}$ (which by the definition of the semantics is always ε) via $\mathtt{start}'$ to $\mathcal{A}'$. Since $q(\mathbf{n}) \geq \mathtt{comsize}(\mathcal{A}) = \mathtt{comsize}(\mathcal{A}')$, this process does not terminate before $\mathcal{A}$ ($\mathcal{A}'$) does, and thus, the signal on $\mathtt{start}$ is always forwarded.

The property MASTER-S-FORWARDER easily follows from the following observations: First note that $\mathcal{E}$ and $\mathcal{E}'_M$ behave exactly the same except that if $\mathcal{E}$ outputs/inputs a message on the (low) channel $c \in \mathsf{C}^{\mathtt{net}}_{\mathtt{ext}}(\mathcal{P})$, then $\mathcal{E}'_M$ outputs/inputs a message on the (low) channel c'. Since these channels are low channels and since by the definition of sequential process expressions at any time there is at most one message on a low channel, messages between $\mathcal{E}'_M$ and $\mathcal{P}$ are immediately forwarded by $\mathcal{D}^{\mathtt{net}}_M$. Second, if $\mathcal{E} \mid \mathcal{P}$, and more precisely $\mathcal{E}$, receives a message on $\mathtt{start}$, then in $\mathcal{E}'_M \mid \mathcal{D}^{\mathtt{net}}_M \mid \mathcal{P}$ the dummy adversary $\mathcal{D}^{\mathtt{net}}_M$ receives a message on $\mathtt{start}$ and by definition immediately forwards it to $\mathcal{E}'_M$ on $\mathtt{start}'$. Then, $\mathcal{E}'_M$ behaves exactly as $\mathcal{E}$. Third, note that if $\mathcal{D}^{\mathtt{net}}_M$ receives a message, then this message comes from $\mathcal{E}'_M$, $\mathcal{P}$, or $\mathtt{start}$. If the message comes from $\mathtt{start}$ and $\mathcal{D}^{\mathtt{net}}_M$ forwards this message on $\mathtt{start}'$ but $\mathcal{E}'_M$ does not read the message, then the overall system stops. Thus, this can happen only ones, and hence, in all other cases if $\mathcal{D}^{\mathtt{net}}_M$ receives a message, the communication size of $\mathcal{E}'_M$ or $\mathcal{P}$ decreases (if the message was sent on $\mathtt{start}$, then the communication size of $\mathcal{E}'_M$ will decrease when reading $\mathtt{start}'$). By the definition of $q(\mathbf{n})$ it is therefore guaranteed that $\mathcal{D}^{\mathtt{net}}_M$ will not terminate before $\mathcal{E}'$ and $\mathcal{P}$ do. The argument for REG-S-FORWARDER is similar.

For MASTER-ADV-FORWARDER note that if MASTER-ADV-FORWARDER receives a message, then this message must have been received from $\mathcal{A}'_M$, $\mathcal{P}$, or $\mathtt{start}$. Now, the argument is analogous to the one for MASTER-S-FORWARDER; similarly for REG-ADV-FORWARDER.

It is clear that MMD-INCLUSION and MD-INCLUSION are sound. The arguments for the guard-free fragment of SPPC are the same. □

Remark 9. *In MASTER-S-FORWARDER it does not suffice to define $q(\mathbf{n})$ independently of* $\mathtt{comsize}(\mathcal{E})$. *If $q(\mathbf{n}) = \mathtt{comsize}(\mathcal{P})(\mathbf{n})$, then consider for instance the environment $\mathcal{E}$ ($\mathcal{E}'_M$) which triggers $\mathcal{D}^{\mathtt{net}}_M$ via* $\mathtt{start}$ *$q(\mathbf{n}) + 1$ times. When interacting with $\mathcal{D}^{\mathtt{net}}_M$, $\mathcal{E}'_M$ will not be triggered via*

$\mathtt{start}'$ *after the* $q(\mathbf{n})+1$*st time, however,* $\mathcal{E}$ *would be triggered via* $\mathtt{start}$. *Thus,* $\mathcal{E}$ $(\mathcal{E}'_M)$ *can "observe"* $\mathcal{D}^{\mathtt{net}}_M$ *(see also the proof of Theorem 21).*

Similarly, in REG-S-FORWARDER it does not suffice to define $q(\mathbf{n}) = \mathtt{comsize}(\mathcal{P})(\mathbf{n})$ *independently of* $\mathtt{comsize}(\mathcal{E})$. *Assume, for example, that* $\mathcal{P}$ *is of the form* $\mathcal{P}_1 \upharpoonright \cdots \upharpoonright \mathcal{P}_k$ *where the* $\mathcal{P}_i$ *model parties running a certain protocol. One of the parties may terminate before others do. Now, if* $\mathcal{E}$ *sends a message to a terminated party, say* $\mathcal{P}_i$, *then* $\mathcal{P}$ *will not consume resources as the message is simply ignored by* $\mathcal{P}_i$ *and no computation will take place. In particular,* $\mathcal{E}$ *can send an unbounded number of messages to* $\mathcal{P}_i$ *without* $\mathcal{P}$ *consuming any resources. Later,* $\mathcal{E}$ *can send a message to a non-terminated party* $\mathcal{P}_j$ *and will (possibly) obtain an answer. Now, if the dummy* $\mathcal{D}^{\mathtt{net}}_R$ *is plugged in between* $\mathcal{E}$ *and* $\mathcal{P}$, *then* $\mathcal{E}$ *can exhaust* $\mathcal{D}^{\mathtt{net}}_R$, *i.e., force it to terminate, by sending sufficiently many messages through* $\mathcal{D}^{\mathtt{net}}_R$ *to the terminated party. Then, when* $\mathcal{E}$ *sends a message to the non-terminated party,* $\mathcal{D}^{\mathtt{net}}_R$ *does not have any resources left to forward this message, and hence,* $\mathcal{E}$ *can detect the presence of* $\mathcal{D}^{\mathtt{net}}_R$ *because if the* $\mathcal{D}^{\mathtt{net}}_R$ *is present no answer will come back from the non-terminated party and if* $\mathcal{D}^{\mathtt{net}}_R$ *is absent the non-terminated party will receive the message from* $\mathcal{E}$ *and can send a reply.*

5.2 On the Necessity and Validity of FORWARDER

We show that

1. The axiom FORWARDER is necessary for universal composability to imply black-box simulatability and strong simulatability (Section 5.2.1).

2. The axiom FORWARDER is not true for all regular process expressions $\mathcal{P} \in \mathbf{R}$. But there are interesting classes of regular process expressions for which FORWARDER is satisfied (Section 5.2.2).

5.2.1 Necessity of FORWARDER

We show that the axiom FORWARDER is necessary for universal composability to imply black-box simulatability and strong simulatability, respectively.

Let $\mathbf{C}$ be a class of sequential process expressions. We say that $\mathbf{C}$ is *closed under renaming* if $\mathcal{P}' \in \mathbf{C}$ for every $\mathcal{P} \in \mathbf{C}$ where $\mathcal{P}'$ is obtained from $\mathcal{P}$ by renaming channels (this does *not* include turning a network channel into an IO channel or vice versa).

The following theorem is stated for a certain variant of universal composability $(\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F}))$, strong black-box simulatability $(\mathtt{SBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F}))$, and strong simulatability $(\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F}))$. In the following sections, we will identify various variants of universal composability as well as weak black-box simulatability equivalent to $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$. Also, there are various variants of strong black-box simulatability and strong simulatability equivalent to $\mathtt{SBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ and $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$, respectively (see Section 5.6 for an overview). To show these equivalences the following theorem is not needed, and hence, this theorem immediately carries over to other variants of security notions.

Theorem 10. *Let* $\mathbf{C}$ *be a class of sequential process expressions closed under renaming.*

1. *Assume that*
$$\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F}) \Rightarrow \mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$$
for every $\mathcal{P},\mathcal{F} \in \mathbf{C}$. *Then, FORWARDER*$(\mathcal{P})$ *for every* $\mathcal{P} \in \mathbf{C}$.

2. Assume that

$$\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F}) \Rightarrow \mathtt{SBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$$

for every $\mathcal{P},\mathcal{F} \in \mathbf{C}$. Then, FORWARDER($\mathcal{P}$) for every $\mathcal{P} \in \mathbf{C}$.

Proof. We first prove the case for strong simulatability. Let $\mathcal{P} \in C$. Under the given assumptions, we want to show that FORWARDER($\mathcal{P}$).

Let $\mathcal{P}'$ be obtained from $\mathcal{P}$ by consistently renaming the network channels such that the set of network channels of $\mathcal{P}'$ is disjoint from the set of network channels of $\mathcal{P}$. Obviously, we have that $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{P}')$ since given the real adversary the ideal adversary can be obtained from the real adversary by renaming the network channels according to the renaming of network channels of $\mathcal{P}'$. Now, since with $\mathcal{P} \in \mathbf{C}$ we have that $\mathcal{P}' \in \mathbf{C}$ and by the assumption that universal composability implies strong simulatability, we obtain $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{P}')$. Hence, there exists a simulator $\mathcal{S} \in \mathbf{R}\text{-Adv}_{\mathcal{P}}(\mathcal{P}')$ such that

$$\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{P}'$$

for all $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$. Note that $\mathcal{S}$ contains network channels from $\mathcal{P}$ and $\mathcal{P}'$. If c is a network channel in $\mathcal{P}$, then let c' denote the corresponding channel in $\mathcal{P}'$. Let $\mathcal{D}$ be obtained from $\mathcal{S}$ by switching the names of network channels, i.e., rename c to c' and c' to c. Now, it is obvious that

$$\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E}' \mid \mathcal{D} \mid \mathcal{P}.$$

for all $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$ where $\mathcal{E}'$ is defined as in FORWARDER($\mathcal{P}$). Thus, FORWARDER($\mathcal{P}$) is true.

The proof for strong black-box simulatability is similar. Analogously to the previous case we conclude that $\mathtt{SBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{P}')$. Hence, there exists a simulator $\mathcal{S} \in \mathbf{R}\text{-Adv}_{\mathcal{P}}(\mathcal{P}')$ such that

$$\mathcal{E} \mid \mathcal{A} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{A} \mid \mathcal{S} \mid \mathcal{P}'.$$

for all $\mathcal{A} \in \mathbf{R}\text{-Adv}(\mathcal{P})$ and all $\mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{A} \mid \mathcal{P})$. In particular, this is true if $\mathcal{A} = \mathcal{D}_R^{\mathtt{net}}$ as defined in REG-S-FORWARDER where we set $q(\mathbf{n}) = \mathtt{comsize}(\mathcal{P})(\mathbf{n}) + \mathtt{comsize}(\mathcal{S})(\mathbf{n}) + \mathtt{comsize}(\mathcal{E})(\mathbf{n})$. Now, together with REG-S-FORWARDER we obtain:

$$\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E}' \mid \mathcal{D}_R^{\mathtt{net}} \mid \mathcal{P} \equiv \mathcal{E}' \mid \mathcal{D}_R^{\mathtt{net}} \mid \mathcal{S} \mid \mathcal{P}' \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{P}'$$

for every $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$ where $\mathcal{E}'$ obtained from $\mathcal{E}$ as described in REG-S-FORWARDER. Now, defining $\mathcal{D}$ as in the previous case, we can conclude that FORWARDER($\mathcal{P}$) holds true. □

In the above proof we have only used *very* basic properties of our computational model, which should be satisfied in most computational models for simulation-based security. Hence, the above theorem should be true in all such models.

5.2.2 On the Validity of FORWARDER

We show that not all process expressions satisfy FORWARDER. However, it is possible to identify an interesting class of process expressions that satisfy this axiom. The following terminology will become clear below.

Definition 11. *We call a sequential process expression $\mathcal{P}$* network predictable *if FORWARDER($\mathcal{P}$) holds.*

Non-network predictable process expressions. We now show that not all regular process expressions satisfy FORWARDER. It is useful to recall Remark 9. Intuitively, as seen in this remark, if the dummy is defined only depending on $\mathcal{P}$ but completely independent of $\mathcal{E}$, then the dummy should only react to inputs from the environment if $\mathcal{P}$ reacts to these inputs since otherwise the dummy may get "exhausted" by the environment. In other words, the environment can force the dummy to terminate before $\mathcal{P}$ does, and in this case, the dummy cannot forward messages anymore. However, in general, the dummy does not know on what channels $\mathcal{P}$ expects messages and it also does not know what messages $\mathcal{P}$ accepts. We use this intuition to show:

Proposition 12. *There exists $\mathcal{P} \in \mathbf{R}$ which is not network predictable.*

Proof. Consider

$$\mathcal{P} = (\mathtt{in}(c, 0).\mathtt{in}(c_0, x).\mathtt{out}(c_{out}, x)) + (\mathtt{in}(c, 1).\mathtt{in}(c_1, x).\mathtt{out}(c_{out}, x)))$$

where c and c_{out} are declared to be IO channels and c_0 and c_1 are network channels. That is, the environment determines via a bit sent on the IO channel c (which is invisible to the dummy) on which network channel—c_0 or c_1—$\mathcal{P}$ will accept a message. Now, assume that there is a dummy $\mathcal{D}$ which satisfies the required conditions for FORWARDER($\mathcal{P}$). It is helpful to think of $\mathcal{D}$ to be in SMNF. By Lemma 3 this is w.l.o.g. Let us first consider the following environment $\mathcal{E}$ (which can easily be described as process expression): $\mathcal{E}$ randomly chooses a bit b and a number i between 1 and $\mathtt{comsize}(\mathcal{D})(\mathbf{n}) + 1$. Then, $\mathcal{E}$ sends $i - 1$ randomly chosen messages, say of the length of the security parameter, on c_0 or c_1 where for every message $\mathcal{E}$ again makes a random decision on which channel—c_0 or c_1—to send the message. In all of these cases, $\mathcal{E}$ will not receive any answer but will be triggered on `start`. Then, for the ith step $\mathcal{E}$ again chooses a random message, say m, then sends b on channel c after which $\mathcal{E}$ will be triggered through `start`, and then sends m on c_b. Now, if $\mathcal{E}$ receives m back on c_{out}, then it outputs 1 on `decision` and otherwise (if $\mathcal{E}$ is triggered by `start`) outputs 0.

We argue that $\mathcal{E}$ can distinguish between $\mathcal{P}$ and $\mathcal{D} \upharpoonright \mathcal{P}$, i.e., $\mathcal{E} \upharpoonright \mathcal{P} \not\equiv \mathcal{E}' \upharpoonright \mathcal{D} \upharpoonright \mathcal{P}$.

The dummy $\mathcal{D}$ can only forward $< \mathtt{comsize}(\mathcal{D})(\mathbf{n}) + 1$ messages from c_0' to c_0 and c_1' to c_1 before it terminates. Thus, if $\mathcal{E}$ chooses $i = \mathtt{comsize}(\mathcal{D})(\mathbf{n}) + 1$, which happens with non-negligible probability, the probability that $\mathcal{D}$ accepts messages on both c_0' and c_1' for all $\mathtt{comsize}(\mathcal{D})(\mathbf{n})$ steps before the last message is sent by $\mathcal{E}$ must be negligible: Otherwise when $\mathcal{E}$ sends the last message to c_b and expects to obtain input on c_{out}, the probability that $\mathcal{E}$ in fact obtains input on this channel is negligible as $\mathcal{D}$ will be terminated with overwhelming probability. Thus, there is a non-negligible probability that $\mathcal{D}$ does not accept a message on some of its input channels—c_0' or c_1'. Consequently, $\mathcal{E}$ has a non-negligible chance of guessing this position in the run. In case $\mathcal{E}$ guessed correctly, it will output 0 since it will not obtain input on c_{out} but will be triggered on `start`. Thus, when interacting with $\mathcal{D}$ and $\mathcal{P}$, the environment $\mathcal{E}$ will output 0 with non-negligible probability while when interacting only with $\mathcal{P}$ it will always return 1. Hence, $\mathcal{E} \upharpoonright \mathcal{P} \not\equiv \mathcal{E}' \upharpoonright \mathcal{D} \upharpoonright \mathcal{P}$. □

The proof of Proposition 12 indicates that in order to obtain a class of network predictable process expressions, one needs to make sure that the dummy can determine from the traffic on network channels on which channels the process accepts messages and of what shape the messages have to be in order to be accepted by the M-terms, since otherwise the dummy can be exhausted. This is why we call such process expressions network predictable.

Using length functions as in the version of PIOA in [6], the process $\mathcal{P}$ in the proof of Proposition 12 can also be expressed in PIOA. Therefore, we can remark:

Remark 13. *In the version of PIOA with length functions, protocols can be expressed that are not network predictable.*

In Section 6.1 we will see that FORWARDER can fail in PIOA even without length functions.

A class of network predictable process expressions. We now define a class of process expressions, called standard protocols, which in fact are network predictable.

Definition 14. *A process expression $\mathcal{P} \in \mathbf{R}$ is called a* standard protocol *if it is of the form $\mathcal{P} = \mathcal{P}_1 \mid \cdots \mid \mathcal{P}_n$ where every $\mathcal{P}_i$ is in SSMNF.*[5]

We note that the class of standard protocols contains the protocols expressible in the models proposed in [27] and [9]. As mentioned in Section 2.1, in a later version of the PIOA model [6], length functions allow to express certain M-terms (guards). Hence, by Remark 13, this yields a class of protocols which goes beyond the class of standard protocols.

We show:

Proposition 15. *Standard protocols are network predictable.*

Proof. Let $\mathcal{P} = \mathcal{P}_1 \mid \cdots \mid \mathcal{P}_n$ be a standard protocol. Define

$$\mathcal{D}_i = \mathcal{D}_R^{\mathtt{net}}(\mathsf{C}_{\mathtt{in}}^{\mathtt{net}}(\mathcal{P}_i), \mathsf{C}_{\mathtt{out}}^{\mathtt{net}}(\mathcal{P}_i), \mathtt{comsize}(\mathcal{P}_i)(\mathbf{n})).$$

(Recall the definition of $\mathcal{D}_R^{\mathtt{net}}$ from Section 5.1). Hence, $\mathcal{D}_i$ simply forwards all message on network channels from and to $\mathcal{P}_i$. To see that

$$\mathcal{E} \mid \mathcal{P}_i \equiv \mathcal{E}' \mid \mathcal{D}_i \mid \mathcal{P}_i,$$

for all $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P}_i)$, it suffices to observe that the number of messages that can be sent to $\mathcal{P}_i$ and that can be received from $\mathcal{P}_i$ before $\mathcal{P}_i$ terminates is bounded by $\mathtt{comsize}(\mathcal{P}_i)$—a bound known by $\mathcal{D}_i$—since $\mathcal{P}_i$ accepts all messages sent on (network) channels. Thus, $\mathcal{D}_i$ only needs to forward $\mathtt{comsize}(\mathcal{P}_i)$ messages. After $\mathcal{P}_i$ has terminated, $\mathcal{D}_i$ does not need to forward messages anymore.

Now, since the set of network channels of the $\mathcal{P}_i$ are pairwise disjoint, it is clear that with $\mathcal{D} = \mathcal{D}_1 \mid \cdots \mid \mathcal{D}_n$ we obtain that

$$\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E}' \mid \mathcal{D} \mid \mathcal{P}.$$

for every $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$. □

Proposition 15 can be extended to bigger classes of protocols. For instance to a class of protocols with a polynomial number of parties where messages addressed to a certain party have to be prefixed by the name of the party and its role in the protocol. More precisely, if the M-terms of parties accept exactly those messages prefixed with the correct recipient and role, a dummy can predict whether or not a message is accepted by a party.

[5] Recall from Definition 2 that SSMNF means that $\mathcal{P}_i$ is in SMNF and the M-terms on external input channels accept every message.

5.3 Declaring the Environment to be the Master Decision Process

The following theorem states the relationships between the security notions introduced in Definition 6 in case the environment may play the role of the master decision process. Among others, it says that the notions strong simulatability, black-box simulatability, and universal composability coincide given that the real protocol $\mathcal{P}$ is network predictable, i.e., the axiom FORWARDER($\mathcal{P}$) is true. However, many of the relationships between the security notions are true independently of this axiom.

