Modular Verification of Protocol Equivalence in the Presence of Randomness

Matthew S. Bauer[1] and Rohit Chadha[2]Mahesh Viswanathan[1]

[1] University of Illinois at Urbana-Champaign
[2] University of Missouri

Abstract. Security protocols that provide privacy and anonymity guarantees are growing increasingly prevalent in the online world. The highly intricate nature of these protocols makes them vulnerable to subtle design flaws. Formal methods have been successfully deployed to detect these errors, where protocol correctness is formulated as a notion of equivalence (indistinguishably). The high overhead for verifying such equivalence properties, in conjunction with the fact that protocols are never run in isolation, has created a need for modular verification techniques. Existing approaches in formal modeling and (compositional) verification of protocols for privacy have abstracted away a fundamental ingredient in the effectiveness of these protocols, randomness. We present the first composition results for equivalence properties of protocols that are explicitly able to toss coins. Our results hold even when protocols share data (such as long term keys) provided that protocol messages are tagged with the information of which protocol they belong to.

1 Introduction

Cryptographic protocols are often analyzed in the so-called *symbolic model*, where the assumption of perfect cryptography is made. Messages are symbolic terms modulo an equational theory (as opposed to bit-strings) and cryptographic operations are modeled via equations in the theory. The threat model is that of the *Dolev-Yao* attacker [33], in which the attacker has the ability to read, intercept and replay all messages on public channels and can also non-deterministically inject its own messages into the network. Verification techniques in this domain are fairly mature and a number of sophisticated analysis tools have been developed [11, 34, 7].

Automated tools based on Dolev-Yao analysis are fundamentally limited to protocols that are purely non-deterministic, where non-determinism is used to model concurrency as well as the interaction between protocol participants with their environment. The order and nature of these interactions is determined entirely by an attacker (also known as a scheduler) who resolves all non-determinism. There are, however, a large class of protocols whose correctness depends on an explicit ability to model and reason about coin tosses. With privacy goals in mind, these protocols lie at the heart of many anonymity systems such as Crowds [49], mix-networks [19], onion routers [37] and Tor [32]. Randomization is also used in cryptographic protocols to achieve fair exchange [10, 35], voter privacy in electronic voting [51] and denial of service prevention [39]. The privacy and anonymity properties achieved by these systems are often formulated in terms protocol equivalence (indistinguishability). For example, protocol equivalence is used in the analysis of properties like anonymity, unlinkability, and vote privacy [30, 5].

Catherine Meadows, in her summary of the over 30 year history of formal techniques in cryptographic protocol analysis [45, 46], identified the development formal analysis techniques for anonymous communication systems as a fundamental and still largely unsolved challenge. The main difficulty in adapting Dolev-Yao analysis to such randomized protocols has been the subtle interaction between non-determinism and randomization — if the attacker is allowed to "observe" the results of the private coin tosses in its scheduling decisions, then the analysis may reveal "security flaws" in correct protocols (see examples in [20, 12, 36, 17, 16]). In order to circumvent this problem, many authors [29, 20, 12, 36, 17, 16, 9, 15] have proposed that protocols be analyzed only with respect to attackers that are forced to take the same action in any two protocol executions that are indistinguishable. For the indistinguishability relation on traces, we propose [9, 15] trace-indistinguishability of applied-pi calculus processes [1]. In this framework, an attacker is a function from *traces*, the equivalence classes on executions under the trace-indistinguishability relation, to the set of attacker actions.

We consider the problem of composition for randomized protocols when the protocols are allowed to share data, such as keys. Our focus here is on equivalence properties. Two randomized protocols P and Q are said to be trace equivalent [15], if for each attacker $\mathcal{A}$ and trace t, the measure of executions in the trace t obtained when $\mathcal{A}$ interacts with protocol P is exactly the same as the measure of executions in the trace t obtained when $\mathcal{A}$ interacts with protocol Q. The protocols themselves are specified as processes in an applied pi-style calculus, parametrized by an equational theory that models cryptographic primitives. The protocols in our formalism are *simple*; a protocol is said to be simple if there is no principal-level nondeterminism [25]. As observed in [15], this notion of indistinguishability coincides with the notion of trace-equivalence for simple non-randomized protocols.

Contributions: We begin by considering the case when the number of sessions in a protocol is bounded. Our first result (Theorem 1 on Page 13) captures the conditions under which the composition of equivalent protocols under disjoint equational theories preserves trace equivalence. Formally, consider trace equivalent protocols P and Q over equational theory E_a, and trace equivalent protocols P' and Q' over equational theory E_b, where E_a and E_b are disjoint. We show that the composition of P and P' is equivalent to the composition of Q and Q', provided the shared secrets between P and P' and those between Q and Q' are kept with probability 1. While such a result also holds for non-randomized protocols (see [27, 4] for example), randomization presents its own challenges.

