Unifying Equivalence-Based Definitions of Protocol Security*

Anupam Datta[1], Ralf Küsters[1], John C. Mitchell[1], Ajith Ramanathan[1], and Vitaly Shmatikov[2]

[1] Stanford University
[2] SRI International

Abstract. Several related research efforts have led to three different ways of specifying protocol security properties by simulation or equivalence. Abstracting the specification conditions away from the computational frameworks in which they have been previously applied, we show that when asynchronous communication is used, universal composability, black-box simulatability, and process equivalence express the same properties of a protocol. Further, the equivalence between these conditions holds for any computational framework, such as process calculus, that satisfies certain structural properties. Similar but slightly weaker results are achieved for synchronous communication.

1 Introduction

One appealing and relatively natural way to specify security properties is through simulation or equivalence. Focusing on protocols and equivalence, we can say what protocol P should achieve by giving an ideal functionality Q and saying that P be equivalent to Q in the face of attack. For example, P may be a key exchange protocol that operates over a public network, and Q an idealized protocol that uses some assumed form of private channel to generate and distribute shared keys. If no adversary can make P behave differently from Q, then since Q is impervious to attack by construction, we are assured that P cannot be successfully attacked. While this intuitive approach may seem clear enough, more precise formulations involve a number of details. For example, we may want to use one form of "ideal key exchange" with few messages to study several competing protocols. This ideal key exchange protocol is distinguishable from key exchange protocols that use different numbers of messages, but we can construct a simulator that uses the ideal key exchange primitive to produce additional messages. Thus a natural variation is not to expect P to be equivalent to Q, but ask that P be equivalent to some extension of Q that simulates P and retains the functionality of Q. Another issue is that we want users of the protocol to have the same positive outcome under all use scenarios.

The main advantage of specification by simulation or equivalence is composability: if protocol P is indistinguishable from ideal behavior Q, and protocol R is similarly indistinguishable from S, then P composed with R is indistinguishable from Q composed with S. Since many forms of security do not compose, the importance of composability should not be underestimated. Another advantage is generality: simulation and equivalence are meaningful when the protocol and the adversary operate in probabilistic polynomial time, and meaningful with nondeterministic computation and idealized cryptography.

We examine three similar specification approaches, and compare the methods over any computational framework satisfying familiar properties of process calculus. In this setting, we prove a very general correspondence: universal composability, black-box simulatability, and process equivalence express the same properties of a protocol, assuming asynchronous communication. Since our proofs hold for any process calculus that satisfies certain equational principles, our results are robust and not dependent on specialized properties of any specific computational setting. However, our results do not immediately apply to Turing machine models [7–11] or IO Automata models [18, 4] unless the assumed structural properties can be established for these models. If synchronous communication is available, one part of the equivalence becomes weaker because synchronous communication allows processes to detect an intermediate process acting as a buffer.

* This work was partially supported by NSF CCR-0121403, Computational Logic Tools for Research and Education, the OSD/ONR CIP/SW URI "Software Quality and Infrastructure Protection for Diffuse Computing," under ONR Grant N00014-01-1-0795, and the "Deutsche Forschungsgemeinschaft (DFG)".

Although our results may be most useful to researchers concerned with one of the three methods, some high-level points may be understood more broadly. First, rather than finding technical differences between competing approaches, we find that three approaches based on essentially similar intuition are in fact technically equivalent. Someone beginning to study this literature can therefore start with any of the approaches. Second, results proved about one form of specification may be transferred to other forms, simplifying the likely future development of this topic. Third, we believe that the equivalence of three different technical definitions, and the fact that this equivalence holds for a broad range of computational models, strongly suggests that there is a robust, fundamental notion underlying the three definitions.

Universal composability [7–9, 11, 10] involves a protocol to be evaluated, an ideal functionality, two adversaries, and an environment. The protocol has the ideal functionality if, for every attack on the protocol, there exists an attack on the ideal functionality, such that the observable behavior of the protocol under attack is the same as the observable behavior of the idealized functionality under attack. Each set of observations is performed by the same environment. Black-box simulatability [18, 12, 4] is a formally stronger notion in which the two attacks must be related in a uniform way. Black-box simulatability involves a protocol to be evaluated, an ideal functionality, a simulator, *one* adversary, and an environment. The protocol has the ideal functionality if there exists a simulator such that the protocol and simulation are indistinguishable by any user environment in the face of any network adversary. The difference between universal composability and black-box simulatability is that in the first case, for every attack on the protocol, there must be an attack on the ideal functionality. In the second case, the same is true, but the second attack must be the same as the first attack, interacting with the the ideal functionality through a fixed simulator. An essential difference between the adversary and the environment is that the adversary only has access to network communication, while the environment interacts with the system through input/output connections that are not accessible to the adversary.

While the first two methods were developed using sets of communicating Turing machines and probabilistic I/O automata, the third method was developed using process calculus. In the third method, associated with spi-calculus [2, 3], applied π-calculus [1], and a probabilistic polynomial-time process calculus [16, 14, 17, 19], a protocol P satisfies specification Q if P is observationally equivalent to Q. The specification Q may be the result of combining some ideal process and a simulator. Observational equivalence is a standard notion from the study of programming languages and concurrency theory [15]. Process P is observationally equivalent to Q, written $P \cong Q$ if, for every context $C[\ ]$ consisting of a process with a place to insert P or Q, the observable behavior of $C[P]$ is the same as $C[Q]$. The reason observational equivalence is relevant to security is that we can think of the context as an attack. Then $P \cong Q$ means that any attack on P must succeed equally well on Q, and conversely. In [16, 14, 17, 19], an asymptotic form of process equivalence is used, making observational equivalence the same as asymptotic indistinguishability under probabilistic polynomial-time attack.

Our main results are that with synchronous communication, process equivalence implies black box simulatability, and black box simulatability is equivalent to universal composability. With asynchronous communication, all three notions are equivalent. These results are demonstrated using formal proofs based on standard process calculus properties such as associativity of parallel composition, commutativity, renaming of private channels, scope extrusion, and congruence, together with a few facts about processes that buffer or forward messages from one channel to another. Since our proofs are based on relatively simple axioms, the proofs carry over to any process calculus that satisfies these reasonable and well-accepted equational principles. Although the likely equivalence between universal composability and black-box simulatability has been mentioned in other work [7], we believe this is the first general proof of a precise relationship; an independent proof of the equivalence of black-box simulatability and universal composability is presented for a specific model (I/O automata) in [5], which appeared between the time we submitted this paper for publication and the time it appeared. Previous work on universal composability and black-box simulatability is not situated in process calculus, making the kind of general result we present here, and comparison with process equivalence methods, difficult. In future work, we hope to extend our analysis to include communicating Turing machines (as in [7] and other work on universal composability) and I/O automata (as in [18, 4] and related work).

The rest of the paper is organized as follows. Section 2 describes process calculus syntax, the equational principles used in the rest of the paper, and the differences between synchronous and asynchronous communication. In Section 3, we define universal composability, black-box simulatability and process equivalence as relations on process calculus. Section 4 proves that universal composability is equivalent to black-box simulatability. Section 5 shows that while process equivalence and black-box simulatability are equivalent with asynchronous communication, the implication holds in one direction only with synchronous communication. The proofs presented in these sections rely just on the equational principles set forth in Section 2 and hence hold for any calculus in that class. Section 6 shows that two concrete calculi (the probabilistic polytime process calculus of [19, 20] and the spi-calculus [3]) satisfy the assumed equational principles and therefore the theorems hold for them. Finally, in Section 7, we summarize our conclusions and mention some directions for future work.

2 Process Calculus

Process calculus is a standard language for studying concurrency [15, 21] that has proved useful for reasoning about security protocols [3, 20]. Two main organizing ideas in process calculus are actions and channels. *Actions* occur on channels and are used to model communication flows. *Channels* provide an abstraction of the communication medium. In practice, channels might represent the communication network in a distributed system environment or the shared memory in a parallel processor. In this section, we describe a family of process calculi by giving a sample syntax and a set of equational principles. Two example calculi that satisfy our equational assumptions, spi-calculus [3] and the probabilistic polynomial-time process calculus of [20], are discussed Section 6.