Theorem 16. *Let $\mathcal{P}, \mathcal{F} \in \mathbf{R}$.*

1. $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.
2. $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{WBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.
3. *The notions in 1. imply those in 2.*
4. *If $\mathcal{P}$ is network predictable, i.e., if FORWARDER($\mathcal{P}$) holds, then the in 1. and 2. mentioned security notions are all equivalent.*

The theorem holds both for the case where $\mathbf{MD}$ *only contains closed process expressions and for the case where* $\mathbf{MD}$ *may contain open process expressions.*

Proof. Statement 1. From MMD-INCLUSION we easily obtain that $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{SBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$. It is easy to see that $\mathtt{SBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{SBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$. We now show that $\mathtt{SBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.

1. Assume that $\mathtt{SBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.
2. The definition implies that $\mathcal{P}$ and $\mathcal{F}$ are IO-compatible and:
   $\exists \mathcal{S} \in \mathbf{R}\text{-Adv}_{\mathcal{P}}(\mathcal{F}).\forall \mathcal{A} \in \mathbf{R}\text{-Adv}(\mathcal{P}).\forall \mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{A} \upharpoonright \mathcal{P}) : \mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{S} \upharpoonright \mathcal{F}$.
3. Choosing $\mathcal{A}$ to be the regular IO dummy adversary $\mathcal{D}_R^{\mathtt{io}} = \mathcal{D}_R^{\mathtt{io}}(\mathsf{C}_{\mathtt{in}}^{\mathtt{net}}(\mathcal{P}), \mathsf{C}_{\mathtt{out}}^{\mathtt{net}}(\mathcal{P}), q(\mathbf{n}))$ we obtain:
   $\exists \mathcal{S} \in \mathbf{R}\text{-Adv}_{\mathcal{P}}(\mathcal{F}).\forall \text{ poly. } q(\mathbf{n}).\forall \mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{D}_R^{\mathtt{io}} \upharpoonright \mathcal{P}) : \mathcal{E} \upharpoonright \mathcal{D}_R^{\mathtt{io}} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{D}_R^{\mathtt{io}} \upharpoonright \mathcal{S} \upharpoonright \mathcal{F}$.
4. Choose $\mathcal{S}$ as in 3., let $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$, and $q(\mathbf{n}) = \mathtt{comsize}(\mathcal{P})(\mathbf{n}) + \mathtt{comsize}(\mathcal{E})(\mathbf{n})$. We have:
   $$\begin{array}{rcll} \mathcal{E} \upharpoonright \mathcal{P} & \equiv & \mathcal{E}' \upharpoonright \mathcal{D}_R^{\mathtt{net}} \upharpoonright \mathcal{P} & \text{(REG-S-FORWARDER)} \\ & \equiv & \mathcal{E}'' \upharpoonright \mathcal{D}_R^{\mathtt{io}} \upharpoonright \mathcal{P} & \text{(RENAME)} \\ & \equiv & \mathcal{E}'' \upharpoonright \mathcal{D}_R^{\mathtt{io}} \upharpoonright \mathcal{S} \upharpoonright \mathcal{F} & (\mathcal{E}'' \in \mathbf{MD}\text{-Env}(\mathcal{D}_R^{\mathtt{io}} \upharpoonright \mathcal{P}), 3.) \\ & \equiv & \mathcal{E}' \upharpoonright \mathcal{D}_R^{\mathtt{net}} \upharpoonright \mathcal{S} \upharpoonright \mathcal{F} & \text{(RENAME)} \\ & \equiv & \mathcal{E} \upharpoonright \mathcal{S} \upharpoonright \mathcal{F} & \text{(REG-S-FORWARDER)} \end{array}$$
   where $\mathcal{E}'$ is defined as in REG-S-FORWARDER and $\mathcal{E}''$ is obtained from $\mathcal{E}'$ by declaring the renamed network channels c' of $\mathcal{P}$ to be IO channels. Since $\mathcal{E}$ is valid for $\mathcal{P}$ and all network channels of $\mathcal{P}$ occurring in $\mathcal{E}$ have been renamed according to $\mathcal{D}_R^{\mathtt{net}}$ and declared to be IO channels, it is clear that $\mathcal{E}'' \in \mathbf{MD}\text{-Env}(\mathcal{D}_R^{\mathtt{io}} \upharpoonright \mathcal{P})$.
5. From 4. we immediately obtain that $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.

Statement 2. It is obvious that $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ since if the real adversary is regular, then so must be the ideal adversary because of the compatability requirement for $\mathcal{A} \upharpoonright \mathcal{P}$ and $\mathcal{I} \upharpoonright \mathcal{F}$. We now show that $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.

Assume that $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ and let $\mathcal{A} \in \mathbf{M}\text{-Adv}(\mathcal{P})$. We need to show (*): There exists $\mathcal{I} \in \mathbf{M}\text{-Adv}_{\mathcal{A} \upharpoonright \mathcal{P}}(\mathcal{F})$ such that $\mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{I} \upharpoonright \mathcal{F}$ for every $\mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{A} \upharpoonright \mathcal{P})$.

If $\mathcal{A} \in \mathbf{R}$, then (*) follows by the assumption $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.

Assume that $\mathcal{A} \in \mathbf{M} \setminus \mathbf{R}$. Let $\mathcal{A}'$ be obtained from $\mathcal{A}$ by replacing every occurrence of `start` by the new channel `start`$'$. Then, $\mathcal{A}' \in \mathbf{R}\text{-Adv}(\mathcal{P})$. By assumption, there exists $\mathcal{I}' \in \mathbf{R}\text{-Adv}_{\mathcal{A}' \upharpoonright \mathcal{P}}(\mathcal{F})$ such that $\mathcal{E} \upharpoonright \mathcal{A}' \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{I}' \upharpoonright \mathcal{F}$ for every $\mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{A}' \upharpoonright \mathcal{P})$. Let $\mathcal{I}$ be obtained from $\mathcal{I}'$ by replacing every occurrence of `start`$'$ by `start`. Let $\mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{A} \upharpoonright \mathcal{P})$. Since `start` occurs in $\mathcal{A}$, we know that $\mathcal{E} \in \mathbf{D}$. The following completes the proof of the equivalence between the two variants of UC. In the second equation we use that $\mathcal{E} \upharpoonright !_{q(\mathbf{n})}\ (\mathtt{in}(\mathtt{start},\varepsilon).\ \mathtt{out}(\mathtt{start}',\varepsilon)) \in \mathbf{MD}\text{-Env}(\mathcal{A}' \upharpoonright \mathcal{P})$ where $q(\mathbf{n}) = \mathtt{comsize}(\mathcal{A})(\mathbf{n}) + \mathtt{comsize}(\mathcal{I})(\mathbf{n})$.

$$\begin{array}{rcll}
\mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} & \equiv & \mathcal{E} \upharpoonright !_{q(\mathbf{n})}\ \mathtt{in}(\mathtt{start},\varepsilon).\mathtt{out}(\mathtt{start}',\varepsilon) \upharpoonright \mathcal{A}' \upharpoonright \mathcal{P} & \text{(RENAME-START)} \\
 & \equiv & \mathcal{E} \upharpoonright !_{q(\mathbf{n})}\ \mathtt{in}(\mathtt{start},\varepsilon).\mathtt{out}(\mathtt{start}',\varepsilon) \upharpoonright \mathcal{I}' \upharpoonright \mathcal{F} & \text{(Definition of } \mathcal{I}'\text{)} \\
 & \equiv & \mathcal{E} \upharpoonright \mathcal{I} \upharpoonright \mathcal{F} & \text{(RENAME-START)}
\end{array}$$

Clearly, we have that $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{WBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$. It is also obvious that $\mathtt{WBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ since $\mathcal{A} \upharpoonright \mathcal{S}$ is the ideal adversary required in $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ where $\mathcal{S}$ is the simulator obtained from $\mathtt{WBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.

We now show that $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ which concludes the proof of statement 2.

1. Assume that $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.

2. The definition yields that $\mathcal{P}$ and $\mathcal{F}$ are IO-compatible and:
$\forall \mathcal{A} \in \mathbf{R}\text{-Adv}(\mathcal{P}).\exists \mathcal{I} \in \mathbf{R}\text{-Adv}_{\mathcal{A} \upharpoonright \mathcal{P}}(\mathcal{F}).\forall \mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{A} \upharpoonright \mathcal{P}) : \mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{I} \upharpoonright \mathcal{F}$.

3. Choosing $\mathcal{A} = \mathcal{D}_R^{\mathtt{io}} = \mathcal{D}_R^{\mathtt{io}}(\mathsf{C}_{\mathtt{in}}^{\mathtt{net}}(\mathcal{P}), \mathsf{C}_{\mathtt{out}}^{\mathtt{net}}(\mathcal{P}), q(\mathbf{n}))$ for some $q(\mathbf{n})$ we obtain:
$\exists \mathcal{S}_{q(\mathbf{n})} = \mathcal{I} \in \mathbf{R}\text{-Adv}_{\mathcal{D}_R^{\mathtt{io}} \upharpoonright \mathcal{P}}(\mathcal{F}).\forall \mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{D}_R^{\mathtt{io}} \upharpoonright \mathcal{P}) : \mathcal{E} \upharpoonright \mathcal{D}_R^{\mathtt{io}} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{S}_{q(\mathbf{n})} \upharpoonright \mathcal{F}$.

4. Let $\mathcal{A} \in \mathbf{M}\text{-Adv}(\mathcal{P})$, $q(\mathbf{n}) = \mathtt{comsize}(\mathcal{A})(\mathbf{n}) + \mathtt{comsize}(\mathcal{P})(\mathbf{n})$, choose $\mathcal{S}_{q(\mathbf{n})}$ as in 3. and let $\mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{A} \upharpoonright \mathcal{P})$. We obtain:

$$\begin{array}{rcll}
\mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} & \equiv & \mathcal{E} \upharpoonright \mathcal{A}' \upharpoonright \mathcal{D}_R^{\mathtt{net}} \upharpoonright \mathcal{P} & \text{(REG-ADV-FORWARDER)} \\
 & \equiv & \mathcal{E} \upharpoonright \mathcal{A}'' \upharpoonright \mathcal{D}_R^{\mathtt{io}} \upharpoonright \mathcal{P} & \text{(RENAME)} \\
 & \equiv & \mathcal{E} \upharpoonright \mathcal{A}'' \upharpoonright \mathcal{S}_{q(\mathbf{n})} \upharpoonright \mathcal{F} & (\mathcal{E} \upharpoonright \mathcal{A}'' \in \mathbf{MD}\text{-Env}(\mathcal{D}_R^{\mathtt{io}} \upharpoonright \mathcal{P}), 3.) \\
 & \equiv & \mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{S}'_{q(\mathbf{n})} \upharpoonright \mathcal{F} & \text{(RENAME)}
\end{array}$$

   where $\mathcal{A}'$ is defined as in REG-ADV-FORWARDER, $\mathcal{D}_R^{\mathtt{io}}$ and $\mathcal{A}''$ are obtained from $\mathcal{D}_R^{\mathtt{net}}$ and $\mathcal{A}'$ by declaring the renamed network channel c' of $\mathcal{P}$ to be IO channels, and $\mathcal{S}'_{q(\mathbf{n})}$ is obtained from $\mathcal{S}_{q(\mathbf{n})}$ by declaring the IO channels c' to be network channels and renaming them to c according to $\mathcal{P}$.

5. By observing that $\mathcal{S}'_{q(\mathbf{n})} \in \mathbf{R}\text{-Adv}_{\mathcal{P}}(\mathcal{F})$ and that $\mathcal{S}'_{q(\mathbf{n})}$ only depends on $\mathcal{F}$, $\mathcal{P}$, and (the communication size of) $\mathcal{A}$, 4. immediately implies that $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ (for both variants of weak blackbox simulatability). □

Statement 3. From $\texttt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ we know that there exists $\mathcal{S} \in \mathbf{R}\text{-Adv}_{\mathcal{P}}(\mathcal{F})$ such that $\mathcal{E} \rceil \mathcal{P} \equiv \mathcal{E} \rceil \mathcal{S} \rceil \mathcal{F}$ for every $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$. To show $\texttt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$, assume that $\mathcal{A} \in \mathbf{M}\text{-Adv}(\mathcal{P})$ and $\mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{A} \rceil \mathcal{P})$. From MMD-INCLUSION it follows that $\mathcal{E} \rceil \mathcal{A} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$, and thus, $\mathcal{E} \rceil \mathcal{A} \rceil \mathcal{P} \equiv \mathcal{E} \rceil \mathcal{A} \rceil \mathcal{S} \rceil \mathcal{F}$. Obviously, $\mathcal{A} \rceil \mathcal{S} \in \mathbf{M}\text{-Adv}_{\mathcal{A} \rceil \mathcal{P}}(\mathcal{F})$. Thus, defining the ideal adversary $\mathcal{I}$ to be $\mathcal{A} \rceil \mathcal{S}$ concludes the proof.

Statement 4. Assume that FORWARDER($\mathcal{P}$). It suffice to show that $\texttt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\texttt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$. Let $\mathcal{D}$ be the dummy whose existence is guaranteed by FORWARDER($\mathcal{P}$). Let $\mathcal{D}^{\texttt{io}}$ be obtained from $\mathcal{D}$ by declaring all the channels c' to be IO channels. (The channels c occurring in $\mathcal{D}$ remain network channels).

1. Assume that $\texttt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.

2. The definition yields that $\mathcal{P}$ and $\mathcal{F}$ are IO-compatible and:
   $\forall \mathcal{A} \in \mathbf{R}\text{-Adv}(\mathcal{P}).\exists \mathcal{I} \in \mathbf{R}\text{-Adv}_{\mathcal{A} \rceil \mathcal{P}}(\mathcal{F}).\forall \mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{A} \rceil \mathcal{P}) : \mathcal{E} \rceil \mathcal{A} \rceil \mathcal{P} \equiv \mathcal{E} \rceil \mathcal{I} \rceil \mathcal{F}$.

3. Choosing $\mathcal{A} = \mathcal{D}^{\texttt{io}}$ we obtain:
   $\exists \mathcal{S} = \mathcal{I} \in \mathbf{R}\text{-Adv}_{\mathcal{D}^{\texttt{io}} \rceil \mathcal{P}}(\mathcal{F}).\forall \mathcal{E} \in \mathbf{MD}\text{-Env}(\mathcal{D}^{\texttt{io}} \rceil \mathcal{P}) : \mathcal{E} \rceil \mathcal{D}^{\texttt{io}} \rceil \mathcal{P} \equiv \mathcal{E} \rceil \mathcal{S} \rceil \mathcal{F}$.

4. Choose $\mathcal{S}$ as in 3. and let $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$. We obtain:

   $$\begin{array}{rcll} \mathcal{E} \rceil \mathcal{P} & \equiv & \mathcal{E}' \rceil \mathcal{D} \rceil \mathcal{P} & (\text{FORWARDER}(\mathcal{P})) \\ & \equiv & \mathcal{E}'' \rceil \mathcal{D}^{\texttt{io}} \rceil \mathcal{P} & (\text{RENAME}) \\ & \equiv & \mathcal{E}'' \rceil \mathcal{S} \rceil \mathcal{F} & (\mathcal{E}'' \in \mathbf{MD}\text{-Env}(\mathcal{D}^{\texttt{io}} \rceil \mathcal{P}), 3.) \\ & \equiv & \mathcal{E} \rceil \mathcal{S}' \rceil \mathcal{F} & (\text{RENAME}) \end{array}$$

   where $\mathcal{E}'$ is defined as in FORWARDER($\mathcal{P}$), $\mathcal{E}''$ is obtained from $\mathcal{E}'$ by declaring the renamed network channels c' of $\mathcal{P}$ to be IO channels, and $\mathcal{S}'$ is obtained from $\mathcal{S}$ by declaring the IO channels c' (which correspond to the IO channels of $\mathcal{D}^{\texttt{io}}$) to be network channels and renaming them to c according to $\mathcal{P}$.

5. By observing that $\mathcal{S}' \in \mathbf{R}\text{-Adv}_{\mathcal{P}}(\mathcal{F})$, 4. immediately implies that $\texttt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$. □

By Theorem 10 and Proposition 12, to show Theorem 16, 4., we cannot dispense with the assumption that $\mathcal{P}$ is network predictable.

5.4 Restricting the Environment to be a Decision Process

In this section, we consider the case were for blackbox simulatability and universal composability the environment is restricted to be a decision process while the adversary may play the role of a master process. Interestingly, in this setting not all three security notions are equivalent even if the real protocol is network predictable.

We first note that certain variants of SS, BB, and UC do not make sense in case the environment is restricted to be a decision process as every two IO-compatible protocols would be related:

Remark 17. *For all IO-compatible protocols $\mathcal{P}$ and $\mathcal{F}$, we have that the relationships $\texttt{SS}_{(\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$, $\texttt{SBB}_{(\mathbf{R},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$, $\texttt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$, and $\texttt{UC}_{(\mathbf{R},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$ are true since there are no master processes and therefore no computation can take place. Note that in UC if the real adversary is a regular process expression, then so is the ideal adversary, and thus, the two variants of UC are equivalent.*

Theorem 18. *Let $\mathcal{P}, \mathcal{F} \in \mathbf{R}$. Then,*

1. $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBB}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBB}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBB}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.
2. $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{WBB}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$.
3. $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *implies* $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$.
4. $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{UC}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{M}\backslash\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{UC}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{M}\backslash\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.
5. *The notions in 1. imply those in 4. , and* $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *implies those in 4.*

The theorem holds both for the case where **MD** *and* **D** *only contain closed process expressions and for the case where* **MD** *and* **D** *may contain open process expressions.*

Proof. Statement 1. Using the definitions and MD-INCLUSION, it immediately follows that $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{SBB}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$. Clearly, $\mathtt{SBB}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{SBB}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$. The converse is also true: If the real adversary $\mathcal{A}$ is regular, then $\mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{S} \upharpoonright \mathcal{F}$ because no one of the two process expressions contains `start`, and thus, no computation takes place. We have that $\mathtt{SBB}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$ and $\mathtt{SBB}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ are equivalent since if the real adversary contains `start`, then the environment may not contain `start`, and hence, belongs to **D**. Similar to the proof of Theorem 16, 1. we now show that blackbox simulatability implies strong simulatability.

1. Assume that $\mathtt{SBB}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$.
2. The definition yields that $\mathcal{P}$ and $\mathcal{F}$ are IO-compatible and:
   $\exists \mathcal{S} \in \mathbf{R}\text{-Adv}_{\mathcal{P}}(\mathcal{F}).\forall \mathcal{A} \in \mathbf{M}\text{-Adv}(\mathcal{P}).\forall \mathcal{E} \in \mathbf{D}\text{-Env}(\mathcal{A} \upharpoonright \mathcal{P}) : \mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{S} \upharpoonright \mathcal{F}$.
3. Choosing $\mathcal{A}$ to be the master IO dummy adversary $\mathcal{D}^{\mathtt{io}}_M = \mathcal{D}^{\mathtt{io}}_M(\mathsf{C}^{\mathtt{net}}_{\mathtt{in}}(\mathcal{P}), \mathsf{C}^{\mathtt{net}}_{\mathtt{out}}(\mathcal{P}), q(\mathbf{n}))$ we obtain:
   $\exists \mathcal{S} \in \mathbf{R}\text{-Adv}_{\mathcal{P}}(\mathcal{F}).\forall$ polynomials $q(\mathbf{n}).\forall \mathcal{E} \in \mathbf{D}\text{-Env}(\mathcal{D}^{\mathtt{io}}_M \upharpoonright \mathcal{P}) : \mathcal{E} \upharpoonright \mathcal{D}^{\mathtt{io}}_M \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{D}^{\mathtt{io}}_M \upharpoonright \mathcal{S} \upharpoonright \mathcal{F}$.
4. Choose $\mathcal{S}$ as in 3., let $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$, and $q(\mathbf{n}) = \mathtt{comsize}(\mathcal{P})(\mathbf{n}) + \mathtt{comsize}(\mathcal{S} \upharpoonright \mathcal{F})(\mathbf{n}) + \mathtt{comsize}(\mathcal{E})(\mathbf{n})$. We have:

$$\begin{array}{rcll}
\mathcal{E} \upharpoonright \mathcal{P} & \equiv & \mathcal{E}'_M \upharpoonright \mathcal{D}^{\mathtt{net}}_M \upharpoonright \mathcal{P} & \text{(MASTER-S-FORWARDER)} \\
& \equiv & \mathcal{E}''_M \upharpoonright \mathcal{D}^{\mathtt{io}}_M \upharpoonright \mathcal{P} & \text{(RENAME)} \\
& \equiv & \mathcal{E}''_M \upharpoonright \mathcal{D}^{\mathtt{io}}_M \upharpoonright \mathcal{S} \upharpoonright \mathcal{F} & (\mathcal{E}''_M \in \mathbf{D}\text{-Env}(\mathcal{D}^{\mathtt{io}}_M \upharpoonright \mathcal{P}), 3.) \\
& \equiv & \mathcal{E}'_M \upharpoonright \mathcal{D}^{\mathtt{net}}_M \upharpoonright \mathcal{S} \upharpoonright \mathcal{F} & \text{(RENAME)} \\
& \equiv & \mathcal{E} \upharpoonright \mathcal{S} \upharpoonright \mathcal{F} & \text{(MASTER-S-FORWARDER)}
\end{array}$$

   where $\mathcal{E}'_M$ is defined as in MASTER-S-FORWARDER and $\mathcal{E}''_M$ is obtained from $\mathcal{E}'_M$ by declaring the renamed network channels c' of $\mathcal{P}$ to be IO channels. Since $\mathcal{E}$ is valid for $\mathcal{P}$, all network channels of $\mathcal{P}$ occurring in $\mathcal{E}$ have been renamed according to $\mathcal{D}^{\mathtt{net}}_M$ and declared to be IO channels, and `start` has been renamed to `start`$'$, it is clear that $\mathcal{E}''_M \in \mathbf{D}\text{-Env}(\mathcal{D}^{\mathtt{io}}_M \upharpoonright \mathcal{P})$.
5. From 4. we immediately obtain that $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.

Statement 2. It suffices to observe that if the real adversary is not a master process, then no computation will take place.

Statement 3. This statement is obvious.

Statement 4. The reasoning here is similar to the one for the different variants of blackbox simulatability above. In addition we use that if the real adversary contains `start`, then so does the ideal adversary.

Statement 5. The first implication was proved in Theorem 16 and the second implication immediately follows since if the real adversary contains `start`, then so does the ideal adversary. □

As illustrated next, $\mathtt{UC}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{M}\backslash\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$, or equivalently $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$, in general does not imply $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ or $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ even if $\mathcal{P}$ is network predictable. Intuitively, in the proof of Theorem 16, 4. if the adversary may be a master process, then the simulator $\mathcal{S}=\mathcal{I}$ we obtain is also a master process. However, to show strong simulatability the simulator needs to be a regular process expression. The example used to prove the following theorem shows that in general master process expressions cannot be turned into regular process expressions without changing the behavior of the overall system. Therefore, the proof of Theorem 16 would not go through if the adversary may be a master process while the environment is a decision process.

Theorem 19. *There exist $\mathcal{P},\mathcal{F}\in\mathbf{R}$ such that $\mathcal{P}$ is network predictable and $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$ does not imply $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ and $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$.*

To prove the theorem, we construct $\mathcal{P},\mathcal{F}\in\mathbf{R}$ and show the properties claimed.

Roughly speaking, $\mathcal{P}$ receives a bit x from the environment on an IO channel, returns an acknowledgment of receipt on a network channel, waits for a send request on a network channel, and then returns x on a network channel. The process $\mathcal{F}$ works exactly in the same way but if $x=0$, then in the last step it will not return x.

Intuitively, a *master* process $\mathcal{S}$ which has only access to the network channels of $\mathcal{F}$ can simulate $\mathcal{P}$ using $\mathcal{F}$ because if in the last step $\mathcal{F}$ does not return an answer, $\mathcal{S}$ will be triggered, i.e., receives input on the channel `start`, and thus knows that $\mathcal{F}$'s answer was 0. If $\mathcal{S}$ is *not* a master process, then there is no way for $\mathcal{S}$ to know what x was, and therefore will not be able to simulate $\mathcal{P}$ (using $\mathcal{F}$). Now, the reason that $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$ holds but $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ and $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$ do not hold is that for the latter two security notions one requires that $\mathcal{P}$ can be simulated using $\mathcal{F}$ by a simulator that is *not* a master process, while for $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ the simulator (i.e., the ideal adversary) may be a master process.

Formally, the process expression $\mathcal{P}$ uses the following channels: $\mathsf{C}^{\mathtt{io}}_{\mathtt{in}}(\mathcal{P})=\{c_0\}$, $\mathsf{C}^{\mathtt{net}}_{\mathtt{out}}(\mathcal{P})=\{c_1,c_2\}$, and $\mathsf{C}^{\mathtt{net}}_{\mathtt{in}}(\mathcal{P})=\{c_3\}$. Now, $\mathcal{P}$ is defined as follows:

$$\mathcal{P} \;=\; \mathtt{in}(c_0,t_{x\in\{0,1\}}).(\mathtt{out}(c_1,\mathtt{received}) \,||\, \mathtt{in}(c_3,\mathtt{send\text{-}req}).\mathtt{out}(c_2,x)).$$

where the M-term $t_{x\in\{0,1\}}$ only accepts a bit string a if it is 0 or 1. In this case, x is set to a. Clearly, $\mathcal{P}$ is network predictable: A possible dummy is

$$\begin{aligned}\mathcal{D} \;=\; & \mathtt{in}(c_1,\mathtt{received}).(\mathtt{out}(c_1',\mathtt{received}) \,||\\ & \quad \mathtt{in}(c_3',\mathtt{send\text{-}req}).(\mathtt{out}(c_3,\mathtt{send\text{-}req}) \,||\, \mathtt{in}(c_2,x).\mathtt{out}(c_2',x))).\end{aligned}$$

The channels of $\mathcal{F}$ are defined just as for $\mathcal{P}$ except that the network channels c_1, c_2, c_3 are renamed to c_1', c_2', c_3'. Also, $\mathcal{F}$ uses the internal channel $c'_{\mathtt{int}}$.

$$\mathcal{F} \;=\; \mathtt{in}(c_0, t_{x\in\{0,1\}}).\Big(\mathtt{out}(c_1', \mathtt{received}) \;||$$
$$\mathtt{in}(c_3', \mathtt{send\text{-}req}).\big(\mathtt{out}(c'_{\mathtt{int}}, x) \;||\; \mathtt{in}(c'_{\mathtt{int}}, 1).\mathtt{out}(c_2', x)\big)\Big)$$

where, formally, 1 is an M-term which only accepts the input if it is 1.
We now show:

Claim I. $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P}, \mathcal{F})$.

Proof sketch of Claim I. By Theorem 18, we know that $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P}, \mathcal{F})$ iff $\mathtt{UC}_{(\mathbf{M}\setminus\mathbf{R},\mathbf{M}\setminus\mathbf{R},\mathbf{D})}(\mathcal{P}, \mathcal{F})$. To prove the claim, let $\mathcal{A} \in (\mathbf{M}\setminus\mathbf{R})\text{-Adv}(\mathcal{P})$. We need to show that there exists $\mathcal{I} {\in} (\mathbf{M}\setminus\mathbf{R})\text{-Adv}_{\mathcal{P}}(\mathcal{I})$ such that $\mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{I} \upharpoonright \mathcal{F}$ for every $\mathcal{E} \in \mathbf{D}\text{-Env}(\mathcal{A} \upharpoonright \mathcal{P})$. We will define a master process expression $\mathcal{S}$ which uses $\mathcal{F}$ to simulate $\mathcal{P}$. Then, $\mathcal{I}$ will be the parallel composition of $\mathcal{A}$ (with $\mathtt{start}$ renamed) and $\mathcal{S}$.

The simulator $\mathcal{S}$ works as follows: It forwards messages on $\mathcal{F}$'s network channels from/to the adversary $\mathcal{A}$. If right after forwarding a message from the adversary $\mathcal{A}$ on c_3 to $\mathcal{F}$ on c_3', $\mathcal{S}$ receives a message on $\mathtt{start}$, then $\mathcal{S}$ sends 0 on c_2 because this situation occurs exactly when $\mathcal{F}$ is expected to send a message on c_2' but does not do so because $x = 0$. In all other situations where $\mathcal{S}$ receives a message on $\mathtt{start}$, $\mathcal{S}$ forwards it on $\mathtt{start}'$ (to the adversary $\mathcal{A}$). One can now show that $\mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright (\mathcal{A} \upharpoonright \mathcal{S}) \upharpoonright \mathcal{F}$ where $\mathcal{I} = \mathcal{A} \upharpoonright \mathcal{S}$ is the ideal adversary, which concludes the proof of the Claim I. Appendix A contains a formulation of $\mathcal{S}$ as a process expression and a more detailed proof.

Claim II. $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P}, \mathcal{F})$ does not hold.

Proof sketch of Claim II . The proof of Claim II is by contradiction. Assume that there exists a simulator $\mathcal{S}' \in \mathbf{R}\text{-Adv}_{\mathcal{P}}(\mathcal{F})$ such that $\mathcal{E} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{S}' \upharpoonright \mathcal{F}$ for every $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$. We construct a closed master decision process expression $\mathcal{E}' \in \mathbf{MD}\text{-Valid}(\mathcal{P})$ such that $\mathcal{E}' \upharpoonright \mathcal{P} \not\equiv \mathcal{E}' \upharpoonright \mathcal{S}' \upharpoonright \mathcal{F}$. The environment $\mathcal{E}'$ works as follows: It generates a random bit, sends it on channel c_0, waits for acknowledgment of receipt (on channel c_1), sends a "send request" on c_3, and then checks whether the bit returned on c_2 is the one sent before. If at some point except at the beginning, $\mathcal{E}'$ receives a message on $\mathtt{start}$, then $\mathcal{E}'$ writes 0 on $\mathtt{decision}$ and terminates. In other words, $\mathcal{E}'$ always expects to receive a message back from the process it is interacting with. Now, while in $\mathcal{E}' \upharpoonright \mathcal{P}$ the environment $\mathcal{E}'$ will output 1 with probability 1, it is not hard to show that in $\mathcal{E}' \upharpoonright \mathcal{S}' \upharpoonright \mathcal{F}$ the environment outputs 1 with at most probability 1/2 since the simulator does not know which bit was sent by $\mathcal{E}'$ to $\mathcal{F}$. Hence, $\mathcal{E}' \upharpoonright \mathcal{P} \not\equiv \mathcal{E}' \upharpoonright \mathcal{S}' \upharpoonright \mathcal{F}$. This concludes the proof of Claim II. Appendix A contains a more precise formulation of $\mathcal{E}'$ as a process expression and a more detailed argument.

Claim III. $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P}, \mathcal{F})$ does not hold.

The proof is similar to the one of Claim II. One simply chooses $\mathcal{A}$ to be a dummy adversary that forwards messages between $\mathcal{E}'$ and $\mathcal{S}'$.

This concludes the proof of Theorem 19. Note that by Theorem 10, 16, and 18 it follows that $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$ (but that the converse is not true if $\mathcal{P}$ is not network predictable). Hence, Claim III implies Claim II. It is open whether $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ if $\mathcal{P}$ is network predictable.

5.5 Making the Simulator the Master Process

Theorems 18 and 19 show that SS (SBB) and UC (WBB) are not equivalent if the adversary may play the role of the master process and the environment is restricted to be a decision process even if the real protocol is network predictable. As mentioned, the reason for this is that to show that UC implies SS, we want to use the ideal adversary in UC as the simulator in SS. However, in SS the simulator has to be a regular process expression while in UC the ideal adversary may be a master process expression. In general, it is not possible to turn a master process expression into a regular process expression without changing the behavior of the overall system.

It is tempting to think that allowing the simulator to play the role of the master process would solve the above problem, and thus, would make UC (WBB) and SS (SBB) equivalent even if the environment may only be a decision process. In this section, we will see that this is not so. In a nutshell, the reason for this is that in UC the run time of the ideal adversary may depend on the run time of the real adversary while the run time of the simulator in SS and BB has to be independent of the run time of the real adversary and the environment, and therefore, the simulator can be exhausted by these entities.

However, WBB is equivalent to UC if the simulator may play the role of the master process both in case the environment is the master process and in case the environment is restricted to be a decision process.

Recall that we have defined variants SSsim, SBBsim, and WBBsim of SS, SBB, and WBB in which the simulator may play the role of the master process in Definition 7.

We note that certain variants of SSsim, SBBsim, and WBBsim do not make sense as all IO-compatible protocols would be related:

Remark 20. *We have that* $\mathtt{SSsim}_{(\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$, $\mathtt{SSsim}_{(\mathbf{M}\setminus\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$, $\mathtt{SBBsim}_{(\mathbf{R},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$, $\mathtt{SBBsim}_{(\mathbf{R},\mathbf{M}\setminus\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$, $\mathtt{WBBsim}_{(\mathbf{R},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$, *and* $\mathtt{WBBsim}_{(\mathbf{R},\mathbf{M}\setminus\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *are true for every IO-compatible protocols* $\mathcal{P}$ *and* $\mathcal{F}$ *since the left hand-side of* $\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E}' \mid \mathcal{S} \mid \mathcal{F}$ *and* $\mathcal{E} \mid \mathcal{A} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{A}' \mid \mathcal{S} \mid \mathcal{F}$ *do not contain* `start`*, and thus, no computation can take place. Consequently, if* $\mathcal{S}$ *does "nothing", then the process expressions on both sides are indistinguishable.*

Theorem 21.
1. *There are no IO-compatible protocols* $\mathcal{P}$ *and* $\mathcal{F}$ *such that:* $\mathtt{SSsim}_{(\mathbf{M}\setminus\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$, $\mathtt{SBBsim}_{(\mathbf{M},\mathbf{M}\setminus\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$, $\mathtt{SBBsim}_{(\mathbf{M},\mathbf{M}\setminus\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$, *or* $\mathtt{SBBsim}_{(\mathbf{R},\mathbf{M}\setminus\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.
2. *For every* $\mathcal{P},\mathcal{F} \in \mathbf{R}$*:* $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SSsim}_{(\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBBsim}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBBsim}_{(\mathbf{R},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBBsim}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$.
3. *For every* $\mathcal{P},\mathcal{F} \in \mathbf{R}$*:* $\mathtt{WBBsim}_{(\mathbf{R},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{WBBsim}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.
4. *For every* $\mathcal{P},\mathcal{F} \in \mathbf{R}$*:* $\mathtt{WBBsim}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{WBBsim}_{(\mathbf{M}\setminus\mathbf{R},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$.

5. *The notions in 3. imply those in 4.*

Proof. Statement 1. Assume that there exists $\mathcal{P}$ and $\mathcal{F}$ such that $\mathtt{SSsim}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$. Hence, there exists $\mathcal{S} \in (\mathbf{M}\backslash\mathbf{R})\text{-Adv}_{\mathcal{P}}(\mathcal{F})$ with $\mathcal{E} \upharpoonright \mathcal{P} \equiv \mathcal{E}' \upharpoonright \mathcal{S} \upharpoonright \mathcal{F}$. Let $q(\mathbf{n}) = \mathtt{comsize}(\mathcal{S})$. Now, to distinguish $\mathcal{P}$ from $\mathcal{S} \upharpoonright \mathcal{F}$, we define an $\mathcal{E}$ that does the following: $\mathcal{E}$ triggers itself via $\mathtt{start}$ $q(\mathbf{n})+1$ times. (It is straightforward to formulate $\mathcal{E}$ as a process expression.) If it interacts with $\mathcal{P}$, then $\mathcal{E}$ is in fact triggered $q(\mathbf{n})+1$ times, and in this case, $\mathcal{E}$ outputs 1 on $\mathtt{decision}$. If $\mathcal{E}$ ($\mathcal{E}'$) interacts with $\mathcal{S} \upharpoonright \mathcal{F}$, then $\mathcal{E}'$ must be triggered through $\mathcal{S}$ via $\mathtt{start}'$. However, since the communication size of $\mathcal{S}$ is $q(\mathbf{n})$, $\mathcal{S}$ cannot trigger $\mathcal{E}$ $q(\mathbf{n})+1$ times, and thus, $\mathcal{E} \upharpoonright \mathcal{P} \not\equiv \mathcal{E}' \upharpoonright \mathcal{S} \upharpoonright \mathcal{F}$. The argument for the variants of SBBsim is similar.

Statement 2. This is an immediate consequence of the first statement, Theorem 16, and Theorem 18.

Statement 3. The implications from right to left are obvious. To see that $\mathtt{WBBsim}_{(\mathbf{R},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$, first note that by Theorem 16 $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ is equivalent to $\mathtt{WBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$. Now, if $\mathtt{WBBsim}_{(\mathbf{R},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ is the variant of WBBsim where the the simulator may only be master if the adversary is, then it immediately follows that the simulator has to be regular. Hence, $\mathtt{WBBsim}_{(\mathbf{R},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{WBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ (and thus $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$). In case, we consider the variant of $\mathtt{WBBsim}_{(\mathbf{R},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ where the simulator may be master independent of whether the adversary is master, we obtain that if the simulator is master even though the adversary is not, then the environment can be a master and can exhaust the simulator just as shown in statement 1. Consequently, this case can not occur. And hence, the simulator has to be regular if the adversary is.

Statement 4. The first equivalence follows from the fact that if the adversary is a regular process expression, then no computation can take place. It is also clear that WBBsim implies UC since the real adversary in parallel with the simulator provides the ideal adversary needed for UC. The implication in the other direction is more interesting:

1. Assume that $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$.
2. The definition yields that $\mathcal{P}$ and $\mathcal{F}$ are IO-compatible and:
   $\forall \mathcal{A} \in \mathbf{M}\text{-Adv}(\mathcal{P}).\exists \mathcal{I} \in \mathbf{M}\text{-Adv}_{\mathcal{A} \upharpoonright \mathcal{P}}(\mathcal{F}).\forall \mathcal{E} \in \mathbf{D}\text{-Env}(\mathcal{A} \upharpoonright \mathcal{P}) : \mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{I} \upharpoonright \mathcal{F}$.
3. Choosing $\mathcal{A} = \mathcal{D}^{\mathtt{io}}_M = \mathcal{D}^{\mathtt{io}}_M(\mathsf{C}^{\mathtt{net}}_{\mathtt{in}}(\mathcal{P}), \mathsf{C}^{\mathtt{net}}_{\mathtt{out}}(\mathcal{P}), q(\mathbf{n}))$ for some $q(\mathbf{n})$ we obtain:
   $\exists \mathcal{S}_{q(\mathbf{n})} = \mathcal{I} \in \mathbf{M}\text{-Adv}_{\mathcal{D}^{\mathtt{io}}_M \upharpoonright \mathcal{P}}(\mathcal{F}).\forall \mathcal{E} \in \mathbf{D}\text{-Env}(\mathcal{D}^{\mathtt{io}}_M \upharpoonright \mathcal{P}) : \mathcal{E} \upharpoonright \mathcal{D}^{\mathtt{io}}_M \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{S}_{q(\mathbf{n})} \upharpoonright \mathcal{F}$.
4. Let $\mathcal{A} \in (\mathbf{M}\backslash\mathbf{R})\text{-Adv}(\mathcal{P})$, $q(\mathbf{n}) = \mathtt{comsize}(\mathcal{A})(\mathbf{n}) + \mathtt{comsize}(\mathcal{P})(\mathbf{n}) + 1$, choose $\mathcal{S}_{q(\mathbf{n})}$ as in 3. and let $\mathcal{E} \in \mathbf{D}\text{-Env}(\mathcal{A} \upharpoonright \mathcal{P})$. We obtain:

   $$\begin{array}{rcll} \mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} & \equiv & \mathcal{E} \upharpoonright \mathcal{A}'_M \upharpoonright \mathcal{D}^{\mathtt{net}}_M \upharpoonright \mathcal{P} & \text{(MASTER-ADV-FORWARDER)} \\ & \equiv & \mathcal{E} \upharpoonright \mathcal{A}''_M \upharpoonright \mathcal{D}^{\mathtt{io}}_M \upharpoonright \mathcal{P} & \text{(RENAME)} \\ & \equiv & \mathcal{E} \upharpoonright \mathcal{A}''_M \upharpoonright \mathcal{S}_{q(\mathbf{n})} \upharpoonright \mathcal{F} & (\mathcal{E} \upharpoonright \mathcal{A}''_M \in \mathbf{D}\text{-Env}(\mathcal{D}^{\mathtt{io}}_M \upharpoonright \mathcal{P}), 3.) \\ & \equiv & \mathcal{E} \upharpoonright \mathcal{A}' \upharpoonright \mathcal{S}'_{q(\mathbf{n})} \upharpoonright \mathcal{F} & \text{(RENAME)} \end{array}$$

   where $\mathcal{A}'_M$ is defined as in MASTER-ADV-FORWARDER, $\mathcal{D}^{\mathtt{io}}_M$ and $\mathcal{A}''_M$ are obtained from $\mathcal{D}^{\mathtt{net}}_M$ and $\mathcal{A}'_M$, respectively, by declaring the renamed network channel c' of $\mathcal{P}$ to be IO channels, $\mathcal{A}'$ is defined as in the definition of WBBsim, and $\mathcal{S}'_{q(\mathbf{n})}$ is obtained from $\mathcal{S}_{q(\mathbf{n})}$ by declaring the IO channels c' to be network channels and renaming them to c according to $\mathcal{P}$.