A process calculus provides a syntax and an associated semantics. For concreteness, we will use the syntax defined by the following grammar, although additions to the language or changes in syntactic presentation are not likely to affect our results.

$$
\begin{array}{lll}
\mathcal{P} ::= & \mathbf{0} & \text{(termination)} \\
& \nu_c(\mathcal{P}) & \text{(private channel)} \\
& \mathtt{in}\,[c,x]\,.(\mathcal{P}) & \text{(input)} \\
& \mathtt{out}\,[c,T]\,.(\mathcal{P}) & \text{(output)} \\
& [T_1 = T_2].(\mathcal{P}) & \text{(match)} \\
& (\mathcal{P} \mid \mathcal{P}) & \text{(parallel composition)} \\
& !_{f(x)}.(\mathcal{P}) & \text{(bounded replication)}
\end{array}
$$

Intuitively $\mathbf{0}$ is the *empty process* taking no action. An input operator $\mathtt{in}\,[c,x]\,.\mathcal{P}$ waits until it receives a value on the channel c and then substitutes that value for the free variable x in $\mathcal{P}$. Similarly, an output $\mathtt{out}\,[c,T]\,.\mathcal{P}$ evaluates the term T, transmits that value on the channel c, and then proceeds with $\mathcal{P}$. Channel names that appear in an input or an output operation can be either public or private, with a channel being private if it is bound by a ν-operator and public otherwise. For convenience, we always α-rename channel names so that they are all distinct. The match operator $[T_1 = T_2]$ executes the process following iff T_1 have the T_2 value. The bounded replication operator has bound determined by the function f affixed as a subscript. The expression $!_{f(x)}.(\mathcal{P})$ is expanded to the $f(x)$-fold parallel composition $\mathcal{P} \mid \cdots \mid \mathcal{P}$ before evaluation.

Since an output process $\mathtt{out}\,[c,T]\,.(\mathcal{P})$ only proceeds when another process is ready to receive its input, this process calculus has synchronous communication. For maximal generality, we proceed using a synchronous calculus, constructing asynchronous channels when desired by inserting buffer processes. In an asynchronous setting, inserting an additional buffer on a channel would presumably have no effect, and our results would therefore remain valid.

2.1 Equational Principles

A process calculus syntax and semantics give rise to an equivalence relation $\cong$ called *observational equivalence*. Informally, two process calculus expressions are observationally equivalent if they produce the same observations, when executed in any context. Traditionally, observations are actions on public channels, with actions on a channel c bound by $\nu_c()$ private and unobservable.

We will assume the standard equational principles collected in Table 1. Rules TRN, SYM, and $CONG$ state that observational equivalence is a congruence. Rule $RENAME$ renames bound channels and $SCOPE$

allows us to "extrude" the scope of a private channel. Intuitively, with channels alpha-renamed apart, we can enlarge the scope of a channel binding without changing the observable behavior of the process. Rule $ZERO$ says that the zero process produces no observable activity. Rules COM and ASC reflect the associativity and commutativity of parallel composition.

$$\mathcal{P} \mid \mathcal{Q} \cong \mathcal{Q} \mid \mathcal{P} \tag{COM}$$

$$(\mathcal{P} \mid \mathcal{Q}) \mid \mathcal{R} \cong \mathcal{P} \mid (\mathcal{Q} \mid \mathcal{R}) \tag{ASC}$$

$$\mathbf{0} \mid \mathcal{P} \cong \mathcal{P} \tag{ZERO}$$

$$\frac{\sigma(c) = \sigma(d)}{\nu_c(\mathcal{P}) \cong \nu_d(\mathcal{P}^{[d/c]})} \tag{RENAME}$$

$$\frac{c \notin Channels(\mathcal{C}[\mathbf{0}])}{\nu_c(\mathcal{C}[\mathcal{P}]) \cong \mathcal{C}[\nu_c(\mathcal{P})]} \tag{SCOPE}$$

$$\frac{\mathcal{P} \cong \mathcal{Q}, \mathcal{Q} \cong \mathcal{R}}{\mathcal{P} \cong \mathcal{R}} \tag{TRN}$$

$$\frac{\mathcal{P} \cong \mathcal{Q}}{\mathcal{Q} \cong \mathcal{P}} \tag{SYM}$$

$$\frac{\mathcal{P} \cong \mathcal{Q}}{\forall \mathcal{C}[\ \ ] \in Con \colon \mathcal{C}[\mathcal{P}] \cong \mathcal{C}[\mathcal{Q}]} \tag{CONG}$$

Table 1. Equivalence Principles

For reasons that will become clear in later sections of the paper, we partition the set of public channel names into two infinite sets: the network channels and the input-output channels. We use the abbreviation *net* to refer to network channels and *io* for input-output channels. The difference between these two sets is that network channels will carry communication accessible to the adversary, while *io* channels allow users (the environment) to provide inputs to and observe the outputs produced by the protocol. We use ν_{net} to indicate binding $\nu_{n1}, \ldots, \nu_{nk}$ of all network channels in a process, and similarly ν_{io} for binding all *io* channels.

Throughout the paper, we use $\mathcal{P}$, $\mathcal{F}$, $\mathcal{A}$, and $\mathcal{S}$ (with superscripts if necessary) for processes that represent a real protocol, an ideal functionality, an adversary and a simulator. These may be arbitrary processes, except that we impose restrictions on the names of public channels that each may contain. Specifically, all public channel names in a protocol $\mathcal{P}$, an ideal functionality $\mathcal{F}$, and an adversary $\mathcal{A}$ must be network or input-output channels, while all public channel names in a simulator $\mathcal{S}$ must be network channels. For any given protocol $\mathcal{P}$, the *io* channels of an adversary $\mathcal{A}$ attacking $\mathcal{P}$ must be disjoint from the *io* channels of $\mathcal{P}$. The purpose of these restrictions is to allow the adversary, for example, to connect to the network channels of a protocol or ideal functionality, but not to its input-output channels. Also, by making all network channels private when a protocol $\mathcal{P}$ is combined with an adversary $\mathcal{A}$, we ensure that only the *io* channels are accessible to the environment.

2.2 Buffers, dummy adversaries, and asynchronous communication

One of the main differences between process equivalence and the two other relations is that process equivalence only involves one form of context (surrounding processes interacting with the protocol), as opposed to separate adversary and environment contexts in the other two relations. Therefore, while investigating the connection between process equivalence and the other relations, we will replace the adversary in the other definitions by a "dummy adversary" that does nothing but pass messages to the surrounding context. Also, since the underlying process calculus is assumed to be synchronous, we interpose "buffers" between processes to enforce asynchronous communication when desired. Consequently, our proofs require certain equational properties of buffers and simple processes that forward messages from one channel to another.

For any pair a and b of disjoint lists of channel names, both of the same length, we assume two processes $\mathcal{D}_a^b$ and $\mathcal{B}_a^b$, which we will call a dummy adversary and a buffer process, respectively. Intuitively, the axioms about $\mathcal{D}_a^b$ and $\mathcal{B}_a^b$ below state that these processes forward data between channels $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$, respectively. A dummy adversary may need to preserve message order to satisfy Axiom 1, but a buffer need not preserve message order. We assume that $\mathcal{D}_a^b$ and $\mathcal{B}_a^b$ have the channel names $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ free, and no other free channel names.