5. Observe that $\mathcal{S}'_{q(\mathbf{n})}$ is adversarially valid for $\mathcal{F}$ and that $\mathcal{P}$ and $\mathcal{S}'_{q(\mathbf{n})} \upharpoonright \mathcal{F}$ are compatible except that $\mathcal{S}'_{q(\mathbf{n})}$ contains $\mathtt{start}$ and $\mathtt{start}'$. Also, $\mathcal{S}'_{q(\mathbf{n})}$ only depends on $\mathcal{F}$, $\mathcal{P}$, and (the communication size of) $\mathcal{A}$. Consequently, 4. implies $\mathtt{WBBsim}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$ (for both variants of WBBsim).

Statement 5. It suffices to observe that $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{WBBsim}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$, which is obvious. □

5.6 Summary of the Relationships

In this section, we summarize the results proved in the previous sections. We have four classes of pairwise equivalent (variants of) security notions. In the following four corollaries we present these classes. We then study the relationships between these classes. All results are also depicted in Figure 1. In this figure, (non-)implications that immediately follow from the ones depicted are not drawn.

The first class, which we call SS/SBB, consists of all variants of strong simulatability and strong black-box simulatability. There equivalence follows immediately from Theorem 16, 18, and 21.

Corollary 22. *All security notions in the class SS/SBB are equivalent, i.e., for every $\mathcal{P},\mathcal{F} \in \mathbf{R}$, we have:*
$\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SSsim}_{(\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBB}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBB}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBB}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBBsim}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBBsim}_{(\mathbf{R},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{SBBsim}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$.

The second class, which we call UC/WBB$_{env}$, consists of all variants of universal composability and weak black-box simulatability where the environment may be a master process. There equivalence follows immediately from Theorem 16 and 21.

Corollary 23. *All security notions in the class UC/WBB$_{env}$ are equivalent, i.e., for every $\mathcal{P},\mathcal{F} \in \mathbf{R}$, we have:*
$\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{WBB}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{WBBsim}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{WBBsim}_{(\mathbf{R},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.

The third class, which we call UC/WBB$_{sim}$, consists of all variants of universal composability and weak black-box simulatability where the simulator is a master process and the environment is restricted to be a decision process. There equivalence follows immediately from Theorem 18 and 21.

Corollary 24. *All security notions in the class UC/WBB$_{sim}$ are equivalent, i.e., for every $\mathcal{P},\mathcal{F} \in \mathbf{R}$, we have:*
$\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{UC}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{M}\backslash\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{UC}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{M}\backslash\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$[6] *iff* $\mathtt{WBBsim}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$ *iff* $\mathtt{WBBsim}_{(\mathbf{M}\backslash\mathbf{R},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$.

The fourth class, which we call WBB$_{adv}$, consists of all variants of weak black-box simulatability where the simulator is a regular process and the environment is restricted to be a decision process. There equivalence follows immediately from Theorem 21.

[6] Note that since real and ideal adversary have to be master process, the environment can not be a master process.

Corollary 25. *All security notions in the class* WBB_{adv} *are equivalent, i.e., for every* $\mathcal{P}, \mathcal{F} \in \mathbf{R}$, *we have:* $\mathtt{WBBsim}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P}, \mathcal{F})$ *iff* $\mathtt{WBBsim}_{(\mathbf{M}\setminus\mathbf{R},\mathbf{R},\mathbf{D})}(\mathcal{P}, \mathcal{F})$.

We now summarize some of the basic relationships between the different classes.

Given a class $\mathbf{C}$ of regular process expressions, we write, for instance, $\mathsf{SS/SBB} \Rightarrow \mathsf{UC/WBB}_{env}$ *for* $\mathbf{C}$ to say that, for every $\mathcal{P}, \mathcal{F} \in \mathbf{C}$, if $\mathcal{P}$ and $\mathcal{F}$ are related with respect to some security notion in $\mathsf{SS/SBB}$ (since all of the security notions in one class are equivalent, it does not matter which one is chosen), then they are also related with respect to (all of) the security notions in $\mathsf{UC/WBB}_{env}$. In particular, $\mathsf{UC/WBB}_{sim} \not\Rightarrow \mathsf{WBB}_{adv}$ for $\mathbf{C}$ means that there exist $\mathcal{P}, \mathcal{F} \in \mathbf{C}$ such that $\mathcal{P}$ and $\mathcal{F}$ are related w.r.t. the security notions in $\mathsf{UC/WBB}_{sim}$ but not w.r.t. those in WBB_{adv}. In case $\mathbf{C} = \mathbf{R}$, we will omit $\mathbf{C}$ and simply say, for instance, $\mathsf{SS/SBB} \Rightarrow \mathsf{UC/WBB}_{env}$.

Corollary 26.

1. $\mathsf{SS/SBB} \Rightarrow \mathsf{UC/WBB}_{env}$.
2. $\mathsf{UC/WBB}_{env} \Rightarrow \mathsf{WBB}_{adv}$.
3. $\mathsf{WBB}_{adv} \Rightarrow \mathsf{UC/WBB}_{sim}$.
4. $\mathsf{UC/WBB}_{sim} \not\Rightarrow \mathsf{WBB}_{adv}$.[7]
5. *In particular:*
   (a) $\mathsf{UC/WBB}_{env} \Rightarrow \mathsf{UC/WBB}_{sim}$.
   (b) $\mathsf{UC/WBB}_{sim} \not\Rightarrow \mathsf{UC/WBB}_{env}$.
   (c) $\mathsf{SS/SBB} \Rightarrow \mathsf{UC/WBB}_{sim}$.
   (d) $\mathsf{SS/SBB} \not\Leftarrow \mathsf{UC/WBB}_{sim}$.

Proof. The first implication follows immediately from Theorem 16. The second implication was shown in Theorem 18, while the third one was proved in Theorem 21. Finally, 4. was stated in Theorem 19. The statements in 5. immediately follow from the previous ones. □

We emphasize the following:

Remark 27. *The equivalences among the security notions in the different classes—*$\mathsf{SS/SBB}$*,* $\mathsf{UC/WBB}_{env}$*,* WBB_{adv}*, and* $\mathsf{UC/WBB}_{sim}$*— as well as the relationships between these classes, as stated in Corollary 26, are proved based on only quite basic properties of the computational model, namely, the axioms listed in Section 5.1, excluding FORWARDER, plus the assumption that the runtime of processes are polynomially bounded in the security parameter (except for guards), which is the case for the models proposed in [5, 9, 27]. Also, the example showing that* $\mathsf{UC/WBB}_{sim}$ *does not imply* WBB_{adv} *is quite basic. This allows us to carry over our results to other models (see Section 6).*

It is open whether or not WBB_{adv} implies $\mathsf{UC/WBB}_{env}$. However, from Corollary 26 and Corollary 28, it follows that WBB_{adv} does not imply $\mathsf{SS/SBB}$ for protocols that are not network predictable, i.e., protocols that do not satisfy FORWARDER. (Otherwise, $\mathsf{SS/SBB}$ and $\mathsf{UC/WBB}_{env}$ would be equivalent even for protocols that do not satisfy FORWARDER, which is a contradiction to Corollary 28 below).

[7] The real (and ideal) protocol chosen to prove this statement is network predictable, i.e., it satisfies FORWARDER.

In Theorem 10, we have shown that for $\mathsf{UC/WBB}_{env}$ to imply $\mathsf{SS/SBB}$ it is necessary that FORWARDER is satisfied. This allows us to characterize when $\mathsf{UC/WBB}_{env}$ and $\mathsf{SS/SBB}$ are equivalent. We write $\mathsf{UC/WBB}_{env} \Leftrightarrow \mathsf{SS/SBB}$ for $\mathbf{C}$ if $\mathsf{UC/WBB}_{env} \Rightarrow \mathsf{SS/SBB}$ for $\mathbf{C}$ and $\mathsf{SS/SBB} \Rightarrow \mathsf{UC/WBB}_{env}$ for $\mathbf{C}$.

Corollary 28. *Let* $\mathbf{C}$ *be a class of regular process expressions closed under renaming of channels (in the same sense used in Theorem 10). Then,*

$$\mathsf{UC/WBB}_{env} \Leftrightarrow \mathsf{SS/SBB} \textit{ for } \mathbf{C} \quad \textit{iff} \quad \textit{FORWARDER}(\mathcal{P}) \textit{ for every } \mathcal{P} \in \mathbf{C}.$$

Proof. The only-if direction immediately follows from Theorem 10. For the if direction, first note that by Corollary 26 we know that $\mathsf{SS/SBB} \Rightarrow \mathsf{UC/WBB}_{env}$ for $\mathbf{R}$. Given that FORWARDER($\mathcal{P}$), we obtain that $\mathsf{UC/WBB}_{env} \Rightarrow \mathsf{SS/SBB}$ for $\mathbf{C}$ by Theorem 16. □

We emphasize:

Remark 29. *The proof of Corollary 28 only uses very basis properties (axioms) which should be satisfied in most or all computational models. Hence, the corollary should carry over to such models.*

Together with Proposition 15, Corollary 28 implies:

Corollary 30. *Let* $\mathbf{C}$ *be the class of standard protocols (see Definition 14). Then,*

$$\mathsf{UC/WBB}_{env} \Leftrightarrow \mathsf{SS/SBB} \textit{ for } \mathbf{C}.$$

We note that this corollary does not hold for the class of all regular process expressions as there exist regular process expression which are not network predictable (Proposition 12). By Corollary 28, for those protocols, $\mathsf{UC/WBB}_{env}$ does not imply $\mathsf{SS/SBB}$.

Corollary 30 tells us that if the environment may play the role of the master process, then for standard protocols (i.e., the class of protocols considered in the computational models by Pfitzmann and Waidner [27] and Canetti [9]) strong simulatability/strong black-box simulatability and universal composability/weak black-box simulatability are equivalent notions in SPPC. The main reason is that in SPPC, for standard protocols, the axiom FORWARDER is true (Proposition 15). Here we use that processes correspond to IO automata/ITMs *with* guards. Without guards, the proposition would not hold true.

6 Implications for Other Models

In this section, we discuss the implications of our results in SPPC for the PIOA and PITM models.

6.1 The PIOA Model and Variants

We refer the reader to [5, 27] for a detailed description of the PIOA model. We examine the relationships between the security notions for PIOA. It turns out that the security notions UC and WBB are not equivalent in PIOA in case the environment may play the role of the master process. This seems counterintuitive and suggests to slightly modified or restrict PIOA. We call the new version the buffer-free version of PIOA (BFPIOA). This version behaves just like SPPC as far as the relationships between the security notions are concerned.

Before studying the relationships between the security notions, we go through the axioms listed in Section 5.1 and see which ones are satisfied and which ones are not. This will allow us to immediately carry over some of the results for SPPC. We then consider the relationships which were proved for SPPC based on axioms not satisfied in PIOA. All of the following is independent of whether or not buffers can be queried an unbounded number of times.

On the validity of the axioms in Section 5.1. It is easy to see that the axioms COM, ASC, TRN, SYM, RENAME, RENAME-START, MMD-INCLUSION, and MD-INCLUSION are satisfied in PIOA. We will see that MASTER-S-FORWARDER, and MASTER-ADV-FORWARDER are satisfied as well.

However, the axioms REG-S-FORWARDER, REG-ADV-FORWARDER, and FORWARDER are not satisfied. For all dummy axioms we assume that the environment/adversary connects only to the channels (or ports, to use the terminology of the PIOA model) of the dummy process. That is, the environment/adversary is not allowed to access the channels of the protocol directly by renaming of channels. Although to some extent this is allowed in the PIOA model, it does not effect the results presented here.

Proposition 31. *The axioms REG-S-FORWARDER, REG-ADV-FORWARDER, and FORWARDER are not satisfied in the PIOA model.*

PROOF. The following example shows that REG-S-FORWARDER is not satisfied. The same example works for REG-ADV-FORWARDER and FORWARDER. The example uses that in the PIOA model entities always communicate through buffers which have to be triggered to deliver messages and which may be triggered by machines (typically the adversary/environment) other than those who write messages into the buffer.

Let $\mathcal{P}$ consist of one IO automaton M which receives a bit on an IO channel and forwards it on a network channel, i.e., writes it into a buffer connected to the adversary/environment. We assume that the buffer is scheduled by the adversary/environment. In what follows, we argue that REG-S-FORWARDER does not hold for $\mathcal{P}$.

If the environment (for which we may assume that it plays the role of the master process) sends a bit to $\mathcal{P}$, then $\mathcal{P}$ outputs the bit on the network channel and according to the computational model of PIOA, this bit is written into the buffer. Next the environment is triggered and it will trigger the buffer in which it expects to find the bit send to $\mathcal{P}$ via the IO channel. In case there is no dummy between $\mathcal{P}$ and the environment, the environment will obtain the bit. Otherwise, if the environment and $\mathcal{P}$ are separated by a dummy, then the environment triggers the buffer which "sits" in between the environment and the dummy, and this buffer does not contain the bit since the dummy was never activated, and thus, could not write into this buffer. Thus, the environment can distinguish whether it only interacts with the protocol $\mathcal{P}$ or with the dummy and the protocol. Consequently, REG-S-FORWARDER does not hold in PIOA. □

We note that by Remark 13, the axiom FORWARDER fails in PIOA (for the same reason it fails in SPPC) even if all machines trigger their own buffers.

The above example does not work for the axioms MASTER-S-FORWARDER and MASTER-ADV-FORWARDER because after $\mathcal{P}$ wrote the bit into the network buffer, the dummy will be triggered next as it is the master process. Hence, the dummy can write the bit into the buffer that sits in between the environment and the dummy, and then can activate the environment. More

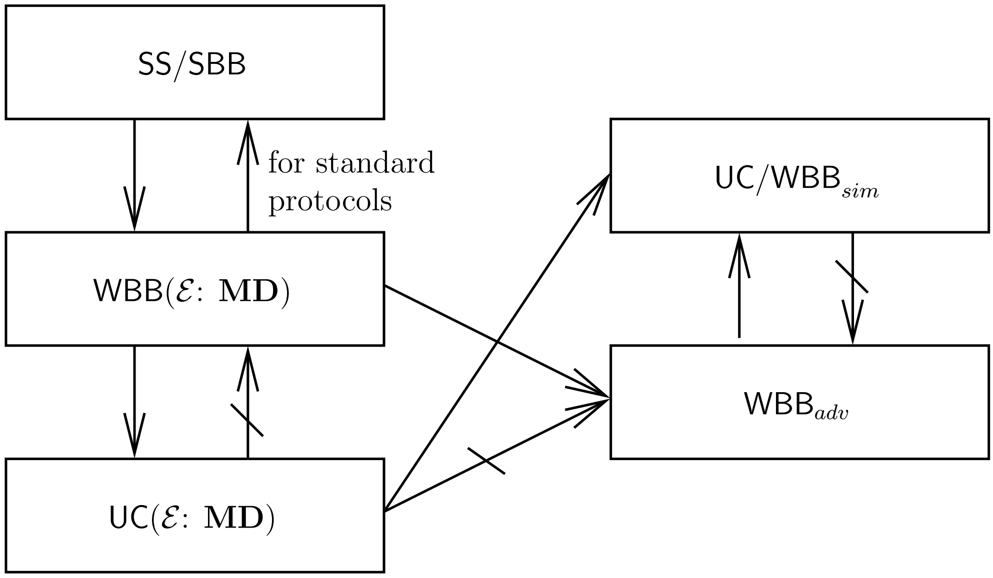

Figure 3: Relationships of the security notions in PIOA

generally, since the dummy is the master it can copy all messages written by $\mathcal{P}$ to the buffers sitting in between the dummy and the protocol into the buffers which sit in between the environment and the dummy, and only then active the environment. Therefore, MASTER-S-FORWARDER and MASTER-ADV-FORWARDER are satisfied in PIOA.

On the relationships of the security notions. In what follows, let $\mathsf{UC}(\mathcal{E}\colon \mathbf{MD})$ consist of the UC notions of the class $\mathsf{UC/WBB}_{env}$ and let $\mathsf{WBB}(\mathcal{E}\colon \mathbf{MD})$ consist of the WBB and WBBsim notions in this class. That is, $\mathsf{UC/WBB}_{env}$ is the union of $\mathsf{UC}(\mathcal{E}\colon \mathbf{MD})$ and $\mathsf{WBB}(\mathcal{E}\colon \mathbf{MD})$.

The relationships between the security notions in the PIOA model are summarized in Figure 3. The most interesting relationship is that of $\mathsf{UC}(\mathcal{E}\colon \mathbf{MD})$ and $\mathsf{WBB}(\mathcal{E}\colon \mathbf{MD})$. While in SPPC these classes are equivalent, this is not the case in the PIOA model. The reason for this is that to prove the equivalence the axiom REG-ADV-FORWARDER is necessary, which, however, does not hold in the PIOA model (see below for more details). In what follows, the relationships depicted in Figure 3 are justified.

The security notions in the classes depicted in Figure 3 (i.e., those in the single boxes) are equivalent because to prove these equivalences only axioms are necessary which are valid in PIOA. And hence, the proofs can be carried out just as for SPPC. (One exception are the equivalences of the notions in $\mathsf{SS/SBB}$. Here, for SPPC, we used the axiom REG-S-FORWARDER which, as shown above, does not hold in the PIOA model. However, this axiom can easily be avoided in the equivalence proof.)

In what follows, we show that $\mathsf{UC}(\mathcal{E}\colon \mathbf{MD})$ does not imply $\mathsf{WBB}(\mathcal{E}\colon \mathbf{MD})$. The other relationships depicted in Figure 3 are obtained from the results obtained for SPPC and the fact that the axioms used to prove the implications are also valid in PIOA. The non-implications carry over from SPPC as the same counterexamples can be used.

To prove that $\mathsf{UC}(\mathcal{E}\colon \mathbf{MD})$ does not imply $\mathsf{WBB}(\mathcal{E}\colon \mathbf{MD})$ we show that $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ does not imply $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ for the following protocols $\mathcal{P}$ and $\mathcal{F}$.

Let $\mathcal{P}$ consist of one IO automaton M which receives a bit on an IO channel and forwards it on a network channel, i.e., writes it into a buffer connected to the adversary/environment. We assume that the buffer is scheduled by the adversary/environment. (This is the same protocol as above.)

Let $\mathcal{F}$ consist of one IO automaton M' which, just as M, has an IO channel and a network channel where the corresponding buffer is scheduled by the adversary. In addition, M' has a secure channel to itself. (Alternatively, one could introduce another machine M'' and two channels, one from M' to M'' and one from M'' to M', controlled by the sending machine, respectively.) M' works as follows: It receives a bit b_1 from the environment, generates a random bit b_2, and writes the two messages (b_1, "environment") and (b_2, "random") into the network buffer. M' chooses the order in which these messages are written into the buffer uniformly at random. Also, the messages are written into two different cells of the network buffer. This is possible by using the secure channel: First, M' writes the first message into the buffer, then M' uses the secure channel to trigger itself, and third M' writes the second message into the buffer.

Claim I. $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.

Proof sketch of Claim I. The ideal adversary simply simulates the real adversary. In case the real adversary triggers the buffer to M to obtain the first message, the ideal adversary would trigger the buffer to M' two times to obtain both messages (if any) and would only use b_1 to simulate the real adversary. The tag "environment" tells the ideal adversary which of the two bits to use.

Claim II. $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ does not hold.

Proof sketch of Claim II. Assume that WBB holds for the real protocol $\mathcal{P}$ and the ideal protocol $\mathcal{F}$. We distinguish three cases and lead them to a contradiction.

1. The simulator $\mathcal{S}$ is "empty", i.e., the ideal adversary $\mathcal{A}'$ (obtained by renaming channels of the real adversary $\mathcal{A}$) connects to the network buffer of M'. This obviously does not work because the network buffers of M and M' contain different information such that it is easy to specify $\mathcal{A}$, $\mathcal{A}'$, and an environment that tell $\mathcal{P}$ and $\mathcal{F}$ apart.

2. The simulator $\mathcal{S}$ connects to the network buffer of M' but the ideal adversary $\mathcal{A}'$ controls the clock channel of this buffer. This also does not work. Let the real adversary $\mathcal{A}$ be one that is triggered by the environment to read out the bit of the network channel. More precisely, $\mathcal{A}$ triggers the buffer to read out the first entry and forwards it to the environment. If $\mathcal{A}'$ triggers the first entry of the network channel of M', then this entry would be given to $\mathcal{S}$. In half of the cases this entry is (b_2,random), and thus, the simulator cannot figure out b_1. Hence, the bit forwarded by $\mathcal{S}$ to $\mathcal{A}'$ is wrong in half of the cases, and thus, so is the bit forwarded by $\mathcal{A}'$ to the environment.

3. The simulator $\mathcal{S}$ completely controls the network buffer of M'. In this case, there must be a buffer from $\mathcal{S}$ to $\mathcal{A}'$ controlled by $\mathcal{A}'$. Now the problem is that after $\mathcal{F}$ wrote the two messages into the buffer (which "sits" between $\mathcal{S}$ and $\mathcal{F}$), the environment, which is assumed to be a master scheduler, is scheduled and asks $\mathcal{A}$ ($\mathcal{A}'$) to deliver the bit from the network channel. The adversary $\mathcal{A}$ does this by triggering the network channel and forwarding the bit to the environment. If $\mathcal{A}'$ does the same, the buffer will be empty since the simulator was never triggered, and thus, could not write anything into the buffer between $\mathcal{A}'$ and $\mathcal{S}$. (We note that this situation corresponds to the failure of REG-ADV-FORWARDER explained above.)

A similar argument works for $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{D})}(\mathcal{P},\mathcal{F})$. The only difference is that we assume the adversary to be the master process.

The failure of the equivalences and the failure of the axioms REG-S-FORWARDER and REG-ADV-FORWARDER seem counterintuitive. The problem vanishes if the PIOA model is modified so that machines always trigger their own buffers. In effect, this is equivalent to not having buffers at all, which is why we call this fragment of the PIOA model the *buffer-free PIOA model (BFPIOA)*. This fragment is essentially as expressive as PIOA and this fragment can easily be embedded into SPPC. In particular, *all* axioms (except the forwarder property of the previous section) are satisfied in BFPIOA and the examples used to prove separation results can also be expressed in BFPIOA. As mentioned in Section 2, starting from the work [6] PIOA (and thus, BFPIOA) has a restricted form of guards. Similar to SPPC, this mechanism suffices to satisfy the forwarder property for standard protocols, but just as in SPPC, there are protocols expressible in BFPIOA which do not satisfy this property. In summary, we obtain for BFPIOA exactly the same relationships as for SPPC (see Figure 1).

In [27], the security notions $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$ and $\mathtt{SBB}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P},\mathcal{F})$ were introduced for the PIOA model, while in [4] the notions $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ and $\mathtt{SBB}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ were considered. Our results clarify the relationships between these security notions: while the two variants of SBB are equivalent (they both belong to the class $\mathsf{SS/SBB}$), these notions are different from the two variants of UC. Also, the two variants of UC are not equivalent. Our results contradict the claim in [4] that $\mathtt{SBB}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ and $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ are equivalent.

6.2 The PITM Model

The PITM model [9] is tailored towards defining UC where the environment is a master process and the adversaries are regular processes i.e., $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$. Depending on which entities are involved, different computational models are defined: the real model (involving the environment, the real adversary, and the real protocol), the ideal model (involving the environment, the ideal adversary, and the ideal functionality together with dummy parties), and the hybrid model which is a combination of the previous two models.

Therefore, it is not immediately clear how the security notions SS, SBB, and WBB, which involve a simulator, would be defined in PITM. Different variants are possible, and as we have seen, differences in the definitions may affect the relationships between the security notions. It is out of the scope of this paper, to extend PITM in order to define SS, SBB, and WBB. However, general points can be made.

The version of PITM as introduced in [9] does not have a mechanism, such as guards of SPPC, that would allow the dummy axiom to be satisfied. As shown, without this axiom being satisfied, UC does not imply SBB (SS).

We finally note that in [9], Canetti introduces a special case of UC where the adversary is restricted to be a dummy which merely forwards messages between the environment and the parties. Canetti proves UC and this notion equivalent. This notion can easily be stated in SPPC and proved equivalent to UC along the lines of the proof which shows that $\mathtt{UC}_{(\mathbf{R},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$ implies $\mathtt{WBB}_{(\mathbf{M},\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$.

7 Reactive Simulatability and Extensions of SPPC

In this section, we consider another security notion, called *reactive simulatability* in [5] and *security with respect to specialized simulators* in [10]. This notion has not drawn as much attention as the other notions studied in the present work because a general composition theorem for composing a polynomial number of copies of protocols along the lines of [9] or the present work was not known, and in fact, as recently shown is not possible in case the environment is uniform and strict polynomial-time [20]. Therefore, in the previous sections, we have concentrated on the other security notions and only very briefly cover reactive simulatability here. In our terminology, reactive simulatability is defined as follows:

Reactive Simulatability: $\mathtt{RS}_{(\mathbf{A},\mathbf{I},\mathbf{E})}(\mathcal{P},\mathcal{F})$ iff $\mathcal{P}$ and $\mathcal{F}$ are IO-compatible and for every $\mathcal{A} \in \mathbf{A}\text{-Adv}(\mathcal{P})$ and for every $\mathcal{E} \in \mathbf{E}\text{-Env}(\mathcal{A} \upharpoonright \mathcal{P})$ there exists $\mathcal{I} \in \mathbf{I}\text{-Adv}_{\mathcal{A} \upharpoonright \mathcal{P}}(\mathcal{F})$ such that $\mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{I} \upharpoonright \mathcal{F}$.

The only difference between reactive simulatability and universal composability (UC) is that for the former notion the ideal adversary is allowed to depend on the environment.

It has been shown by Canetti [8] that reactive simulatability is equivalent to UC for non-uniform environments whose runtime may depend on the lengths of their input on the input tape. The fact that UC implies reactive simulatability follows simply from the order of quantifiers. For the other direction, one considers a "universal" environment which interprets (part of its) input as an encoding of another environment and simulates this environment. In this way, one effectively can quantify over all environments, and hence, switch the order of quantification over environments and ideal adversaries. When allowing the runtime of the environment to depend on the length of the input on its `start` channel, then in SPPC we obtain the same result. Moreover, one can show that with this extension of SPPC, the security notions in the classes $\mathsf{UC/WBB}_{env}$ and $\mathsf{UC/WBB}_{sim}$ are equivalent, respectively; the axioms used to prove these relationships also hold true for the extension of SPPC.

Hofheinz and Unruh [19] show that for strictly polynomial-time, uniform environments reactive simulatability does not imply universal composability. While this result is shown in the PIOA model, it immediately carries over to SPPC in case the environment does not get auxiliary input, i.e., initially ε is written on the `start` channel.

8 On the Composability of Network Predictable Processes

In this section, we show that the property of being network predictable is composable, i.e., if regular process expressions are network predictable, then so is their parallel composition. This will allow us to carry over the composition theorem proved for strong simulatability, and thus, by Corollary 22, also blackbox simulatability to universal composability/weak black-box simulatability (Section 9).

We first consider the case of composing a constant number of network predictable processes and then the case of composing a polynomial number of copies of processes. While the construction presented here is tailored to the composition theorem, a similar construction shows that network predictable multi-party protocols with a polynomial number of participants can be defined from network predictable processes.

8.1 Composing a Constant Number of Network Predictable Processes

Lemma 32. *Let $\mathcal{P}_1, \ldots, \mathcal{P}_k \in \mathbf{R}$ be network predictable processes such that $\mathcal{P}_j$ is environmentally valid for $\mathcal{P}_{j+1} \upharpoonright \cdots \upharpoonright \mathcal{P}_k$ for every j. Then, $\mathcal{P}_1 \upharpoonright \cdots \upharpoonright \mathcal{P}_k$ is network predictable.*

Proof. We prove the lemma for $k = 2$. For $k > 2$, the statement follows by induction on k.

Assume that $\mathcal{P}_1$ and $\mathcal{P}_2$ are network predictable and that $\mathcal{P}_1$ is environmentally valid for $\mathcal{P}_2$, and thus, $\mathcal{P}_2$ is environmentally valid for $\mathcal{P}_1$. Let $\mathcal{D}_i$ be the dummy for $\mathcal{P}_i$ whose existence is guaranteed by FORWARDER($\mathcal{P}_i$). Since the set of network channels of $\mathcal{P}_1$ and $\mathcal{P}_2$ are disjoint, we may assume that this is also true for $\mathcal{D}_1$ and $\mathcal{D}_2$. Note that neither $\mathcal{D}_1$ nor $\mathcal{D}_2$ has IO channels. We want to show that $\mathcal{D}_1 \upharpoonright \mathcal{D}_2$ is a dummy for $\mathcal{P}_1 \upharpoonright \mathcal{P}_2$. Obviously, $\mathcal{D}_1 \upharpoonright \mathcal{D}_2 \in \mathbf{R}\text{-Adv}(\mathcal{P}_1 \upharpoonright \mathcal{P}_2)$ and the conditions on the external channels of $\mathcal{D}_1 \upharpoonright \mathcal{D}_2$ are satisfied as well. We need to show that

$$\mathcal{E} \upharpoonright \mathcal{P}_1 \upharpoonright \mathcal{P}_2 \equiv \mathcal{E}' \upharpoonright \mathcal{D}_1 \upharpoonright \mathcal{D}_2 \upharpoonright \mathcal{P}_1 \upharpoonright \mathcal{P}_2$$

for every $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P}_1 \upharpoonright \mathcal{P}_2)$ where $\mathcal{E}'$ is defined as in FORWARDER($\mathcal{P}_1 \upharpoonright \mathcal{P}_2$).

We have

$$\begin{array}{rcll} \mathcal{E} \upharpoonright \mathcal{P}_1 \upharpoonright \mathcal{P}_2 & \equiv & \mathcal{E}_{\mathcal{P}_2} \upharpoonright \mathcal{P}_1 \upharpoonright \mathcal{D}_2 \upharpoonright \mathcal{P}_2 & (i) \\ & \equiv & \mathcal{E}_{\mathcal{P}_2} \upharpoonright \mathcal{D}_2 \upharpoonright \mathcal{P}_2 \upharpoonright \mathcal{P}_1 & (ii) \\ & \equiv & \mathcal{E}_{\mathcal{P}_1,\mathcal{P}_2} \upharpoonright \mathcal{D}_2 \upharpoonright \mathcal{P}_2 \upharpoonright \mathcal{D}_1 \upharpoonright \mathcal{P}_1 & (iii) \\ & \equiv & \mathcal{E}_{\mathcal{P}_1,\mathcal{P}_2} \upharpoonright \mathcal{D}_1 \upharpoonright \mathcal{D}_2 \upharpoonright \mathcal{P}_1 \upharpoonright \mathcal{P}_2 & (iv) \\ & \equiv & \mathcal{E}' \upharpoonright \mathcal{D}_1 \upharpoonright \mathcal{D}_2 \upharpoonright \mathcal{P}_1 \upharpoonright \mathcal{P}_2 & (v) \end{array}$$

where in (i) $\mathcal{E}_{\mathcal{P}_2}$ is obtained from $\mathcal{E}$ by replacing all network channels c of $\mathcal{P}_2$ occurring in $\mathcal{E}$ by c' according to the renaming of network channels in $\mathcal{D}_2$. The equivalence holds due to the definition of $\mathcal{D}_2$ and the fact that $\mathcal{E} \upharpoonright \mathcal{P}_1 \in \mathbf{MD}\text{-Valid}(\mathcal{P}_2)$. Also note that $\mathcal{P}_1$ does not contain network channels of $\mathcal{P}_2$, and thus, no channels need to be renamed in $\mathcal{P}_1$ on the right hand-side of the equivalence. In (ii) we use commutativity (COM) and associativity (ASC) of parallel composition. In (iii) we consider $\mathcal{E}_{\mathcal{P}_2} \upharpoonright \mathcal{D}_2 \upharpoonright \mathcal{P}_2 \in \mathbf{MD}\text{-Valid}(\mathcal{P}_1)$ as a valid process for $\mathcal{P}_1$ and use the definition of $\mathcal{D}_1$ where $\mathcal{E}_{\mathcal{P}_1,\mathcal{P}_2}$ is obtained from $\mathcal{E}_{\mathcal{P}_2}$ by replacing every occurrence of network channels c of $\mathcal{P}_1$ by c' according to the renaming of network channels in $\mathcal{D}_1$. Again, note that this set of network channels is disjoint from the set of network channels of $\mathcal{P}_2$. In (iv) we again apply COM and ASC, and (v) is obtained by noting that $\mathcal{E}_{\mathcal{P}_1,\mathcal{P}_2} = \mathcal{E}'$. □

8.2 Composing a Polynomial Number of Copies of Network Predictable Processes

We want to show that if $\mathcal{P}$ is network predictable, then so are a polynomial number of copies of $\mathcal{P}$. Using the results of the previous section, this is then extended to the case where we have a polynomial number of copies of the protocols $\mathcal{P}_1, \ldots, \mathcal{P}_k$.

Similar to the setting of Canetti [9], we employ session identifiers (SIDs) to be able to distinguish between the different copies of $\mathcal{P}$. We also introduce a SID manager $\mathcal{M}_{\mathcal{P}}^{q(\mathbf{n})}$ which manages $q(\mathbf{n})$ copies of $\mathcal{P}$. To invoke a new copy of $\mathcal{P}$, a process needs to send an SID s to the manager who generates a new copy $\mathcal{P}_s$ of $\mathcal{P}$ provided that the SID has not been used before. The copy $\mathcal{P}_s$ is then accessible via s, i.e., if a process wants to send a message a to this copy via some channel c, then the message (s, a) would have to be sent on channel c since $\mathcal{P}_s$ only accepts messages where the first component is s. Conversely, every message sent by $\mathcal{P}_s$ is prefixed by s. In this way, SIDs allow to generate new (virtual) channels.

To be able to generate copies of $\mathcal{P}$, we consider what we call the *SID version* $\underline{\mathcal{P}}$ of $\mathcal{P}$. This SID version receives a SID as input, announces the SID on a network channel, acknowledges receipt of the SID, and generates a new copy of $\mathcal{P}$. Formally:

$$\underline{\mathcal{P}} \quad = \quad \mathtt{in}(\underline{\mathtt{sid}}, x).(\mathtt{out}(\mathtt{netsend}, x) \,||\, \mathtt{in}(\mathtt{netrec}, x).(\mathtt{out}(\mathtt{received}, x) \,||\, \mathcal{P}(x))) \tag{4}$$

where $\underline{\mathtt{sid}}$ and $\mathtt{received}$ are IO channels (and thus, low channels), $\mathtt{netsend}$ and $\mathtt{netrec}$ are network channels (and thus, low channels), and $\mathcal{P}(x)$ denotes the process expression obtain from $\mathcal{P}$ as follows: Every subexpression $\mathtt{in}(c,t).\mathcal{P}'$ of $\mathcal{P}$ is replaced by $\mathtt{in}(c,(x,t))$ where the M-term (x,t) accepts a message a if it is of the form (x,y) and y is accepted by t. Every subexpression $\mathtt{out}(c,T)$ of $\mathcal{P}$ is replaced by $\mathtt{out}(c,(x,T))$ where the C-term (x,T) returns (x,a) with probability p iff T returns a with probability p. Note that $\underline{\mathcal{P}} \in \mathbf{R}$ if $\underline{\mathcal{P}} \in \mathbf{R}$.

We often consider what we will call a *simple SID version* of a protocol where the SID is not announced on a network channel. That is, after receiving the SID on $\underline{\mathtt{sid}}$, the protocol immediately acknowledges receipt of the SID on $\mathtt{received}$.

We can now define the SID manager $\mathcal{M}_{\underline{\mathcal{P}}}^{q(\mathbf{n})}$ for $\mathcal{P}$. For simplicity we assume that $q(\mathbf{n}) \geq 1$, i.e., $q(i) \geq 1$ for every $i \geq 0$. This manager is a closed regular process expression which will run in parallel with $!_{q(\mathbf{n})}\ \underline{\mathcal{P}}$. Intuitively, the manager receives a SID, checks whether it exists already. If not, then a new copy of $\mathcal{P}$ is generated by sending the SID to $\underline{\mathcal{P}}$ and the process who called the manager is notified. In case the SID was not new, the manager notifies the failure to the calling process. Initially, the manager will always notify success since no SID has been sent so far. Formally:

$$\begin{aligned}
\mathcal{M}_{\underline{\mathcal{P}}}^{q(\mathbf{n})} \quad = \quad & \mathtt{in}(\mathtt{sid}, t_x).(\mathtt{out}(\underline{\mathtt{sid}}, x) \,||\, \mathtt{in}(\mathtt{received}, x).(\mathtt{out}(\mathtt{state}, x) \,||\, \mathtt{out}(\mathtt{done}, \mathtt{suc}))) \,|| \\
& !_{q(\mathbf{n})-1}\ \mathtt{in}(\mathtt{state}, y).(\\
& \quad \mathtt{in}(\mathtt{sid}, t_{x \notin y}).(\\
& \qquad \mathtt{out}(\underline{\mathtt{sid}}, x) \,||\, \mathtt{in}(\mathtt{received}, x).(\mathtt{out}(\mathtt{state}, (y,x)) \,||\, \mathtt{out}(\mathtt{done}, \mathtt{suc}))) + \\
& \quad \mathtt{in}(\mathtt{sid}, t_{x \in y}).(\\
& \qquad \mathtt{out}(\mathtt{state}, y) \,||\, \mathtt{out}(\mathtt{done}, \mathtt{fail})))
\end{aligned}$$

where $\mathtt{state}$ is a high channel and all other channels are low, the M-term t_x accepts a bit string only if it is of length $q(\mathbf{n})$ and then substitutes x with this bit string (in fact, any format would work as long as in a message of the form $x \cdot z$, the SID x can uniquely be extracted from such a message), and the M-term $t_{x \notin y}$ accepts a bit string a if a does not occur in the sequence y (the current state of the manager). In this case, x is substituted with a. Similarly, $t_{x \notin y}$ is an M-term which accepts a bit string a if a occurs in the sequence y. (The variable x does not need to be substituted by a since x is not used later on.) Note that the definition of $\mathcal{M}_{\underline{\mathcal{P}}}^{q(\mathbf{n})}$ is almost independent of $\underline{\mathcal{P}}$. We only require that $\underline{\mathtt{sid}}$ is the same channel as the one used by $\underline{\mathcal{P}}$ to receive SIDs.

We note that $\mathcal{M}_{\underline{\mathcal{P}}}^{q(\mathbf{n})}$ could be generalized in that this manager not only accepts session IDs on one channel, namely $\mathtt{sid}$, but on different channels. This would allow different external processes to use the same manager. Our results easily carry over to such managers.