Axiom 1 (Dummy Adversary (DUMMY)). *Let $\mathcal{P}$ be a protocol and $\mathcal{A}$ be an adversary. Then $\nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net,dummy}(\mathcal{P} \mid \mathcal{D}_{net}^{dummy} \mid \mathcal{A}^{[dummy/net]})$ where dummy is a set of fresh channels of cardinality $|net|$ used to communicate between the dummy adversary and the modified adversary.*

Axiom 2 (Double Buffering (DBLBUF)). *Let $\mathcal{B}_a^b$, $\mathcal{B}_b^c$ and $\mathcal{B}_a^c$ be three buffers, for disjoint lists of channel names a, b, c of the same length. Then, $\nu_b(\mathcal{B}_a^b \mid \mathcal{B}_b^c) \cong \mathcal{B}_a^c$.*

Axiom 3 (Dummy and Buffer (DUMBUF)). *Let $\mathcal{B}_a^b$, $\mathcal{B}_b^c$ and $\mathcal{B}_a^c$ be three buffers and let $\mathcal{D}_b^c$ and $\mathcal{D}_a^b$ be dummy adversaries, for disjoint lists of channel names a, b, c of the same length. Then, $\nu_b(\mathcal{B}_a^b \mid \mathcal{D}_b^c) \cong \mathcal{B}_a^c$ and $\nu_b(\mathcal{D}_a^b \mid \mathcal{B}_b^c) \cong \mathcal{B}_a^c$*

Intuitively, Axiom 1 states that the interaction between a protocol and adversary through the network is indistinguishable from a situation when the communication between the protocol and the adversary is routed through the dummy adversary. Axiom 2 states that two buffers placed on a channel are indistinguishable from one buffer on that channel and Axiom 3 states that placing a dummy adversary and a buffer in sequence on a channel is equivalent to just having a buffer on that channel. Specific buffer and dummy adversary processes are presented in Section 6.

3 Security Definitions

In this section, we define three relations on processes, universal composability, black-box simulatability and process equivalence. These definitions are first presented in the synchronous form, then modified at the end of the section to assume asynchronous communication by placing buffers between process, adversary, and environment.

Definition 4. *Universal Composability: A protocol $\mathcal{P}$ is said to securely realize an ideal functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$ attacking the protocol, there exists an adversary $\mathcal{A}^*$ attacking the ideal functionality, such that no context can distinguish whether it is interacting with $\mathcal{P}$ and $\mathcal{A}$ or with $\mathcal{F}$ and $\mathcal{A}^*$. Formally,*

$$\forall \mathcal{A}. \exists \mathcal{A}^*.\ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\mathcal{F} \mid \mathcal{A}^*)$$

Figure 1 provides further intuition. The protocol as well as the ideal functionality communicate with the respective adversary processes over the network channels (denoted *net* in the figure). These channels are not visible to the context (or "environment" to use the terminology of [7, 18]). However, the context gets to communicate with these processes over the input-output channels (denoted *io* in the figure). All other channels of $\mathcal{P}$, $\mathcal{A}$, $\mathcal{F}$, and $\mathcal{A}^*$ are private. The intuition behind the distinction between channels is that if you are a user of SSL (Secure Sockets Layer), for example, your browser communicates with the implementation of SSL through *io* channels, while an attacker on the network has control of traffic on *net* channels.

Since the two process expressions in the definition of Universal Composability are observationally equivalent, this implies that if there is an attack on the real protocol, then there exists an equivalent attack on the ideal functionality. Hence, if the ideal functionality is impervious to attack by construction, then a real protocol that satisfies the above definition wrt the ideal functionality also cannot be attacked. While [7–9, 11, 10] discuss an adversary and environment, the environment here is provided by the context used in the definition of $\cong$.

In the definition of black-box simulatability and process equivalence, we use a simulator process whose public channels correspond to the union of the network channels of the ideal functionality (denoted *sim* below) and the network channels of the adversary (denoted *net* below).

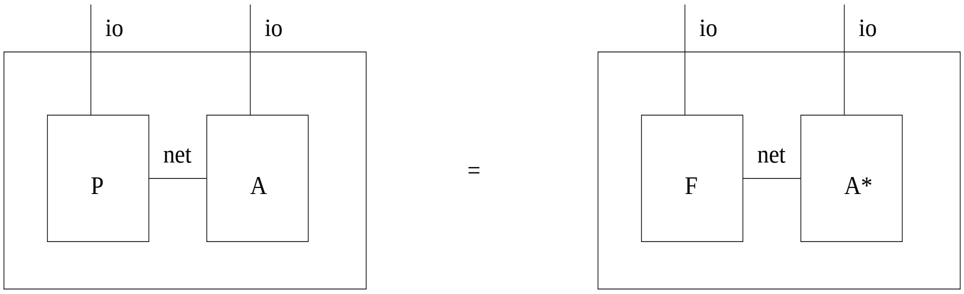

Fig. 1. Universal Composability

Definition 5. *Black-box Simulatability: A protocol $\mathcal{P}$ is said to securely realize an ideal functionality $\mathcal{F}$ if there exists a simulator $\mathcal{S}$ such that for any adversary $\mathcal{A}$, no context can distinguish whether it is interacting with $\mathcal{P}$ and $\mathcal{A}$ or with $\mathcal{F}$, $\mathcal{S}$ and $\mathcal{A}$. Formally,*

$$\exists \mathcal{S}. \forall \mathcal{A}.\ \ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\nu_{sim}(\mathcal{F} \mid \mathcal{S}) \mid \mathcal{A})$$

Figure 2 depicts this scenario. In effect, the simulator $\mathcal{S}$ uses the ideal functionality $\mathcal{F}$ to simulate the real protocol $\mathcal{P}$. The difference between universal composability and black-box simulatability is that in the first case, for every attack on the protocol, there must be an attack on the ideal functionality. In the second case, the same is true, but the second attack must be the same as the first attack, carried out on a simulation of the protocol that may rely on the ideal functionality.

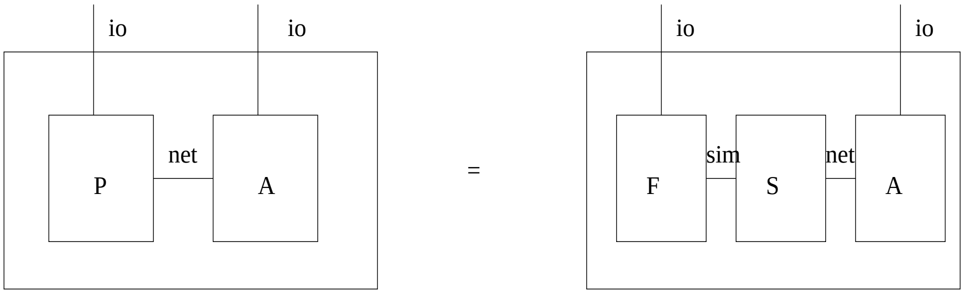

Fig. 2. Black Box Simulatability

Definition 6. *Process Equivalence: A protocol $\mathcal{P}$ is said to securely realize an ideal functionality $\mathcal{F}$ if there exists a simulator $\mathcal{S}$ such that no context can distinguish whether it is interacting with $\mathcal{P}$ or with $\mathcal{F}$ and $\mathcal{S}$. Formally,*

$$\exists \mathcal{S}.\ \ \mathcal{P} \cong \nu_{sim}(\mathcal{F} \mid \mathcal{S})$$

Figure 3 depicts this situation. Note that, unlike the first two definitions, the context has access to both the network and the input-output channels. Intuitively, the context used in the definition of observational equivalence serves the roles of both the adversary and the environment.

For each of these three relations, we formulate below corresponding asynchronous conditions by interposing message buffers or "bags" [15] on the network, input-output, and simulation channels. A buffer is any process satisfying the syntactic restrictions and axioms described in Section 2.2.