We define

$$\mathcal{P}^{q(\mathbf{n})} = \mathcal{M}_{\underline{\mathcal{P}}}^{q(\mathbf{n})} \mid !_{q(\mathbf{n})}\ \underline{\mathcal{P}}$$

Lemma 33. *If $\mathcal{P} \in \mathbf{R}$ is a network predictable process, then so is $\mathcal{P}^{q(\mathbf{n})}$.*

Proof. Let $\mathcal{P} \in \mathbf{R}$ be a network predictable process. Thus, there exists a dummy $\mathcal{D} \in \mathbf{R}\text{-Adv}(\mathcal{P})$ which satisfies the conditions stipulated in FORWARDER($\mathcal{P}$). We define

$$\underline{\mathcal{D}} = \mathtt{in}(\mathtt{netsend}, x).(\mathtt{out}(\mathtt{netsend}', x) \mid\mid \mathtt{in}(\mathtt{netrec}', x).(\mathtt{out}(\mathtt{netrec}, x) \mid\mid \mathcal{D}(x)))$$

where $\mathcal{D}(x)$ is defined analogously to $\mathcal{P}(x)$. That is, $\underline{\mathcal{D}}$ first forwards the SID x coming from the protocol on $\mathtt{netsend}'$ and after receiving x again on $\mathtt{netrec}'$ forwards x and generates a copy of $\mathcal{D}$. Since $\mathcal{P}(x)$ is generated only after $\underline{\mathcal{P}}$ has received the responds on $\mathtt{netrec}$, it suffices for $\underline{\mathcal{D}}$ to generate a copy $\mathcal{D}(x)$ of $\mathcal{D}$ only after this responds has been sent. Clearly, we have that

$$\mathcal{E} \upharpoonright \underline{\mathcal{P}} \equiv \mathcal{E}' \upharpoonright \underline{\mathcal{D}} \upharpoonright \underline{\mathcal{P}}$$

for every $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\underline{\mathcal{P}})$. Now, it is easy to conclude that

$$\mathcal{E} \upharpoonright \mathcal{P}^{q(\mathbf{n})} \equiv \mathcal{E}' \upharpoonright !_{q(\mathbf{n})}\, \underline{\mathcal{D}} \upharpoonright \mathcal{P}^{q(\mathbf{n})}$$

for every $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P}^{q(\mathbf{n})})$. □

Typically, a protocol is first activated through some IO channel and then *sends* a message on a network channel before it *accepts* a message on some network channel. Let us call such protocols which first send a message on a network channel before accepting a message on a network channel *initially network sending.* For such protocols the above lemma also holds if in the definition of $\mathcal{P}^{q(\mathbf{n})}$ the simple SID version of $\mathcal{P}$ is used because the dummy will first only accept input from the protocol rather than from the environment. After having received the first message from the protocol, the dummy can then extract the SID from this message and from now on only accept messages prefixed with this SID. If the protocol was not initially network sending, the dummy would not know in case the first message comes from the environment whether this message will be accepted by the protocol since the dummy does not know the SID yet.

Lemma 34. *If $\mathcal{P} \in \mathbf{R}$ is a network predictable process and is initially network sending, then $\mathcal{P}^{q(\mathbf{n})}$ is network predictable even if $\underline{\mathcal{P}}$ is the simple SID version of $\mathcal{P}$.*

As an immediate consequence of Lemma 32, 33, and 34 we obtain the following theorem:

Theorem 35. *If $\mathcal{P}_1, \ldots, \mathcal{P}_k \in \mathbf{R}$ are network predictable processes and $q_1(\mathbf{n}), \ldots, q_k(\mathbf{n}) \geq 1$ are polynomials such that $\mathcal{P}_j^{q_j(\mathbf{n})}$ is environmentally valid for $\mathcal{P}_{j+1}^{q_{j+1}(\mathbf{n})} \upharpoonright \cdots \upharpoonright \mathcal{P}_k^{q_k(\mathbf{n})}$ for every j, then $\mathcal{P}_1^{q_1(\mathbf{n})} \upharpoonright \cdots \upharpoonright \mathcal{P}_k^{q_k(\mathbf{n})}$ is network predicatable. In case $\mathcal{P}_i$ is initially network sending, then $\underline{\mathcal{P}_i}$ can be chosen to be the simple SID version of $\mathcal{P}_i$.*

9 A General Composition Theorem

We prove a general composition theorem for composing a polynomial number of instances (also called copies) of protocols. This is done in two steps. We first, in Section 9.1, show that a constant number of (possibly) different protocols can securely be composed by parallel composition. We then, in Section 9.2, consider the (parallel) composition of a polynomial number of instances of one protocol. From this we immediately obtain the general composition theorem (Section 9.3) which states that a polynomial number of instances of protocols derived from a constant number of different protocols can securely be composed. In Section 9.3, we also show that the restriction to a constant number of different protocols is necessary, unless additional assumptions are made in the composition theorem.

9.1 Composing a Constant Number of Protocols

The following lemma shows how a constant number of possibly different protocols can securely be composed.

Lemma 36. *As usual, let* $\mathbf{R}$ *be the set of all closed regular process expressions and* $\mathbf{MD}$ *be the set of all (open/closed) master decision process expressions. Let* $\mathcal{P}_1, \ldots, \mathcal{P}_k, \mathcal{F}_1, \ldots, \mathcal{F}_k \in \mathbf{R}$ *such that for every* j *the following conditions are true:*

1. $\mathcal{P}_j$ *is environmentally valid for* $\mathcal{P}_{j+1} \upharpoonleft \cdots \upharpoonleft \mathcal{P}_k$.
2. $\mathcal{F}_j$ *is environmentally valid for* $\mathcal{F}_{j+1} \upharpoonleft \cdots \upharpoonleft \mathcal{F}_k$.
3. $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P}_j, \mathcal{F}_j)$.

Then,

$$\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P}_1 \upharpoonleft \cdots \upharpoonleft \mathcal{P}_k, \mathcal{F}_1 \upharpoonleft \cdots \upharpoonleft \mathcal{F}_k).$$

Proof. We prove the lemma for $k = 2$. For $k > 2$, the statement follows by induction on k.

Assume that $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P}_j, \mathcal{F}_j)$ for every $j \in \{1, 2\}$. Thus, there exists $\mathcal{S}_j \in \mathbf{R}\text{-Adv}_{\mathcal{P}_j}(\mathcal{F}_j)$ such that $\mathcal{E} \upharpoonleft \mathcal{P}_j \equiv \mathcal{E} \upharpoonleft \mathcal{S}_j \upharpoonleft \mathcal{F}_j$ for every $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P}_j)$.

Let $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P}_1 \upharpoonleft \mathcal{P}_2)$. In particular, $\mathcal{E}$ is valid for $\mathcal{S}_1 \upharpoonleft \mathcal{F}_1 \upharpoonleft \mathcal{S}_2 \upharpoonleft \mathcal{F}_2$. We have

$$\begin{array}{rcll}
\mathcal{E} \upharpoonleft \mathcal{P}_1 \upharpoonleft \mathcal{P}_2 & \equiv & \mathcal{E} \upharpoonleft \mathcal{P}_1 \upharpoonleft \mathcal{S}_2 \upharpoonleft \mathcal{F}_2 & (i) \\
& \equiv & \mathcal{E} \upharpoonleft \mathcal{S}_2 \upharpoonleft \mathcal{F}_2 \upharpoonleft \mathcal{P}_1 & (ii) \\
& \equiv & \mathcal{E} \upharpoonleft \mathcal{S}_2 \upharpoonleft \mathcal{F}_2 \upharpoonleft \mathcal{S}_1 \upharpoonleft \mathcal{F}_1 & (iii) \\
& \equiv & \mathcal{E} \upharpoonleft \mathcal{S}_1 \upharpoonleft \mathcal{S}_2 \upharpoonleft \mathcal{F}_1 \upharpoonleft \mathcal{F}_2 & (iv)
\end{array}$$

where in (i) we use that $\mathcal{E}' \upharpoonleft \mathcal{P}_2 \equiv \mathcal{E}' \upharpoonleft \mathcal{S}_2 \upharpoonleft \mathcal{F}_2$ for every $\mathcal{E}' \in \mathbf{MD}\text{-Valid}(\mathcal{P}_2)$, and that $\mathcal{E} \upharpoonleft \mathcal{P}_1 \in \mathbf{MD}\text{-Valid}(\mathcal{P}_2)$; in (ii) we use COM and ASC (see Section 5), and thus, $\mathcal{P}_1 \upharpoonleft \mathcal{S}_2 \upharpoonleft \mathcal{F}_2 \equiv \mathcal{S}_2 \upharpoonleft \mathcal{F}_2 \upharpoonleft \mathcal{P}_1$; in (iii) we use that a) $\mathcal{E} \upharpoonleft \mathcal{S}_2 \upharpoonleft \mathcal{F}_2 \in \mathbf{MD}\text{-Valid}(\mathcal{P}_1)$, and that b) $\mathcal{E}' \upharpoonleft \mathcal{P}_1 \equiv \mathcal{E}' \upharpoonleft \mathcal{S}_1 \upharpoonleft \mathcal{F}_1$ for every $\mathcal{E}' \in \mathbf{MD}\text{-Valid}(\mathcal{P}_1)$; in (iv) we again use COM and ASC.

Finally, since $\mathcal{P}_1$ and $\mathcal{F}_1$ are environmentally valid for $\mathcal{P}_2$ and $\mathcal{F}_2$, respectively, and $\mathcal{P}_j$ and $\mathcal{S}_j \upharpoonleft \mathcal{F}_j$ are compatible, it follows that $\mathcal{S}_1$ is environmentally valid for $\mathcal{S}_2$ and that $\mathcal{S}_1 \upharpoonleft \mathcal{S}_2 \in \mathbf{R}\text{-Adv}_{\mathcal{P}_1 \upharpoonleft \mathcal{P}_2}(\mathcal{F}_1 \upharpoonleft \mathcal{F}_2)$. Consequently, (iv) yields $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P}_1 \upharpoonleft \mathcal{P}_2, \mathcal{F}_1 \upharpoonleft \mathcal{F}_2)$. □

By Corollary 22, this lemma immediately carries over to other variants of strong simulatability and strong black-box simulatability. Given that the $\mathcal{P}_i$ are network predictable, by Corollary 28 and Lemma 32, the lemma also holds for universal composability/weak black-box simulatability.

Corollary 37. *Lemma 36 is also true if the version of strong simulatability used in the lemma is replaced by*

1. *Any of the security notions in the class* SS/SBB *(see Corollary 22),*
2. *Any of the security notions in the class* $\mathsf{UC/WBB}_{env}$ *(see Corollary 23) given that the* $\mathcal{P}_j$ *are network predictable.*

Clearly, one can proof the composition theorem also for versions of security notions not equivalent to strong simulatability, such as the security notions in $\mathsf{UC/WBB}_{sim}$, but the point here is that one does not have to do so for equivalent security notions.

9.2 Composing a Polynomial Number of Instances of one Protocol

We now show that we can securely compose a polynomial number of instances of one protocol. More precisely, we prove that if $\mathcal{P}$ and $\mathcal{F}$ are strong simulatable, then so are $\mathcal{P}^{q(\mathbf{n})}$ and $\mathcal{F}^{q(\mathbf{n})}$. Observe that if $\mathcal{P}$ and $\mathcal{F}$ are IO-compatible, then $\mathcal{P}^{q(\mathbf{n})}$ and $\mathcal{F}^{q(\mathbf{n})}$ are IO-compatible if the SID managers use the same channel to receive SIDs and if, say, in the SID version of $\mathcal{F}$ the channels `netsend` and `netrec` are renamed to the new channels `simsend` and `simrec`.

Lemma 38. *As usual, let* $\mathbf{R}$ *be the set of all closed regular context expressions and* $\mathbf{MD}$ *be the set of all (possibly open) master decision context expressions. Let* $q(\mathbf{n}) \geq 1$ *be a polynomial. If* $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$*, and* $\mathcal{P}^{q(\mathbf{n})}$ *and* $\mathcal{F}^{q(\mathbf{n})}$ *are IO-compatible, then*

$$\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P}^{q(\mathbf{n})}, \mathcal{F}^{q(\mathbf{n})}).$$

This is true independently of whether in $\mathcal{P}^{q(\mathbf{n})}$ *the simple or non-simple SID version of* $\mathcal{P}$ *is used.*

In the remainder of this subsection, we prove the lemma for the case that $\underline{\mathcal{P}}$ is the simple SID version of $\mathcal{P}$. Throughout the proof, we point out the changes necessary if $\underline{\mathcal{P}}$ is the non-simple SID version of $\mathcal{P}$. We stress that for $\mathcal{F}$ we always consider the (non-simple) SID version since the SID version of $\mathcal{F}$ needs to announce the SID to its simulator.

We use the following abbreviation. For every process expression $\mathcal{Q}(\vec{x})$, we set

$$f_{\mathcal{Q}(\vec{x})}(i, \vec{a}) := \mathtt{Prob}[\mathcal{Q}(\vec{a})^{\mathbf{n}\leftarrow i} \rightsquigarrow 1]$$

for every i and $\vec{a}$.

Since $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P},\mathcal{F})$, we know that there exists $\mathcal{S} \in \mathbf{R}\text{-Adv}_{\mathcal{P}}(\mathcal{F})$ such that $\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}$ for every $\mathcal{E} \in \mathbf{MD}\text{-Valid}(\mathcal{P})$. Define

$$\mathcal{S}' \quad = \quad \mathtt{in}(\mathtt{simsend}, x).(\mathtt{out}(\mathtt{simrec}, x) \mid\mid \mathcal{S}(x))$$

where $\mathcal{S}(x)$ is defined just as $\mathcal{P}(x)$ in Section 8.2. Informally speaking, $\mathcal{S}'$ receives the SID from $\underline{\mathcal{F}}$, acknowledges the receipt, and generates the copy $\mathcal{S}(x)$ of $\mathcal{S}$ which works exactly as $\mathcal{S}$ but requires incoming and outgoing messages to be prefixed by the SID x.[8]

It is easy to verify that $!_{q(\mathbf{n})}\, \mathcal{S}' \in \mathbf{R}\text{-Adv}_{\mathcal{P}^{q(\mathbf{n})}}(\mathcal{F}^{q(\mathbf{n})})$.

To prove the lemma, it remains to show that

$$\mathcal{E}(\vec{z}) \mid \mathcal{P}^{q(\mathbf{n})} \quad \equiv \quad \mathcal{E}(\vec{z}) \mid (!_{q(\mathbf{n})}\, \mathcal{S}') \mid \mathcal{F}^{q(\mathbf{n})}$$

for every $\mathcal{E}(\vec{z}) \in \mathbf{MD}\text{-Valid}(\mathcal{P}^{q(\mathbf{n})})$.

In what follows, let $\mathcal{E}(\vec{z}) \in \mathbf{MD}\text{-Valid}(\mathcal{P}^{q(\mathbf{n})})$ and let $p(\mathbf{n})$ be a polynomial. We need to show that there exists i_0 such that

$$\begin{aligned} f(i, \vec{a}) \quad &:= \quad |f_{\mathcal{E}(\vec{z}) \mid \mathcal{P}^{q(\mathbf{n})}}(i, \vec{a}) - f_{\mathcal{E}(\vec{z}) \mid (!_{q(\mathbf{n})}\, \mathcal{S}') \mid \mathcal{F}^{q(\mathbf{n})}}(i, \vec{a})| \\ &\leq \quad \frac{1}{p(i)} \end{aligned}$$

[8] In case $\underline{\mathcal{P}}$ is the (non-simple) SID version of $\mathcal{P}$, after receiving x on `simsend`, $\mathcal{S}'$ would have to send x on `netsend` and wait for responds on `netrec` before sending an acknowledgment on `simrec`.

for every $i \geq i_0$ and every tuple $\overrightarrow{a}$ of bit strings. The proof continues by a hybrid argument.

From now on, we assume that in the SID version of $\underline{\mathcal{F}}$ the channel $\underline{\mathtt{sid}}$ is replaced by $\underline{\mathtt{sid}}_F$ and that in $\underline{\mathcal{P}}$ the channel $\underline{\mathtt{sid}}$ is replaced by the channel $\underline{\mathtt{sid}}_P$ different from $\underline{\mathtt{sid}}_F$.

We now define a hybrid variant $\mathcal{M}^{q(\mathbf{n})}(z)$ with free variable z of the SID manager which in the first z invocations generates copies of $\mathcal{F}$ and in the remaining invocations generates copies of $\mathcal{P}$. The manager will interpret z as a unary encoding of a non-negative integer. Formally,

$$\begin{array}{rcl}
\mathcal{M}^{q(\mathbf{n})}(z) & = & \mathtt{in}(\mathtt{sid}, x).(\mathtt{out}(\mathtt{zero}, \varepsilon) \mid\mid \\
& & \big(\mathtt{in}(\mathtt{zero}, t_{z=0}).(\mathtt{out}(\underline{\mathtt{sid}}_P, x) \mid\mid \mathtt{in}(\mathtt{received}, x).(\mathtt{out}(\mathtt{state}, x) \mid\mid \mathtt{out}(\mathtt{done}, \mathtt{suc}))) + \\
& & \quad \mathtt{in}(\mathtt{zero}, t_{z\neq 0}).(\mathtt{out}(\underline{\mathtt{sid}}_F, x) \mid\mid \mathtt{in}(\mathtt{received}, x).(\mathtt{out}(\mathtt{state}, x) \mid\mid \mathtt{out}(\mathtt{done}, \mathtt{suc})))\big) \mid\mid \\
& & !_{q(\mathbf{n})-1}\ \mathtt{in}(\mathtt{state}, y).(\\
& & \quad \mathtt{in}(\mathtt{sid}, t^{<z}_{x\notin y}).(\\
& & \quad\quad \mathtt{out}(\underline{\mathtt{sid}}_F, x) \mid\mid \mathtt{in}(\mathtt{received}, x).(\mathtt{out}(\mathtt{state}, (y, x)) \mid\mid \mathtt{out}(\mathtt{done}, \mathtt{suc}))) + \\
& & \quad \mathtt{in}(\mathtt{sid}, t^{\geq z}_{x\notin y}).(\\
& & \quad\quad \mathtt{out}(\underline{\mathtt{sid}}_P, x) \mid\mid \mathtt{in}(\mathtt{received}, x).(\mathtt{out}(\mathtt{state}, (y, x)) \mid\mid \mathtt{out}(\mathtt{done}, \mathtt{suc}))) + \\
& & \quad \mathtt{in}(\mathtt{sid}, t_{x\in y}).(\\
& & \quad\quad \mathtt{out}(\mathtt{state}, y) \mid\mid \mathtt{out}(\mathtt{done}, \mathtt{fail})))
\end{array}$$

where $t_{z=0}$ is an M-term which accepts if the input string z represents 0, and otherwise rejects. The M-term $t_{z\neq 0}$ is defined dually. Note that since $t_{z=0}$ and $t_{z\neq 0}$ are Turing Machines which run in polynomial time in the size of the security parameter, these machines may not be able to read the whole string z. However, they only have to examine the first bit of z to return their answer. The M-term $t^{<z}_{x\notin y}$ is defined just as $t_{x\notin y}$ except that $t^{<z}_{x\notin y}$ only accepts a string a if the length of the sequence y (recall that y represents the state of the manager) is $< z$, i.e., so far $< z$ copies of $\mathcal{F}$ have been generated excluding the copy that is about to be generated. Analogously for $t^{\geq z}_{x\notin y}$. Again, $t^{<z}_{x\notin y}$ and $t^{\geq z}_{x\notin y}$ may not be able to read the whole bit string z. However, the size of y is bounded by a polynomial in the security parameter, and thus, the M-term can check their length. To compare this length with z, it is only necessary to read a polynomial number of bits of z.

Let

$$\mathcal{S}(\overrightarrow{z}, z) = \mathcal{E}(\overrightarrow{z}) \upharpoonleft \mathcal{M}^{q(\mathbf{n})}(z) \upharpoonleft !_{q(\mathbf{n})}\ \mathcal{S}' \upharpoonleft !_{q(\mathbf{n})}\ \underline{\mathcal{F}} \upharpoonleft !_{q(\mathbf{n})}\ \underline{\mathcal{P}}$$

Clearly, $\mathcal{E}(\overrightarrow{z})$ is valid for $\mathcal{M}^{q(\mathbf{n})}(z) \upharpoonleft !_{q(\mathbf{n})}\ \mathcal{S}' \upharpoonleft !_{q(\mathbf{n})}\ \underline{\mathcal{F}} \upharpoonleft !_{q(\mathbf{n})}\ \underline{\mathcal{P}}$.

Obviously, we have that

$$\begin{array}{rcl}
f_{\mathcal{E}(\overrightarrow{z}) \upharpoonleft \mathcal{P}^{q(\mathbf{n})}}(i, \overrightarrow{a}) & = & f_{\mathcal{S}(\overrightarrow{z}, z)}(i, \overrightarrow{a}, 0) \text{ and} \\
f_{\mathcal{E}(\overrightarrow{z}) \upharpoonleft \mathcal{F}^{q(\mathbf{n})}}(i, \overrightarrow{a}) & = & f_{\mathcal{S}(\overrightarrow{z}, z)}(i, \overrightarrow{a}, q(i))
\end{array}$$

for every i and tuple $\overrightarrow{a}$ of bit strings. Define

$$f_j(i, \overrightarrow{a}) = |f_{\mathcal{S}(\overrightarrow{z}, z)}(i, \overrightarrow{a}, j) - f_{\mathcal{S}(\overrightarrow{z}, z)}(i, \overrightarrow{a}, j{+}1)|.$$

By the triangle inequality, we have

$$f(i, \overrightarrow{a}) = |f_{\mathcal{S}}(i, \overrightarrow{a}, 0) - |f_{\mathcal{S}}(i, \overrightarrow{a}, q(i))| \leq \sum_{j=0}^{q(i)-1} f_j(i, \overrightarrow{a}).$$

We show that for every polynomial $p'(i)$ there exists i_0 such that $f_j(i, \overrightarrow{a}) \leq \frac{1}{p'(i)}$ for every $i \geq i_0$, tuple $\overrightarrow{a}$, and $j \geq 0$. Now, if we set $p'(i) = p(i) \cdot q(i)$ it follows that $f(i, \overrightarrow{a}) \leq \frac{1}{p(i) \cdot q(i)} \cdot q(i) = \frac{1}{p(i)}$ for every $i \geq i_0$ and every tuple $\overrightarrow{a}$, which would conclude the proof of the lemma.

To show the above, let us consider the processes $\mathcal{S}(\overrightarrow{z}, j)$ and $\mathcal{S}(\overrightarrow{z}, j{+}1)$ more closely. These processes almost coincide. The only difference is that after the $j{+}1$st invocation of the hybrid manager in $\mathcal{S}(\overrightarrow{z}, j)$ a copy of $\mathcal{P}$ is generated while in $\mathcal{S}(\overrightarrow{z}, j{+}1)$ a copy of $\mathcal{F}$ is generated (this copy of $\mathcal{F}$, then interacts with some copy of the simulator $\mathcal{S}$). Note that if $j{+}1 > q(i)$, i.e., $j{+}1$ is greater than the total number of copies that are going to be generated for the security parameter i, then $\mathcal{S}(\overrightarrow{z}, j)$ and $\mathcal{S}(\overrightarrow{z}, j{+}1)$ coincide completely. If we now "extract" these copies (of $\mathcal{P}$ and $\mathcal{F}$) from $\mathcal{S}(\overrightarrow{z}, j)$ and $\mathcal{S}(\overrightarrow{z}, j{+}1)$, then the processes that remain coincide, let us call this process $\mathcal{E}'$, and can be considered an environment trying to distinguish the copy of $\mathcal{P}$ from the copy of $\mathcal{F}$ and $\mathcal{S}$. However, we first need to consistently rename the channels of the copies of $\mathcal{P}$, $\mathcal{F}$, and $\mathcal{S}$ such that $\mathcal{E}'$ is in fact a valid context for these copies.

Formally, let $\widehat{\mathcal{P}}$ be obtained from $\mathcal{P}$ by renaming all external channels by new channels. Let $\widehat{\mathcal{F}}$ be defined analogously where the set of IO channels of $\widehat{\mathcal{P}}$ and $\widehat{\mathcal{F}}$ coincide and the sets of network channels of $\widehat{\mathcal{P}}$ and $\widehat{\mathcal{F}}$ are disjoint. Finally, let $\widehat{\mathcal{S}}$ be obtained from $\mathcal{S}$ as follows: First note that $\mathcal{S}$ does not have IO channels. The network channels of $\mathcal{S}$ are renamed consistently with the renaming of network channels of $\mathcal{F}$ and $\mathcal{P}$, i.e., the network channels with which $\mathcal{S}$ connects to $\mathcal{F}$ are renamed according to the renaming of network channels of $\mathcal{F}$. The other network channels coincide with those of $\mathcal{P}$ and they are renamed according to the renaming of network channels of $\mathcal{P}$. Since $\mathcal{S}$ is adversarially valid for $\mathcal{F}$, by construction we have that $\widehat{\mathcal{S}}$ is adversarially valid for $\widehat{\mathcal{F}}$. Moreover, just as $\mathcal{P}$ and $\mathcal{S} \upharpoonright \mathcal{F}$, the process expressions $\widehat{\mathcal{P}}$ and $\widehat{\mathcal{S}} \upharpoonright \widehat{\mathcal{F}}$ are compatible.

We now modify $\mathcal{M}^{q(\mathbf{n})}(j)$ such that after the $j{+}1$st invocation, no copy of $\mathcal{P}$ (or $\mathcal{F}$) is generated. Instead $\mathcal{M}^{q(\mathbf{n})}(j)$ will simply return `suc` on the channel `done` to indicate that a copy has been generated provided that the SID sent in the $j{+}1$st invocation is in fact new. The idea is that the $j{+}1$st copy is $\widehat{\mathcal{P}}$ (or $\widehat{\mathcal{F}}$ which will run in parallel with $\widehat{\mathcal{S}}$). The new version of the hybrid manager is denoted by $\widehat{\mathcal{M}}^{q(\mathbf{n})}(z)$ (where z is the variable which will be substituted for j) and is defined as in Figure 4 where the M-term $t_{z=0}(x)$ accepts a if $z = 0$ and then substitutes a for x. Dually for $t_{z\neq 0}(x)$. The other M-terms are defined analogously to the ones in $\mathcal{M}^{q(\mathbf{n})}(z)$. As also pointed for $\mathcal{M}^{q(\mathbf{n})}(z)$, the M-terms can perform there tasks even if they cannot read all of z.

If $\mathcal{E}(\overrightarrow{z})$ in the $j{+}1$st invocation of $\widehat{\mathcal{M}}^{q(\mathbf{n})}(j)$ sent the SID a and this SID was new, then $\widehat{\mathcal{M}}^{q(\mathbf{n})}(j)$ will still not generate a copy of $\mathcal{P}$ (or $\mathcal{F}$) since, as mentioned before, the $j{+}1$st copy will be represented by $\widehat{\mathcal{P}}$ or $\widehat{\mathcal{F}}$. Thus, we need to modify $\mathcal{E}(\overrightarrow{z})$ such that instead of accessing $\mathcal{P}$ or $\mathcal{F}$ using the SID a it uses the new channels of $\widehat{\mathcal{P}}$ and $\widehat{\mathcal{F}} \upharpoonright \widehat{\mathcal{S}}$. To adjust $\mathcal{E}(\overrightarrow{z})$ in this way, it is helpful to think of $\mathcal{E}(\overrightarrow{z})$ to be in single machine normal form. By Lemma 3, this is w.l.o.g. We define the new variant $\widehat{\mathcal{E}}(\overrightarrow{z}, z)$ of $\mathcal{E}(\overrightarrow{z})$ as follows: The new free variable z will take the value j. To the set of external channels of $\mathcal{E}(\overrightarrow{z})$, the set of external channels of $\widehat{\mathcal{P}}$, and thus, $\widehat{\mathcal{F}} \upharpoonright \widehat{\mathcal{S}}$, are added where the input channels of $\widehat{\mathcal{P}}$ are added as output channels of $\widehat{\mathcal{E}}(\overrightarrow{z}, z)$ and the output channels of $\widehat{\mathcal{P}}$ are added as input channels of $\widehat{\mathcal{E}}(\overrightarrow{z}, z)$ to the set of channels of $\widehat{\mathcal{E}}(\overrightarrow{z}, z)$. Now, intuitively, $\widehat{\mathcal{E}}(\overrightarrow{z}, z)$ simulates $\mathcal{E}(\overrightarrow{z})$ with the following modifications. First, $\widehat{\mathcal{E}}(\overrightarrow{z}, z)$ records in its internal state the number of times a SID has been sent to the manager, i.e., a message a has been sent on channel `sid` (to the hybrid manager). If z is substituted by j, in the $j{+}1$st invocation, $\widehat{\mathcal{E}}(\overrightarrow{z}, z)$ records the SID a sent to the hybrid manager and also records whether `suc` or `fail` is returned by the hybrid manager

$$
\begin{aligned}
\widehat{\mathcal{M}}^{q(\mathbf{n})}(z) \quad = \quad & \mathtt{in}(\mathtt{sid}, t_{z=0}(x)).(\mathtt{out}(\mathtt{state}, x) \mid\mid \mathtt{out}(\mathtt{done}, \mathtt{suc})) + \\
& \mathtt{in}(\mathtt{sid}, t_{z\neq 0}(x)).(\mathtt{out}(\underline{\mathtt{sid}}_F, x) \mid\mid \mathtt{in}(\mathtt{received}, x)(\mathtt{out}(\mathtt{state}, x) \mid\mid \mathtt{out}(\mathtt{done}, \mathtt{suc}))) \mid\mid \\
& !_{q(\mathbf{n})-1}\ \mathtt{in}(\mathtt{state}, y).(\\
& \quad \mathtt{in}(\mathtt{sid}, t^{<z}_{x\notin y}).(\\
& \quad\quad \mathtt{out}(\underline{\mathtt{sid}}_F, x) \mid\mid \mathtt{in}(\mathtt{received}, x).(\mathtt{out}(\mathtt{state}, (y, x)) \mid\mid \mathtt{out}(\mathtt{done}, \mathtt{suc}))) + \\
& \quad \mathtt{in}(\mathtt{sid}, t^{=z}_{x\notin y}).(\\
& \quad\quad (\mathtt{out}(\mathtt{state}, (y, x)) \mid\mid \mathtt{out}(\mathtt{done}, \mathtt{suc}))) + \\
& \quad \mathtt{in}(\mathtt{sid}, t^{>z}_{x\notin y}).(\\
& \quad\quad \mathtt{out}(\underline{\mathtt{sid}}_P, x) \mid\mid \mathtt{in}(\mathtt{received}, x).(\mathtt{out}(\mathtt{state}, (y, x)) \mid\mid \mathtt{out}(\mathtt{done}, \mathtt{suc}))) + \\
& \quad \mathtt{in}(\mathtt{sid}, t_{x\in y}).(\\
& \quad\quad \mathtt{out}(\mathtt{state}, y) \mid\mid \mathtt{out}(\mathtt{done}, \mathtt{fail})))
\end{aligned}
$$

Figure 4: The hybrid manager.

on channel `done`. Now, every time $\mathcal{E}(\vec{z})$ would send a message of the form (a, b) on an external channel of $\mathcal{P}$ (and thus, $\mathcal{F} \upharpoonright \mathcal{S}$), the process $\widehat{\mathcal{E}}(\vec{z}, z)$ sends the message b on the corresponding channel of $\widehat{\mathcal{P}}$ ($\widehat{\mathcal{F}} \upharpoonright \widehat{\mathcal{S}}$) provided that the j+1st invocation of the hybrid manager was successful, otherwise nothing is sent. If $\widehat{\mathcal{E}}(\vec{z}, z)$ receives a message b on some external channel of $\widehat{\mathcal{P}}$ ($\widehat{\mathcal{F}} \upharpoonright \widehat{\mathcal{S}}$), then $\widehat{\mathcal{E}}(\vec{z}, z)$ simulates $\mathcal{E}(\vec{z})$ pretending to have received the message (a, b) on the corresponding channel of $\mathcal{P}$ ($\mathcal{F} \upharpoonright \mathcal{S}$). It is straightforward to define the internal state of $\widehat{\mathcal{E}}(\vec{z}, z)$ and the C- and M-terms of $\widehat{\mathcal{E}}(\vec{z}, z)$ to obtain the behavior just described.[9] We note that for this it is important that $\widehat{\mathcal{E}}$ has direct access to z. If we restricted environments, and in particular $\mathcal{E}$ and $\widehat{\mathcal{E}}$ to have only one free variable, then we would need to assume that $\widehat{\mathcal{E}}$ interprets the bit string given to it as a tuple of the form (a, j) where a is the input for $\mathcal{E}$ and j is the length. However, for small security parameters, $\widehat{\mathcal{E}}$ might not be able to read even part of j, and thus, would not be able to perform the simulation described above. (A simular argument applies if $\widehat{\mathcal{E}}$ interprets the input string as (j, a).)

Define

$$\mathcal{E}'(\vec{z}, z) = \widehat{\mathcal{E}}(\vec{z}, z) \upharpoonright \widehat{\mathcal{M}}^{q(\mathbf{n})} z \upharpoonright !_{q(\mathbf{n})}\ \mathcal{S}' \upharpoonright !_{q(\mathbf{n})}\ \underline{\mathcal{F}} \upharpoonright !_{q(\mathbf{n})}\ \underline{\mathcal{P}}.$$

Now, it is easy to verify that

$$
\begin{aligned}
f_{\mathcal{S}(\vec{z},z)}(i, \vec{a}, j) \quad &= \quad f_{\mathcal{E}'(\vec{z},z) \upharpoonright \widehat{\mathcal{P}}}(i, \vec{a}, j) \qquad \text{and} \qquad (5) \\
f_{\mathcal{S}(\vec{z},z)}(i, \vec{a}, j{+}1) \quad &= \quad f_{\mathcal{E}'(\vec{z},z) \upharpoonright \widehat{\mathcal{S}} \upharpoonright \widehat{\mathcal{F}}}(i, \vec{a}, j) \qquad (6)
\end{aligned}
$$

[9]If $\underline{\mathcal{P}}$ is the (non-simple) SID version of $\mathcal{P}$, then in $\widehat{\mathcal{E}}(\vec{z}, z)$ after sending the j+1st invocation the interaction with $\underline{\mathcal{P}}$ ($\underline{\mathcal{F}}$ together with the simulator) needs to be simulated as well: As above, first $\widehat{\mathcal{E}}(\vec{z}, z)$ would read the result on `done` to check whether the SID existed already or not. Now, if the SID is new, then $\mathcal{E}(\vec{z})$ is simulated pretending that the SID x is sent on `netsend`. If $\mathcal{E}(\vec{z})$ returns x on `netrec`, then $\mathcal{E}(\vec{z})$ is simulated pretending that on `done suc` or `fail` is sent depending on what was received on this channel before. From here on, the simulation continues as described above. If $\mathcal{E}(\vec{z})$ never sends x on `netrec`, then access to channels of $\mathcal{P}$ with SID x will be ignored because such a copy of $\mathcal{P}$ would not have been generated. Also, further access to the SID manager is ignored as this manager would wait for response from $\underline{\mathcal{P}}$ ($\underline{\mathcal{F}}$).

for every i, j, and tuple $\vec{a}$.

By construction, $\mathcal{E}'(\vec{z}, z) \in \mathbf{MD}\text{-Valid}(\widehat{\mathcal{P}})$, and thus, $\mathcal{E}'(\vec{z}, z)$ is valid for $\widehat{\mathcal{S}} \mid \widehat{\mathcal{F}}$.

We know that $\mathcal{E}'' \mid \mathcal{P} \equiv \mathcal{E}'' \mid \mathcal{S} \mid \mathcal{F}$ for every $\mathcal{E}'' \in \mathbf{MD}\text{-Valid}(\mathcal{P})$, and thus, $\mathcal{E}'' \mid \widehat{\mathcal{P}} \equiv \mathcal{E}'' \mid \widehat{\mathcal{S}} \mid \widehat{\mathcal{F}}$ for every $\mathcal{E}'' \in \mathbf{MD}\text{-Valid}(\widehat{\mathcal{P}})$. In particular, for the polynomial $p'(i)$ there exists i_0 such that

$$\begin{aligned} f_j(i, \vec{a}) &= |f_{\mathcal{E}'(\vec{z},z) \mid \widehat{\mathcal{P}}}(i, \vec{a}, j) - f_{\mathcal{E}'(\vec{z},z) \mid \widehat{\mathcal{S}} \mid \widehat{\mathcal{F}}}(i, \vec{a}, j)| \\ &\leq \frac{1}{p'(i)} \end{aligned}$$

for every i, tuple $\vec{a}$, and j, where the first equality is by (5), (6), and the definition of $f_j(i, \vec{a})$. This concludes the proof of Lemma 38

Just as in Section 9.1, by Corollary 22, this lemma immediately carries over to other variants of strong simulatability and strong black-box simulatability. Given that the $\mathcal{P}_i$ are network predictable, by Corollary 28 and Lemma 33, the lemma also holds for universal composability/weak black-box simulatability.

Corollary 39. *Lemma 38 is also true if the version of strong simulatability used in the lemma is replaced by*

1. *Any of the security notions in the class SS/SBB (see Corollary 22),*
2. *Any of the security notions in the class UC/WBB$_{env}$ (see Corollary 23) given that the $\mathcal{P}_j$ are network predictable and the non-simple SID version of $\mathcal{P}_j$ is used. In case $\mathcal{P}_j$ is initially network sending, the simple SID version of $\mathcal{P}_j$ may be used as well.*

As already mentioned in Section 9.1, even though one can proof the composition theorem also for versions of security notions not equivalent to strong simulatability, such as the security notions in UC/WBB$_{sim}$, the point here is that one does not have to do so for equivalent security notions.

9.3 The Composition Theorem

Putting Lemma 36 and 38 together, we immediately obtain the following composition theorem.

Theorem 40. *Let $\mathbf{R}$ be the set of all closed regular context expressions and $\mathbf{MD}$ be the set of all (possibly open) master decision context expressions. Let $\mathcal{P}_1, \ldots, \mathcal{P}_k, \mathcal{F}_1, \ldots, \mathcal{F}_k \in \mathbf{R}$ and $q_1(\mathbf{n}), \ldots, q_k(\mathbf{n}) \geq 1$ be polynomials such that for every j the following conditions are true:*

1. *$\mathcal{P}_j^{q_j(\mathbf{n})}$ is environmentally valid for $\mathcal{P}_{j+1}^{q_{j+1}(\mathbf{n})} \mid \cdots \mid \mathcal{P}_k^{q_k(\mathbf{n})}$.*
2. *$\mathcal{F}_j^{q_j(\mathbf{n})}$ is environmentally valid for $\mathcal{F}_{j+1}^{q_{j+1}(\mathbf{n})} \mid \cdots \mid \mathcal{F}_k^{q_k(\mathbf{n})}$.*
3. *$\mathcal{P}_j^{q_j(\mathbf{n})}$ and $\mathcal{F}_j^{q_j(\mathbf{n})}$ are IO-compatible.*
4. *$\mathsf{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P}_j, \mathcal{F}_j)$.*

Then,

$$\mathsf{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P}_1^{q_1(\mathbf{n})} \mid \cdots \mid \mathcal{P}_k^{q_k(\mathbf{n})}, \mathcal{F}_1^{q_1(\mathbf{n})} \mid \cdots \mid \mathcal{F}_k^{q_k(\mathbf{n})}).$$

This is true independently of whether in $\mathcal{P}^{q(\mathbf{n})}$ the simple or non-simple SID version of $\mathcal{P}$ is used.

Just as for Lemma 36 and 38, these results immediately carry over to other security notions.