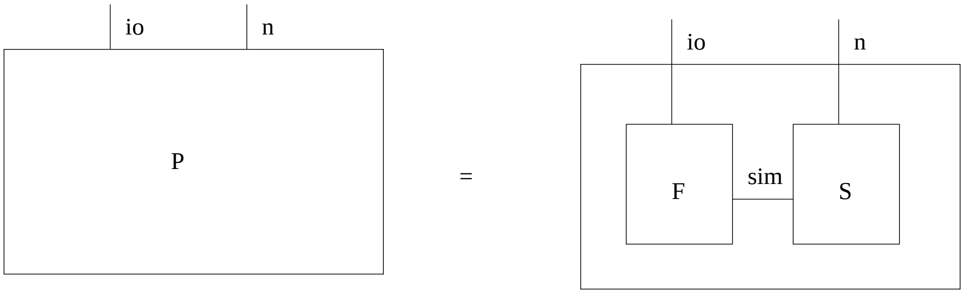

Fig. 3. Process Equivalence

$$UC : \forall\mathcal{A}.\exists\mathcal{A}^*.\ \nu_{ph,pn,an,ah}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{A} \mid \mathcal{B}^{h'}_{ah}) \cong \nu_{fh,fn,sn,sh}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sn}_{fn} \mid \mathcal{A}^* \mid \mathcal{B}^{h'}_{sh})$$
$$BB : \exists\mathcal{S}.\forall\mathcal{A}\ \nu_{ph,pn,h',ah'}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^{h'}_{pn} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) \cong \nu_{fh,fn,sn,sh,h',ah'}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sn}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{h'}_{sh} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'})$$
$$PE : \exists\mathcal{S}.\ \nu_{ph,pn}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^n_{pn}) \cong \nu_{fh,fn,sh,sn}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sh}_{fn} \mid \mathcal{S} \mid \mathcal{B}^n_{sn})$$

The binding of channels used in these definitions should be intuitively clear. In the UC condition, for example, $\mathcal{B}^h_{ph}$ buffers messages on $\mathcal{P}'s$ input-output channels; it forwards messages on the channels labelled h to $\mathcal{P}'s$ input-output channels (denoted ph). By binding the channels ph, we ensure that they are not observable by the environmental context. Similarly, $\mathcal{B}^{an}_{pn}$ and $\mathcal{B}^{h'}_{ah}$ buffer messages on the network channels between $\mathcal{P}$ and $\mathcal{A}$ and the io channels of $\mathcal{A}$.

4 Black-box Simulatability and Universal Composability

In this section, we prove that universal composability and black-box simulatability are equivalent for both synchronous and asynchronous communication.

Theorem 7. *Universal composability is equivalent to black-box simulatability with synchronous communication.*

Proof. $\Leftarrow$: Follows immediately by scope extrusion (**SCOPE**), associativity of parallel-or (**ASC**) and renaming of private channels (**RENAME**). The formal proof is given in Table 2. $\mathcal{A}^*$ is simply $\nu_{sim}(\mathcal{S}^R \mid \mathcal{A}^R)$. Thus, the combination of the simulator and the real adversary gives us the ideal process adversary demanded by the universal composability definition.

BB	$\exists\mathcal{S}.\forall\mathcal{A}.\ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\nu_{sim}(\mathcal{F} \mid \mathcal{S}) \mid \mathcal{A})$	(1)
(1), **SCOPE**, **ASC**	$\exists\mathcal{S}.\forall\mathcal{A}.\ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{sim}(\mathcal{F} \mid \nu_{net}(\mathcal{S} \mid \mathcal{A}))$	(2)
(2), **RENAME**	$\exists\mathcal{S}.\forall\mathcal{A}.\ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\mathcal{F}^R \mid \nu_{sim}(\mathcal{S}^R \mid \mathcal{A}^R))$	(3)
(3)	$\forall\mathcal{A}.\exists\mathcal{A}^*.\ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\mathcal{F}^R \mid \mathcal{A}^*)$	(4)

Table 2. Black-Box Simulatability implies Universal Composability (Synchronous Communication)

$\Rightarrow$: The formal proof is in Table 3. In Figure 4, the same proof is sketched out using intuitive diagrams of the form introduced in Section 3. We use standard process calculus proof rules: congruence (**CONG**), associativity of parallel-or (**ASC**), renaming of private channels (**RENAME**), and scope extrusion (**SCOPE**). The only step in the proof that does not immediately follow from our general equational principles rules is (4). We use the network-specific equivalence rule **DUMMY** (see Axiom 1) here. This rule captures the intuition that the environment cannot distinguish whether it is interacting with a process $\mathcal{P}$ and adversary

$\mathcal{A}$ or it is interacting with $\mathcal{P}$ and $\mathcal{A}$ where the communication between them is forwarded through a "dummy adversary", $\mathcal{D}$, which just forwards messages in the order in which it receives them. Naturally, a dummy adversary process has to be defined and the assumed equivalence has to be proven in any concrete calculus in which we wish to apply our general results. In particular, as discussed in a later section, the notion of a "dummy adversary" is made rigorous in Definition 16 and the equivalence proved in Lemma 18 for the probabilistic polytime process calculus of [20].

$$
\begin{array}{rll}
\textbf{UC} & \exists\mathcal{S}.\ \nu_{net}(\mathcal{P} \mid \mathcal{D}) \cong \nu_{net}(\mathcal{F} \mid \mathcal{S}) & (1)\\
(1), \textbf{CONG} & \exists\mathcal{S}.\forall\mathcal{A}.\ \nu_{a'}(\nu_{net}(\mathcal{P} \mid \mathcal{D} \mid \mathcal{A}^R)) \cong \nu_{a'}(\nu_{net}(\mathcal{F} \mid \mathcal{S}) \mid \mathcal{A}^R) & (2)\\
(2), \textbf{SCOPE}, \textbf{ASC} & \exists\mathcal{S}.\forall\mathcal{A}.\ \nu_{net}(\mathcal{P} \mid \nu_{a'}(\mathcal{D} \mid \mathcal{A}^R)) \cong \nu_{a'}(\nu_{net}(\mathcal{F} \mid \mathcal{S}) \mid \mathcal{A}^R) & (3)\\
(3), \textbf{DUMMY} & \exists\mathcal{S}.\forall\mathcal{A}.\ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{a'}(\nu_{net}(\mathcal{F} \mid \mathcal{S}) \mid \mathcal{A}^R) & (4)\\
(4), \textbf{RENAME} & \exists\mathcal{S}.\forall\mathcal{A}.\ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\nu_{sim}(\mathcal{F}^R \mid \mathcal{S}^R) \mid \mathcal{A}) & (5)
\end{array}
$$

Table 3. Universal Composability implies Black-Box Simulatability (Synchronous Communication)

Theorem 8. *Universal composability is equivalent to black-box simulatability with asynchronous communication.*

Proof. $\Leftarrow$: Follows immediately by scope extrusion (**SCOPE**), associativity of parallel-or (**ASC**) and renaming of private channels (**RENAME**). The formal proof is exactly the same as the one for the synchronous model. $\mathcal{A}^*$ is simply $\nu_{sn,an}(\mathcal{S} \mid \mathcal{B}^{an}_{sn} \mid \mathcal{A})$.

$\Rightarrow$: The formal proof is given in Table 4. The standard process calculus rules used in the proof are congruence (**CONG**) and scope extrusion (**SCOPE**). The two non-standard rules used are (**DBLBUF**) and (**DUMBUF**). As for (**DUMMY**) these rules need to be proven in any concrete calculus in which we wish to apply our general results. These rules are formally proved for the probabilistic polytime calculus in Lemma 19 and Lemma 20 respectively.

$$
\begin{array}{rll}
\textbf{UC} & \exists\mathcal{S}.\ \nu_{ph,pn,an,ah}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{D}^{ah}_{an} \mid \mathcal{B}^{h'}_{ah}) \cong \nu_{fh,fn,sn,sh}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sn}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{h'}_{sh}) & (1)\\
(1), \textbf{CONG} & \exists\mathcal{S}.\forall\mathcal{A}\ \nu_{ph,pn,an,ah,h',ah'}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{D}^{ah}_{an} \mid \mathcal{B}^{h'}_{ah} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) \cong & \\
 & \nu_{fh,fn,sn,sh,h',ah'}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sn}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{h'}_{sh} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) & (2)\\
(2), \textbf{SCOPE}, \textbf{DUMBUF} & \exists\mathcal{S}.\forall\mathcal{A}\ \nu_{ph,pn,an,h',ah'}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{B}^{h'}_{an} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) \cong & \\
 & \nu_{fh,fn,sn,sh,h',ah'}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sn}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{h'}_{sh} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) & (3)\\
(3), \textbf{SCOPE}, \textbf{DBLBUF} & \exists\mathcal{S}.\forall\mathcal{A}\ \nu_{ph,pn,h',ah'}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^{h'}_{pn} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) \cong & \\
 & \nu_{fh,fn,sn,sh,h',ah'}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sn}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{h'}_{sh} \mid \mathcal{A} \mid \mathcal{B}^{h''}_{ah'}) & (4)
\end{array}
$$

Table 4. Universal Composability implies Black-Box Simulatability (Asynchronous Communication)

5 Process Equivalence and Black-box Simulatability

Process equivalence and black-box simulatability are equivalent with asynchronous communication. With synchronous communication, however, process equivalence implies black-box simulatability but not conversely.