Corollary 41. *Theorem 40 is also true if the version of strong simulatability used in the theorem is replaced by*

1. *Any of the security notions in the class* SS/SBB *(see Corollary 22),*
2. *Any of the security notions in the class* $\mathsf{UC/WBB}_{env}$ *(see Corollary 23) given that the* $\mathcal{P}_j$ *are network predictable and the non-simple SID version of* $\mathcal{P}_j$ *is used. In case* $\mathcal{P}_j$ *is initially network sending, the simple SID version of* $\mathcal{P}_j$ *may be used as well.*

Remark 42. *Theorem 40 does not hold if the number of different processes* $\mathcal{P}_j$ *and* $\mathcal{F}_j$ *is not a constant* k *but a polynomial in the security parameter.*

Strictly speaking, this situation cannot be described in SPPC since in SPPC we can only talk about a polynomial number of processes of a constant number of different "types". To illustrate Remark 42, we extend SPPC in an ad hoc way.

Consider the infinite sequences $\mathcal{P}_1, \mathcal{P}_2, \ldots$ and $\mathcal{F}_1, \mathcal{F}_2, \ldots$ of regular process expressions with only IO channels. Let the IO-channels be the input channel c and the output channel c'. The process expressions use the same channel names but the jth process expression assumes a message to start with j and outputs messages that start with j.

We define

$$\mathcal{P}_j = \mathtt{in}(c_j, (j, x)).\mathtt{out}(c'_j, (j, 0))$$

and

$$\mathcal{F}_j = \mathtt{in}(c_j, (j, x)).\mathtt{out}(c'_j, (j, T_j))$$

where the C-term T_j returns 0 if the security parameter i satisfies $i > j$, and 1 otherwise.

Let $!_{j=1}^{q(\mathbf{n})}\mathcal{P}_j$ denote the process expression where $(!_{j=1}^{q(\mathbf{n})}\mathcal{P}_j)^{\mathbf{n}\leftarrow i} = \mathcal{P}_1^{\mathbf{n}\leftarrow i} \mid\mid \cdots \mid\mid \mathcal{P}_{q(i)}^{\mathbf{n}\leftarrow i}$. Note that strictly speaking $!_{j=1}^{q(\mathbf{n})}P_j$ is not a process expression in SPPC.

Obviously, $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(\mathcal{P}_j, \mathcal{F}_j)$ for every j since if for the security parameter i we have $i > j$, then $\mathcal{P}_j$ and $\mathcal{F}_j$ behave exactly in the same way.

However, $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(!_{j=1}^{\mathbf{n}}\mathcal{P}_j, !_{j=1}^{\mathbf{n}}\mathcal{F}_j)$ is *not* true: Since $\mathcal{P}_j$ and $\mathcal{F}_j$ do not have network channels, we do not need to consider a simulator for $!_{j=1}^{\mathbf{n}}\mathcal{F}_j$ since such a simulator would not have external channels, and thus, would never be active. Thus, $\mathtt{SS}_{(\mathbf{R},\mathbf{MD})}(!_{j=1}^{\mathbf{n}}\mathcal{P}_j, !_{j=1}^{\mathbf{n}}\mathcal{F}_j)$ implies that $\mathcal{E} \upharpoonright !_{j=1}^{\mathbf{n}}\mathcal{P}_j \equiv \mathcal{E} \upharpoonright !_{j=1}^{\mathbf{n}}\mathcal{F}_j$ for every $\mathcal{E} \in \mathbf{MD}\text{-Valid}(!_{j=1}^{\mathbf{n}}\mathcal{P}_j)$. However, an environment $\mathcal{E}$ can distinguish between $!_{j=1}^{\mathbf{n}}\mathcal{P}_j$ and $!_{j=1}^{\mathbf{n}}\mathcal{F}_j$ by sending a message to the ith process. If the answer is 0, then $\mathcal{E}$ interacts with $!_{j=1}^{\mathbf{n}}\mathcal{P}_j$ and otherwise with $!_{j=1}^{\mathbf{n}}\mathcal{F}_j$. Formally,

$$\mathcal{E} = \mathtt{in}(\mathtt{start}, x).(\mathtt{out}(c, T) \mid\mid \mathtt{in}(c', t_{=0}).\mathtt{out}(\mathtt{decision}, 1) + \mathtt{in}(c', t_{\neq 0}).\mathtt{out}(\mathtt{decision}, 0)$$

where the C-term T returns (i, ε) if invoked with the security parameter i. The M-term $t_{=0}$ accepts the input iff it is of the form $(i, 0)$ where i is the security parameter. Analogously for the M-term $t_{\neq 0}$. Obviously, $\mathcal{E} \upharpoonright !_{j=1}^{\mathbf{n}}\mathcal{P}_j \not\equiv \mathcal{E} \upharpoonright !_{j=1}^{\mathbf{n}}\mathcal{F}_j$.

Remark 43. *Backes et al. [4] prove a composition theorem for a non-uniform set of systems which in particular includes processes of the form* $!_{j=1}^{q(\mathbf{n})}\mathcal{P}_j$*. As the above example shows, in their composition theorem one therefore has to make additional assumptions not present in Theorem 40 or Canetti's composition theorem [9]. Also, their composition theorem is proved for black-box simulatability only. Since in their model, black-box simulatability does not imply universal composability, the theorem does not immediately carry over to universal composability.*

10 Conclusion

We have carried out a thorough study of the relationships among various notions of simulation-based security, identifying two properties of the computational model that determine equivalence between these notions. Our main results are that all variants of SS (strong simulatability) and SBB (strong black box simulatability) are equivalent, regardless of the selection of the master process, and they imply UC (universal composability) and WBB (weak black box simulatability). Conditions UC and WBB are equivalent as long as the role (master process or not) of the environment is the same in both. However, the variant of UC in which the environment may be a master process (as in [4, 9]) is strictly stronger than the variants in which the environment must not assume this role (as in [27]). In addition, the weaker forms of WBB do not imply SS/SBB. Finally, we prove a necessary and sufficient condition for UC/WBB to be equivalent to SS/SBB, based on the ability to define forwarders. These results all show that the relationship between universal composability and black-box simulatability is more subtle than previously described. In particular, the composability theorem of Canetti [9] does not necessarily imply that blackbox simulatability is a composable security notion over any computational model in which the forwarding property is not satisfied. Another technical observation is that making the environment the master process typically yields a stronger security notion. Hence, we recommend that in subsequent developments of the various models, the environment is always assigned the role of the master process.

Since our proofs are carried out axiomatically using the equational reasoning system developed for SPPC, we are able to apply the same arguments to suitably modified versions of the alternative computational models. We emphasize that our suggested modifications to the other systems are motivated by the failure, in those systems, of simple equational principles. In particular, it seems reasonable to adopt a buffer-free variant of PIOA.

While our study concentrates on models where the runtime of processes is bounded by a polynomial in the security parameter, our results involving the issue of placements of the master process should also carry over to models where the runtime of processes may depend on the number of invocations and the length of inputs [8, 18, 21].

Acknowledgments: We thank Michael Backes, Ran Canetti, Birgit Pfitzmann, Andre Scedrov, and Vitaly Shmatikov for helpful discussions.

References

[1] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *28th ACM Symposium on Principles of Programming Languages*, pages 104–115, 2001.

[2] Martín Abadi and Andrew D. Gordon. A bisimulation method for cryptographic protocol. In *Proc. ESOP 98*, Lecture notes in Computer Science. Springer, 1998.

[3] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 143:1–70, 1999. Expanded version available as SRC Research Report 149 (January 1998).

[4] M. Backes, B. Pfitzmann, and M. Waidner. A General Composition Theorem for Secure Reactive Systems. In *Proceedings of the 1st Theory of Cryptography Conference (TCC 2004)*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2004.

[5] M. Backes, B. Pfitzmann, and M. Waidner. Secure asynchronous reactive systems. Technical Report 082, Eprint, 2004.

[6] Michael Backes, Birgit Pfitzmann, Michael Steiner, and Michael Waidner. Polynomial fairness and liveness. In *Proceedings of 15th IEEE Computer Security Foundations Workshop*, pages 160–174, Cape Breton, Nova Scotia, Canada, 2002.

[7] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Reactively secure signature schemes. In *Proceedings of 6th Information Security Conference*, volume 2851 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2003.

[8] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Technical report, Cryptology ePrint Archive, December 2005. Online available at `http://eprint.iacr.org/2000/067.ps`.

[9] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symp. on the Foundations of Computer Science*. IEEE, 2001.

[10] Ran Canetti. Personal communication, 2004.

[11] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Proc. CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40, Santa Barbara, California, 2001. Springer.

[12] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In *Advances in Cryptology—EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002.

[13] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *Advances in Cryptology—EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 68–86. Springer, 2003.

[14] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proc. ACM Symp. on the Theory of Computing*, pages 494–503, 2002.

[15] A. Datta, R. Küsters, J.C. Mitchell, and A. Ramanathan. On the Relationships Between Notions of Simulation-Based Security. In J. Kilian, editor, *Proceedings of the 2nd Theory of Cryptography Conference (TCC 2005)*, volume 3378 of *Lecture Notes in Computer Science*, pages 476–494. Springer-Verlag, 2005.

[16] Anupam Datta, Ralf Küsters, John C. Mitchell, Ajith Ramanathan, and Vitaly Shmatikov. Unifying equivalence-based definitions of protocol security. In *ACM SIGPLAN and IFIP WG 1.7, 4th Workshop on Issues in the Theory of Security*, 2004. No formal proceedings.

[17] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[18] D. Hofheinz, J. Müller-Quade, and D. Unruh. Polynomial Runtime in Simulatability Definitions. In *18th IEEE Computer Security Foundations Workshop (CSFW-18 2005)*, pages 156–169. IEEE Computer Society, 2005.

[19] D. Hofheinz and D. Unruh. Comparing two notions of simulatability. In J. Kilian, editor, *Theory of Cryptography, Proceedings of TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 86–103. Springer-Verlag, 2005.

[20] D. Hofheinz and D. Unruh. Simulatable Security and Concurrent Composition. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, 2006. To appear.

[21] R. Küsters. Simulation-based security with inexhaustible interactive turing machines. Submitted., 2006.

[22] Patrick D. Lincoln, John C. Mitchell, Mark Mitchell, and Andre Scedrov. Probabilistic polynomial-time equivalence and security protocols. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *Formal Methods World Congress, vol. I*, number 1708 in Lecture Notes in Computer Science, pages 776–793, Toulouse, France, 1999. Springer.

[23] Robin Milner. *A Calculus of Communicating Systems*. Springer, 1980.

[24] Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[25] John C. Mitchell, Mark Mitchell, and Andre Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th Annual IEEE Symposium on the Foundations of Computer Science*, pages 725–733, Palo Alto, California, 1998. IEEE.

[26] John C. Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. A probabilistic polynomial-time calculus for the analysis of cryptographic protocols (preliminary report). In Stephen Brookes and Michael Mislove, editors, *17th Annual Conference on the Mathematical Foundations of Programming Semantics, Arhus, Denmark, May, 2001*, volume 45. Electronic notes in Theoretical Computer Science, 2001.

[27] B. Pfitzmann and M. Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In *IEEE Symposium on Security and Privacy*, pages 184–200. IEEE Computer Society Press, 2001.

[28] Ajith Ramanathan, John C. Mitchell, Andre Scedrov, and Vanessa Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. Unpublished, see `http://www-cs-students.stanford.edu/~ajith/`, 2004.

[29] Ajith Ramanathan, John C. Mitchell, Andre Scedrov, and Vanessa Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In *FOSSACS 2004 - Foundations of Software Science and Computation Structures*, March 2004. Summarizes results in [28]; to appear.

A Proof of Theorem 19

We present the details of the proof of Claim I and Claim II within the proof of Theorem 19.

Claim I. In Figure 5 the simulator is formulated as a master process expression, where $q(\mathbf{n}) = \mathtt{comsize}(\mathcal{F}) + \mathtt{comsize}(\mathcal{A})$, $c_{\mathtt{int}}$ and $\mathtt{state}$ are internal high channels, and "$\cdot$" means concatenation. The M-term $t_{=c'_1 c_3}$ accepts sequences of channels of the form $wc'_1w'c_3$ where w is some sequence which does not contain c'_1 and w' does not contain c_3. The M-term $t_{\neq c'_1 c_3}$ accepts all other sequences. Intuitively, if the state y of $\mathcal{S}$ is some sequence of the form $wc'_1w'c_3$ and $\mathcal{S}$ was triggered on $\mathtt{start}$, then this means that $\mathcal{F}$ was actived before, just read a message on c_3 but did not return a message on c'_2. In other words, the bit $\mathcal{F}$ obtained from the environment was 0.

$$
\begin{aligned}
\mathcal{S} \quad = \quad & \mathtt{in}(\mathtt{start}, x).\Big(\mathtt{out}(\mathtt{start}', x) \mid\mid \mathtt{out}(\mathtt{state}, \varepsilon) \mid\mid \\
& \quad !_{q(\mathbf{n})}\ \mathtt{in}(\mathtt{state}, y).\Big(\\
& \qquad \mathtt{in}(c'_1, z).\big(\mathtt{out}(c_1, z) \mid\mid \mathtt{out}(\mathtt{state}, y \cdot c'_1)\big) + \\
& \qquad \mathtt{in}(c'_2, z).\big(\mathtt{out}(c_2, z) \mid\mid \mathtt{out}(\mathtt{state}, y \cdot c'_2)\big) + \\
& \qquad \mathtt{in}(c_3, \mathtt{send\text{-}req}).\big(\mathtt{out}(c'_3, \mathtt{send\text{-}req}) \mid\mid \mathtt{out}(\mathtt{state}, y \cdot c_3)\big) + \\
& \qquad \mathtt{in}(\mathtt{start}, z).\Big(\mathtt{out}(c_{\mathtt{int}}, y) \mid\mid \\
& \qquad\quad \Big(\mathtt{in}(c_{\mathtt{int}}, t_{=c'_1 c_3}).\big(\mathtt{out}(c_2, 0) \mid\mid \mathtt{out}(\mathtt{state}, y \cdot \mathtt{start})\big)\Big) + \\
& \qquad\quad \Big(\mathtt{in}(c_{\mathtt{int}}, t_{\neq c'_1 c_3}).\big(\mathtt{out}(\mathtt{start}', z) \mid\mid \mathtt{out}(\mathtt{state}, y \cdot \mathtt{start})\big)\Big) \\
& \qquad \Big) \\
& \quad \Big) \\
& \Big)
\end{aligned}
$$

Figure 5: The simulator $\mathcal{S}$.

Let $\mathcal{A}'$ be obtained from $\mathcal{A}$ by replacing every occurrence of $\mathtt{start}$ by $\mathtt{start}'$. Next, we argue that $\mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{A}' \upharpoonright \mathcal{S} \upharpoonright \mathcal{F}$ for every $\mathcal{E} \in \mathbf{D}\text{-Env}(\mathcal{A} \upharpoonright \mathcal{P})$. Since $\mathcal{A}' \upharpoonright \mathcal{S} \in \mathbf{M}$, this implies $\mathtt{UC}_{(\mathbf{M},\mathbf{M},\mathbf{D})}(\mathcal{P}, \mathcal{F})$, and concludes the proof of Claim I.

First observe that $\mathcal{A}' \upharpoonright \mathcal{S} \in \mathbf{M}\text{-Adv}_{\mathcal{A} \upharpoonright \mathcal{P}}(\mathcal{F})$. Let $\mathcal{E} \in \mathbf{D}\text{-Env}(\mathcal{A} \upharpoonright \mathcal{P})$. We want to show that $\mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{A}' \upharpoonright \mathcal{S} \upharpoonright \mathcal{F}$: The simulator $\mathcal{S}$ records in its state the sequence of channels on which messages have been sent between $\mathcal{A}'$ and $\mathcal{F}$, and most of the time simply forwards messages between $\mathcal{A}'$ and $\mathcal{F}$. The only exception is if $\mathcal{S}$ reaches a state y accepted by $t_{=c'_1 c_3}$ after $\mathcal{S}$ received a message on $\mathtt{start}$. It is easy to see that $\mathcal{S}$ is in a state y (the value of y before reading a new message) accepted by $t_{=c'_1 c_3}$ iff $\mathcal{F}$ just received a message on c'_3 and then either returned a message on c'_2 or not. If a message is written on c'_2, then $\mathcal{S}$ simply forwards it on c_2, and otherwise, by the definition of reductions, a message is written on $\mathtt{start}$ which is read by $\mathcal{S}$. In the latter case, $\mathcal{S}$ outputs 0 on c_2 because if after reading a message on c'_3, the process $\mathcal{F}$ did not return a message, then this means that the bit $\mathcal{F}$ received from the environment was 0. From these arguments,

$\mathcal{E} \upharpoonright \mathcal{A} \upharpoonright \mathcal{P} \equiv \mathcal{E} \upharpoonright \mathcal{A}' \upharpoonright \mathcal{S} \upharpoonright \mathcal{F}$ easily follows.

Proof of Claim II. The environment $\mathcal{E}'$ is defined as follows:

$$
\begin{aligned}
\mathcal{E}' \quad = \quad & \mathtt{in}(\mathtt{start}, \varepsilon).\Bigg(\\
& \Big(\mathtt{in}(\mathtt{start}, \varepsilon).\mathtt{out}(\mathtt{decision}, 0)\Big) \;||\; \\
& \Big(\mathtt{out}(c_{\mathtt{in}}, T_{\mathtt{flip}}) \;||\; \\
& \quad \mathtt{in}(c_{\mathtt{in}}, x_0).\Big(\mathtt{out}(c_0, x_0) \;||\; \\
& \qquad \mathtt{in}(c_1, \mathtt{received}).\big(\mathtt{out}(c_3, \mathtt{send\text{-}req}) \;||\; \\
& \qquad\quad \mathtt{in}(c_2, x_1).\mathtt{out}(\mathtt{decision}, 1 \oplus x_0 \oplus x_1) \\
& \qquad \big) \\
& \quad \Big) \\
& \Big) \\
& \Bigg)
\end{aligned}
$$

where the C-term $T_{\mathtt{flip}}$ generates a random bit. Obviously we have $\mathtt{Prob}[(\mathcal{E}' \upharpoonright \mathcal{P})^{\mathbf{n} \leftarrow i} \rightsquigarrow 1] = 1$ for every i.

Now we show that $\mathtt{Prob}[(\mathcal{E}' \upharpoonright \mathcal{S}' \upharpoonright \mathcal{F})^{\mathbf{n} \leftarrow i} \rightsquigarrow 1] \leq 1/2$ for every i, and thus, $\mathcal{E}' \upharpoonright \mathcal{P} \not\equiv \mathcal{E}' \upharpoonright \mathcal{S}' \upharpoonright \mathcal{F}$. The process $(\mathcal{E}' \upharpoonright \mathcal{S}' \upharpoonright \mathcal{F})^{\mathbf{n} \leftarrow i}$ works as follows: First $\mathcal{E}'$ is activated (by reading a mesasge on `start`), then it sends a random bit on c_0 to $\mathcal{F}$, and expects to receive answer on c_1. After $\mathcal{F}$ has been activated, it outputs a bit on c_1', which activates $\mathcal{S}'$. If $\mathcal{S}'$ does not send a message on c_1, $\mathcal{E}'$ is activated (by reading a message on `start`) and outputs 0 on `decision`. So, we may assume that $\mathcal{S}'$ outputs `received` on c_1. Then, $\mathcal{E}'$ is activated again and sends `send-req` on c_3 to $\mathcal{S}'$. Now, if $\mathcal{S}'$ activates $\mathcal{F}$, the probability that $\mathcal{F}$ does not return output is 1/2 in which case $\mathcal{E}'$ would be activated and would output 0 on `decision`. Otherwise, $\mathcal{S}'$ does not send a message on c_3' to $\mathcal{F}$ but computes a bit by itself and sends it to $\mathcal{E}'$ (if it does not send a message on c_3 to $\mathcal{E}'$, again $\mathcal{E}'$ would be activated and output 0 on `decision`). The probability that this bit coincides with the one generated by $\mathcal{E}'$, i.e., that $1 = 1 \oplus x_0 \oplus x_1$, is 1/2. This shows that $\mathtt{Prob}[(\mathcal{E}' \upharpoonright \mathcal{S}' \upharpoonright \mathcal{F})^{\mathbf{n} \leftarrow i} \rightsquigarrow 1] \leq 1/2$ for every i, which concludes the proof of Claim II.