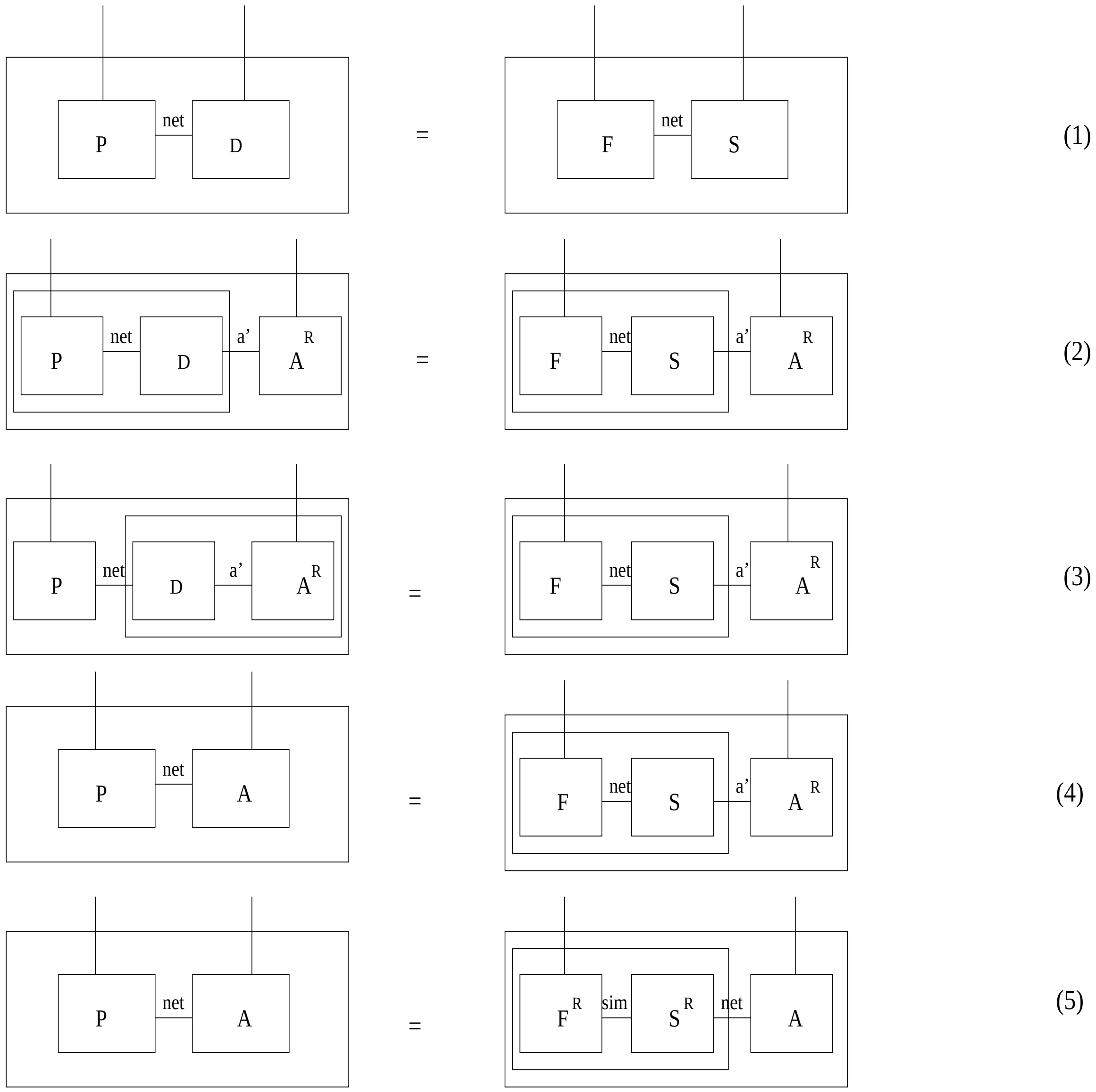

Fig. 4. Universal Composability implies Black Box Simulatability: Proof Sketch

Theorem 9. *Process equivalence implies black-box simulatability with synchronous communication.*

Proof. By definition, we have $\exists \mathcal{S}.\ \mathcal{P} \cong \nu_{sim}(\mathcal{F} \mid \mathcal{S})$. Hence, by the congruence rule, **CONG**, we have that $\exists \mathcal{S}. \forall \mathcal{A}.\ \nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net}(\nu_{sim}(\mathcal{F} \mid \mathcal{S}) \mid \mathcal{A})$. This is precisely the definition of black-box simulatability in the synchronous communication.

The reason that process equivalence is strictly stronger than black-box simulatability in the synchronous case is that when the the adversary and environment are combined into one surrounding process context, this context may use the global ordering of events on the *net* and *io* channels to distinguish between real and ideal processes. This global ordering is not available when the adversary and the environment are separate processes as in the definition of black-box simulatability. Consider the two processes $\mathcal{P} ::= \texttt{out}\,[io,\alpha]\,.\texttt{out}\,[io,\gamma]\,.\texttt{out}\,[net,\beta]$ and $\mathcal{Q} ::= \texttt{out}\,[io,\alpha]\,.\texttt{out}\,[net,\beta]\,.\texttt{out}\,[net,\gamma]$. These two processes satisfy the definition of black-box simulatability in a non-deterministic process calculus like spi-calculus (using a simulator that just forwards messages to the adversary). However, they do not satisfy the definition of process equivalence since the global ordering of observables on the *io* and *net* channels is α, γ, β in one case and α, β, γ in the other.

Theorem 10. *Process equivalence is equivalent to black-box simulatability with asynchronous communication.*

Proof. $\Rightarrow$: By definition, $\exists \mathcal{S}.\ \nu_{ph,pn}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^n_{pn}) \cong \nu_{fh,fn,sh,sn}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sh}_{fn} \mid \mathcal{S} \mid \mathcal{B}^n_{sn})$. Hence, by the congruence rule, **CONG**, we have $\exists \mathcal{S}.\forall \mathcal{A}.\ \nu_{ph,pn,an,ah}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{A} \mid \mathcal{B}^{h'}_{ah}) \cong \nu_{fh,fn,sh,sn,an,ah}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sh}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{an}_{sn} \mid \mathcal{A} \mid \mathcal{B}^{h'}_{ah})$. This is precisely the definition of black-box simulatability when communication is asynchronous. The proof follows the same line of reasoning as the one for synchronous communication.

$\Leftarrow$: The formal proof is in Table 5. Besides scope extrusion, it uses (**DBLBUF**) and (**DUMBUF**) to replace a dummy adversary and buffer process combination as well as two sequentially connected buffers by a single instance of a buffer process.

$$
\begin{array}{rll}
\textbf{BB} & \exists \mathcal{S}.\ \nu_{ph,pn,an,ah}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{D}^{ah}_{an} \mid \mathcal{B}^{h'}_{ah}) \cong & \\
& \nu_{fh,fn,sh,sn,an,ah}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sh}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{an}_{sn} \mid \mathcal{D}^{ah}_{an} \mid \mathcal{B}^{h'}_{ah}) & (1) \\
(1), \textbf{SCOPE}, \textbf{DUMBUF} & \exists \mathcal{S}.\ \nu_{ph,pn,an}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^{an}_{pn} \mid \mathcal{B}^{h'}_{an}) \cong & \\
& \nu_{fh,fn,sh,sn,an}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sh}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{an}_{sn} \mid \mathcal{B}^{h'}_{an}) & (2) \\
(2), \textbf{SCOPE}, \textbf{DBLBUF} & \exists \mathcal{S}.\ \nu_{ph,pn,an}(\mathcal{B}^h_{ph} \mid \mathcal{P} \mid \mathcal{B}^{h'}_{pn}) \cong & \\
& \nu_{fh,fn,sh,sn,an}(\mathcal{B}^h_{fh} \mid \mathcal{F} \mid \mathcal{B}^{sh}_{fn} \mid \mathcal{S} \mid \mathcal{B}^{h'}_{sn}) & (3)
\end{array}
$$

Table 5. Black-Box Simulatability implies Process Equivalence (Asynchronous Communication)

6 Applications to specific process calculi

In this section, we demonstrate that several standard process calculi used for reasoning about security protocols (the probabilistic polynomial-time process calculus of [20], the spi-calculus [3], and the applied π-calculus [1]) satisfy the equational principles used in the axiomatic proofs in the previous sections. The proved relations between the various security definitions therefore hold in these calculi.

6.1 Probabilistic Poly-time Process Calculus

A probabilistic polynomial-time process calculus (PPC) for security protocols is developed in [16, 14, 17]; the best current presentations are [19, 20]. It consists of a set of *terms* that do not perform any communications, *expressions* that can communicate with other expressions, and, *channels* that are used for communication. Terms contain variables that receive values over channels. There is also a special variable **n** called the *security parameter*. Each expression defines a set of *processes*, one for each choice of value for the security parameter. Each channel name has a bandwidth polynomial in the security parameter associated with it by a function called σ. The bandwidth ensures that no message gets too large and, thus, ensures that the expression can be evaluated in time polynomial in the security parameter.

The class of terms used must satisfy the following two properties:

1. If θ is a term with k variables, then there exists a probabilistic Turing machine M_θ with k inputs and a polynomial $q_\theta(x_1, \ldots, x_k)$ such that:
   (a) The term θ, with $a_1, \ldots, a_k$ substituted for its k variables, reduces to a with probability p if and only if $M_\theta(a_1, \ldots, a_k)$ returns a with probability p; and,
   (b) For any choice of $a_1, \ldots, a_k$ we have that $M_\theta(a_1, \ldots, a_k)$ halts in time at most $q_\theta(|a_1|, \ldots, |a_k|)$.
2. For each probabilistic polynomial-time function $f\colon \mathbb{N}^m \to \mathbb{N}$, there exists a term θ such that M_θ computes f.

Essentially, the term language completely captures the class of probabilistic polynomial-time Turing machines. One example of such a set of terms is based on a term calculus called OSLR studied in [16] (based in turn on [6, 13]).

Although any probabilistic polynomial-time function can be computed by a term, communication requires additional syntactic forms. *Expressions* of PPC are given by the grammar in Section 2. The contexts, *Con*, of PPC are obtained from the grammar by adding a placeholder symbol for a "hole" to be filled in, as usual.

Operational Semantics The evaluation of a variable-closed process proceeds in three steps: reduction, selection, and communication. In the *reduction step*, all terms and matches that are not in the scope of an input expression are evaluated. Since the expression is variable-closed and only inputs can bind variables, we know that every term outside the scope of an input has no free variables. This step simulates computation.

In the *selection step*, we use a probabilistic scheduler to select an action to perform. Actions include the silent action, τ; the input action $\mathtt{in}\langle c, a\rangle$ that reads the value a from the channel c into the variable x; the output action $\mathtt{out}\langle c, a\rangle$ that places the value a on the channel c; and the simultaneous action $\alpha \cdot \beta$ where one of α and β is an input action from the channel c of the value a and the other action is an output of the value a on the channel c obtained by using the action product $\cdot$ on α and β. We will say that two actions are of the same type if they are both inputs, outputs, or simultaneous actions with the same channel and value. The *scheduler* picks a particular type of simultaneous action from the set of available simultaneous action types according to the distribution defining the scheduler. However, silent actions must be performed if they are available since silent actions have higher priority. Then, one action of that type is picked uniformly at random from the set of available actions of that type. Further discussion may be found in [19].

In the *communication step*, we perform the indicated substitution taking care to truncate the value according to the bandwidth associated with the channel name. This is important for preserving the polynomial-time property of the process calculus.

We call this three-stage procedure an *evaluation step*; and evaluation proceeds in evaluation steps until the set of schedulable actions becomes empty. We refer the reader to [19] for more details.

Theorem 11. *Let P be a process. Then the evaluation of P can be performed in time polynomial in the security parameter.*

The proof proceeds by constructing a machine that evaluates P. The time-bound follows from the representation of terms and schedulers as probabilistic polynomial-time Turing machines.

A form of *weak probabilistic bisimulation* over asymptotically polynomial-time processes, or more simply *probabilistic bisimulation*, is developed in [19, 20] (see also [21]). Two processes P and Q are probabilistically bisimilar just when

1. If P can take an action α and with probability p become P', then Q must be able to take α to become processes $Q_1, \ldots, Q_k$ with total probability p; and,
2. If Q can take an action α and with probability p become Q', then P must be able to take α to become processes $P_1, \ldots, P_k$ with total probability p.

Using $\simeq$ to denote the bisimulation equivalence relation, [19, 20] show that $\simeq$ is a congruence.

Theorem 12. $\forall P, Q \in Proc. \forall C[\ \ ] \in Con\colon P \simeq Q \implies C[P] \simeq C[Q]$

Definition 13. *Let $\mathcal{P}$ and $\mathcal{Q}$ be two PPC expressions. Then $\mathcal{P} \cong \mathcal{Q}$ if, for sufficiently large n, $P^{\mathbf{n} \leftarrow n}$ is observationally indistinguishable from $Q^{\mathbf{n} \leftarrow n}$.*

A more precise definition can be found in [19, 20]. We also have the following theorem, proved in [19], which states that if two processes are probabilistically bisimilar, then they are observationally equivalent (in the sense of [19]). Hence, to prove observational equivalence, it is sufficient to demonstrate a probabilistic bisimulation.

Theorem 14. $\mathcal{P} \simeq \mathcal{Q} \implies \mathcal{P} \cong \mathcal{Q}$.

In [19], it is proved that all the equational principles of Table 1 hold in PPC. It remains to show that (**DUMMY**), (**DBLBUF**), and (**DUMBUF**) hold in PPC. We give below precise definitions of the dummy adversary and the buffer process in PPC, relegating proofs of the equivalences to Appendix A. Hence, we can conclude that the results proved in the previous sections about the relationship between the various security properties hold for PPC.

For simplicity we will construct uni-directional buffers, assuming that each public channel is *directional i.e.*, a channel name is used in a process only for inputs or only for outputs. We will say that a channel is an input channel (resp. output channel) just when it is to be used only for inputs (resp. outputs). Bi-directional buffers may be constructed by composing a pair of uni-directional channels.

Definition 15. *Let $A = \{a_1, \ldots, a_k\}$ and $B = \{b_1, \ldots, b_k\}$ be two equinumerous sets of channel names such that $a_i \in A$ is an input channel iff $b_i \in B$ is an output channel. We define $\mathcal{B}_{a_i}^{b_i}$ as*

$$!_{q(\cdot)}.\big(\texttt{in}\,[a_i, y]\,.\texttt{out}\,[b_i, y]\big)$$

in the case that a_i is an output channel, and

$$!_{q(\cdot)}.\big(\texttt{in}\,[b_i, y]\,.\texttt{out}\,[a_i, y]\big)$$

in the case that a_i is an input channel. Then we define the asynchronous buffer between A and B, *$\mathcal{B}_A^B$, as the expression $\mathcal{B}_{a_1}^{b_1} \mid \cdots \mid \mathcal{B}_{a_k}^{b_k}$.*

Essentially, an asynchronous buffer forwards messages between channels in A and channels in B without preserving any message-ordering since, for example, it is possible that an input on a_i is read, then a second input on a_i is read and forwarded onto b_i before the first input on a_i is forwarded onto b_i.

Definition 16. *Let $A = \{a_1, \ldots, a_k\}$ and $B = \{b_1, \ldots, b_k\}$ be two equinumerous sets of channel names such that $a_i \in A$ is an input channel iff $b_i \in B$ is an output channel. We define $\mathcal{D}_{a_i}^{b_i}$ as*

$$\texttt{in}\,[a_i, y]\,.\texttt{out}\,[b_i, y]\,.\texttt{out}\,[syn_i, 1] \mid\, !_{q(\cdot)}.\big(\texttt{in}\,[syn_i, x]\,.\texttt{in}\,[a_i, y]\,.\texttt{out}\,[b_i, y]\,.\texttt{out}\,[syn_i, 1]\big)$$

in the case that a_i is an output channel, and

$$\texttt{in}\,[b_i, y]\,.\texttt{out}\,[a_i, y]\,.\texttt{out}\,[syn_i, 1] \mid\, !_{q(\cdot)}.\big(\texttt{in}\,[syn_i, x]\,.\texttt{in}\,[b_i, y]\,.\texttt{out}\,[a_i, y]\,.\texttt{out}\,[syn_i, 1]\big)$$

in the case that a_i is an input channel. Then we define the dummy adversary between A and B, *$\mathcal{D}_A^B$, as the expression*

$$\nu_{syn}(\mathcal{D}_{a_1}^{b_i} \mid \cdots \mid \mathcal{D}_{a_k}^{b_k})$$

The expression $\mathcal{D}_A^B$ simply forwards communications between each channel $a_i \in A$ and $b_i \in B$. The channel syn_i is used to synchronize between the various inputs and outputs on the channel a_i in $\mathcal{D}_A^B$ to avoid situations where, for example, a value has been read on the channel b_i and, before it is forwarded, a new value is read on the channel b_i and then forwarded. Essentially, the use of syn_i allows us to preserve the ordering on communications on a_i by guaranteeing that if $\mathcal{D}_A^B$ receives the message o before o', it will transmit o before o'. Thus a dummy adversary is just a message-order-preserving buffer.

Theorem 17. *The equivalence principles* (**DUMMY**), (**DBLBUF**), *and* (**DUMBUF**) *hold in PPC.*

We prove these equivalences by constructing a probabilistic bisimulation and then applying Theorem 14. Proof sketches are available in Appendix A.

6.2 Spi-Calculus and Applied π-Calculus

Spi-calculus [3] and applied π-calculus [1] are two other process calculi that have been used to reason about security protocols. All the standard structural equivalence rules: associativity of parallel composition (**ASC**), renaming of private channels (**RENAME**), scope extrusion (**SCOPE**), congruence (**CONG**), which were collected in Table 1, hold in these calculi. The network-specific equivalences are also satisfied with appropriate definitions of dummy adversary and buffer processes. Hence the results proved in Section 4 and Section 5 also hold for these calculi. A representative proof for spi-calculus is given in Appendix B.

7 Conclusions

We compare three similar ways of specifying protocol properties by formulating *universal composability*, *black-box simulatability*, and *process equivalence* over process calculus. Our main results are that all three are equivalent when asynchronous communication is used; with synchronous communication, the first two are equivalent and implied by the third. While some process calculi provide synchronous communication, the asynchronous case is closer to computational practice. Although we model asynchronous communication by adding buffers to a synchronous calculus, we conjecture that the same results could also be achieved starting with a purely asynchronous form of process calculus.

Since universal composability, black-box simulatability, and process equivalence are all based on similar intuition about specifying security properties using indistinguishability, it is reassuring to know that they can be proved technically equivalent. We expect this equivalence to be useful in further research, since it allows us to transfer results about one form of specification to other forms. In addition, the equivalence of three different technical definitions, and the fact that this equivalence holds for a broad range of computational models, indicate the mathematical robustness of the underlying concept.

Our proofs use standard process calculus proof rules such as associativity of parallel composition, commutativity, renaming of private channels, scope extrusion, and congruence. The only subtlety is that two processes communicating over a private channel must be observationally equivalent to two processes communicating through a dummy process that just forwards messages in both directions. Therefore, the proofs will carry over to any process calculus that has the necessary features (such as private channels) and satisfies reasonable and well-accepted equational principles. In future work, we hope to extend our arguments to cover communicating Turing machines (as in [7] and other work on universal composability) and I/O automata (as in [18] and related work).

Acknowledgements Thanks to Andre Scedrov and Paulo Mateus 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. 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.
5. Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In *Proceedings of 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*. Springer, 2004.
6. Stephen Bellantoni. *Predicative Recursion and Computational Complexity*. PhD thesis, University of Toronto, 1992.
7. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symp. on the Foundations of Computer Science*. IEEE, 2001. Full version available at `http://eprint.iacr.org/2000/067/`.

8. 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.
9. 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.
10. 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.
11. 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.
12. Ivan Damgard and Jesper B. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Proc. CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264, Santa Barbara, California, 2003. Springer.
13. Martin Hofmann. *Type Systems for Polynomial-Time Computation.* Habilitation Thesis, Darmstadt; see `www.dcs.ed.ac.uk/home/mxh/papers.html`, 1999.
14. 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.
15. Robin Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 1989.
16. 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.
17. 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.
18. Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–200, Washington, 2001.
19. 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/`, 2003.
20. 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 [19]; to appear.
21. Robert J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, generative, and stratified models of probabilistic processes. *International Journal on Information and Computation*, 121(1), August 1995.

A Proofs of Lemmas for PPC

Lemma 18 (Dummy Adversary). *Let $\mathcal{P}$ be a protocol and $\mathcal{A}$ be an adversary. Then $\nu_{net}(\mathcal{P} \mid \mathcal{A}) \cong \nu_{net,dummy}(\mathcal{P} \mid \mathcal{D}_{net}^{dummy} \mid \mathcal{A}^{[dummy/net]})$ where dummy is a set of fresh channels of cardinality $|net|$ used to communicate between the dummy adversary and the modified adversary.*

Proof. (Sketch) From Definition 16, we see that $\mathcal{D}_{net}^{dummy}$ is of the form $\nu_{syn}(Du_{net}^{dummy})$. The proof proceeds by showing that for every path π in the evaluation graph of $\nu_{net}(P \mid A)$ there is a unique path π' in the evaluation graph of $\nu_{net,dummy}(P \mid \nu_{syn}(Du_{net}^{dummy}) \mid A^{[dummy/net]})$ such that the probability of every edge in π is preserved by the matching edge in π', and vice versa. It is easy to see that every transition of $\nu_{net}(P \mid A)$ that does not signify a communication between P and A can be identified immediately with transitions of $\nu_{net,dummy}(P \mid \nu_{syn}(Du_{net}^{dummy}) \mid A^{[dummy/net]})$. The only concern are those communications between P and A. These, though, can be simulated via a three stage procedure. Without loss of generality let us assume that a message is going from A to P along a *net* channel. In the first stage, the message is transmitted from $A^{[dummy/net]}$ to the dummy. In the second stage, the message is transmitted along the appropriate *net* channel to P. Finally, a synchronization bit in the dummy is transmitted to allow further communications to occur. While there is only one choice of transition for the first two stages, there are several choices for the third since any one of the $q(i)$ inputs for the synchronization bit in the dummy can receive the message.

However, each of the processes obtained after the third step are structurally identical (they only vary in which of the $q(i)$ remaining bridges between the *dummy* channel and its corresponding *net* channel accepts the synchronization bit). Thus we can identify them and replace the final $q(i)$-fold step with a single step. This means that every transition of $\nu_{net}(P \mid A)$ can be uniquely matched with either a transition or a length-three path of $\nu_{net,dummy}(P \mid \nu_{syn}(Du_{net}^{dummy}) \mid A^{[dummy/net]})$. The uniqueness of the mapping follows from its construction. Thus we can infer the desired bisimilarity which implies observational equivalence.

Lemma 19 (Double Buffering). *Let $\mathcal{B}_a^b$, $\mathcal{B}_b^c$ and $\mathcal{B}_a^c$ be asynchronous buffers. Then, $\nu_b(\mathcal{B}_a^b \mid \mathcal{B}_b^c) \cong \mathcal{B}_a^c$.*

Proof. We will show that $\nu_b(\mathcal{B}_a^b \mid \mathcal{B}_b^c)$ is probabilistically bisimilar to $\mathcal{B}_a^c$ from which it follows that the two expressions are observationally equivalent. For simplicity, we will assume that $|a| = |b| = |c| = 1$. Let us also assume that a is an input channel (whence c must be an output channel). In that case $\mathcal{B}_a^c$ is just

$$!_{q(\cdot)}.\big(\texttt{in}\,[a,y]\,.\texttt{out}\,[c,y]\big)$$

and $\nu_b(\mathcal{B}_a^b \mid \mathcal{B}_b^c)$ is just

$$\nu_b(!_{q(\cdot)}.\big(\texttt{in}\,[a,y]\,.\texttt{out}\,[b,y]\big) \mid !_{q(\cdot)}.\big(\texttt{in}\,[b,y]\,.\texttt{out}\,[c,y]\big))$$

We will identify expressions equivalent by virtue of the associativity and commutativity of $|$, the parallel composition operator, and the equivalence $\mathcal{P} \mid \mathbf{0} \cong \mathcal{P}$. Making use of these identifications, it is easy to verify that the set of pairs

$$\{\langle !_{q(\cdot)}.\big(\texttt{in}\,[a,y]\,.\texttt{out}\,[c,y]\big) \mid !_{q(\cdot)}.\big(\texttt{out}\,[c,y]\big), \\ \nu_b(!_{q(\cdot)}.\big(\texttt{in}\,[a,y]\,.\texttt{out}\,[b,y]\big) \mid !_{q(\cdot)}.\big(\texttt{in}\,[b,y]\,.\texttt{out}\,[c,y]\big)) \mid !_{q(\cdot)}.\big(\texttt{out}\,[c,y]\big)\rangle\}$$

is a suitable probabilistic bisimulation.

Lemma 20 (Dummy and Buffer). *Let $\mathcal{B}_a^b$, $\mathcal{B}_b^c$ and $\mathcal{B}_a^c$ be three asynchronous buffers and let $\mathcal{D}_b^c$ and $\mathcal{D}_a^b$ be dummy adversaries. Then, $\nu_b(\mathcal{B}_a^b \mid \mathcal{D}_b^c) \cong \mathcal{B}_a^c$ and $\nu_b(\mathcal{D}_a^b \mid \mathcal{B}_b^c) \cong \mathcal{B}_a^c$*

An asynchronous buffer just forwards messages without preserving the message order. Thus, intuitively, placing it after any structure that preserves message order or before any such structure should be the same as just using the asynchronous buffer. The formal proof is similar to the proof of Lemma 19 and is omitted due to space constraints.

B Proof of Equivalence for Spi-Calculus

Theorem 21. *Theorem 7 and Theorem 9 hold for spi-calculus.*

Proof. The standard equivalence rules used in proving the two theorems: associativity of parallel-or (**ASC**), renaming of private channels (**RENAME**), congruence (**CONG**), and scope extrusion (**SCOPE**) hold in spi-calculus. The only non-standard step corresponds to **DUMMY**. The proof relies on the observation that the situation in which processes $\mathcal{P}$ and $\mathcal{A}$ communicate over a private channel is observationally equivalent to the one in which all such communication is routed through a dummy process that just forwards messages in both directions. For simplicity, we consider only the case when there are two channels c_0 and c_1 between P and A. Since channels are directional, without loss of generality, we assume that channel c_0 is from A to P (*i.e.*, only A outputs messages on c_0, and only P receives messages on c_0), and channel c_1 is from P to A. The proof extends directly to the multiple-channel case.

Rewriting the statement using spi-calculus formalism and letting A^d stand for $A[d/c]$, we wish to demonstrate that

$$(\nu c_0, c_1)(P \mid A) \quad \simeq \quad (\nu c_0, c_1, d_0, d_1)(P \mid ((\nu s_0, s_1)(D_0 \mid D_1) \mid A^d))$$

where

$$D_0 = d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle \mid !\ s_0(x).d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle \\ D_1 = d_1(y).\overline{c_1}\langle y\rangle.\overline{s_1}\langle 1\rangle \mid !\ s_1(x).d_1(y).\overline{c_1}\langle y\rangle.\overline{s_1}\langle 1\rangle$$

We outline the proof in the direction $(\nu c_0, c_1)(P \mid A) \sqsubseteq (\nu c_0, c_1, d_0, d_1)(P \mid ((\nu s_0, s_1)(D_0 \mid D_1) \mid A^d))$. The proof in the other direction is similar. For our purposes, it is sufficient to recall that, informally, P *passes a test* (R, β) if P produces an observable on a channel named β when run in parallel with R. By definition, $P_1 \sqsubseteq P_2$ if, for any test (R, β) passed by P_1, P_2 also passes the test.

Let (R, β) be some test passed by $(\nu c)(P \mid A)$. By Proposition 4 [3], this implies that there exist an agent A and a process Q such that $(\nu c_0, c_1)(P \mid A)|R \xrightarrow{\tau}^{*} Q$ and $Q \xrightarrow{\beta} A$. Since we assume that P and A communicate only via channels c_0 and c_1, every reaction of $P \mid A$ is a reaction of P, a reaction of A, or an interaction between P and A. In the latter case, because we assumed that channels are directional, $P = c_0(x).P', A = \overline{c_0}\langle m\rangle.A', P|A \xrightarrow{\tau} P'[m/x]|A'$, or $P = \overline{c_1}\langle m\rangle.P', A = c_1(x).A', P|A \xrightarrow{\tau} P'|A'[m/x]$. To prove the lemma by induction over all reactions of $(\nu c_0, c_1)(P|A)|R$, it is sufficient to demonstrate that, if $P = c_0(x).P', A = \overline{c_0}\langle m\rangle.A', c_0(x).P'|\overline{c_0}\langle m\rangle.A' \xrightarrow{\tau} P'[m/x]|A'$, then $c_0(x).P'|((\nu s_0, s_1)(D_0|D_1)|\overline{d_0}\langle m\rangle.A'^d) \xrightarrow{\tau} P'[m/x]|((\nu s_0, s_1)(D_0|D_1)|A'^d)$. The proof for the case $P = \overline{c_1}\langle m\rangle.P', A = c_1(x).A', \overline{c_1}\langle m\rangle.P'|c_1(x).A' \xrightarrow{\tau} P'|A'[m/x]$ is symmetric.

$$
\begin{array}{l}
c_0(x).P'|((\nu s_0, s_1)(D_0|D_1)|\overline{d_0}\langle m\rangle.A'^d) \ = \\
c_0(x).P'|((\nu s_0, s_1)((d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle|!\ s_0(x).d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle)|D_1)|\overline{d_0}\langle m\rangle.A'^d) \ \xrightarrow{\tau} \\
c_0(x).P'|((\nu s_0, s_1)((\overline{c_0}\langle m\rangle.\overline{s_0}\langle 1\rangle|!\ s_0(x).d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle)|D_1)|A'^d) \ \xrightarrow{\tau} \\
P'[m/x]|((\nu s_0, s_1)((\overline{s_0}\langle 1\rangle|!\ s_0(x).d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle)|D_1)|A'^d) \ \xrightarrow{\tau} \\
P'[m/x]|((\nu s_0, s_1)((d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle|!\ s_0(x).d_0(y).\overline{c_0}\langle y\rangle.\overline{s_0}\langle 1\rangle)|D_1)|A'^d) \ = \\
P'[m/x]|((\nu s_0, s_1)(D_0|D_1)|A'^d)
\end{array}
$$