The first challenge arises from the fact that even if P' and Q' do not leak shared secrets (with P and Q, respectively), they may still reveal the equalities that hold amongst the shared secrets. Revealing these equalities may, in some cases, allow the attacker to infer the result of a private coin toss (See Example 7 on Page 15). Consequently, our composition theorem requires that P and Q remain trace equivalent even when such equalities are revealed. The revelation of the equalities is achieved by adding actions to protocols P and Q that reveal "hashes" of shared secrets.

As in the case of non-randomized protocols [27, 4], the proof proceeds by showing that it suffices to consider the case when P (Q) does not share any secrets with P' (Q' respectively). This is achieved by observing that if the composition of P and P' is not trace equivalent to the composition of Q and Q', then there must be a trace t and an individual execution ρ in the composition of P and P' (or of Q and Q') such that ρ belongs to t and there is no execution in the composition of Q and Q' (or of P and P' respectively) in the trace t. It is then observed that if the shared secrets between P and P', and between Q and Q', are *re-initialized* to fresh values respecting the same equalities amongst them as in the execution ρ, then the transformed protocols continue to remain trace inequivalent. For randomized protocols, we no longer have an individual execution that witnesses protocol inequivalence. Instead we have an attacker $\mathcal{A}$ and a trace t which occurs with different probabilities when the protocols interact with $\mathcal{A}$. Observe that the executions corresponding to the trace t will then form a tree, and different equalities amongst the shared secrets may hold in different branches. Thus, a simple transformation as in the case of non-randomized protocols no longer suffices (see Example 5 on Page 14). Instead, we have to perform a non-trivial inductive argument (with induction on number of coin tosses) to show that it suffices to consider the case when P (Q) does not share any secrets with P' (Q' respectively).

Our second result concerns the case when the equational theories E_a and E_b are the same, each containing cryptographic primitives for symmetric encryption, symmetric decryption and hashes (see Theorem 2 on Page 17). For this case, we show that the composition of randomized protocols preserves trace equivalence when the protocols are allowed to share secrets, provided protocol messages are tagged with the information of which protocol they belong to. As in the case of non-randomized protocols, this is achieved by showing that in presence of tagging, the protocols can be transformed to new protocols $P_{\mathsf{new}}, P'_{\mathsf{new}}, Q_{\mathsf{new}}, Q'_{\mathsf{new}}$ such that P_{new} and Q_{new} are trace equivalent protocols with equational theory E_{new}, and P'_{new} and Q'_{new} are trace equivalent protocols with disjoint equational theory E'_{new}. Thus, this result follows from our first result.

Our final result extends the above result to the case of unbounded number of sessions (see Theorem 3 on Page 18). We again consider the case when the equational theories E_a and E_b are the same, containing cryptographic primitives for symmetric encryption, symmetric decryption and hashes. In order to achieve this result, we additionally require that messages from each session are tagged with a unique session identifier.

Related Work. For the non-randomized case, a number of papers have identified requirements for proving protocol compositions secure. Safety properties are considered in [28, 42, 40, 41, 2, 24, 26, 6, 31, 21, 27, 47, 4] and indistinguishability properties in [3, 4]. A recent work [9] has also explored the composition of randomized protocols with respect to reachability properties. In the computational model, the problem of composing protocols securely has been studied in [13, 14]. Our result is most closely related to [3, 4, 9].

Much of research effort on mechanically analyzing anonymity systems has used techniques based on model checking and simulation. For example, [53] uses the PRISM model checker [44] to analyze the Crowds system. While these works are valuable, the techniques are ad-hoc in nature, and don't naturally extend to larger classes of protocols. In [52], an analysis of Chaum's Dinning cryptographers protocol [18] was carried out the in CSP framework [43]. In all of these works, the messages constructed by the attacker is assumed to be bounded. [15] consider the complexity of the problem of verifying bounded number of sessions of simple randomized cryptographic protocols that use symmetric and asymmetric encryption. They show that checking secrecy properties is **coNEXPTIME**-complete and the problem of checking indistinguishability is decidable in **coNEXPTIME**. In contrast, both of these problems are known to be **coNP**-complete for non-randomized protocols [25, 22, 8, 50]. The increased overhead in the verification effort that comes with the introduction of randomization in protocols places a premium on modular verification techniques, allowing smaller analysis efforts to be stitched together.

2 Preliminaries

We start by reviewing some standard notions for describing systems that exhibit both non-deterministic and probabilistic behavior. This begins with treatment of Markov chains in Section 2.1 followed by a discussion on partially observable Markov decision processes (POMDPs) in Section 2.2. POMDPs will form the basis for the semantics of our process calculus, which is paramaterized by an equational theory, the specifics of which we discuss in Section 2.3.

2.1 Probability spaces, Markov chains

We will assume the reader is familiar with probability spaces and Markov chains and give only the necessary definitions. A (sub)-probability space on S is a tuple $\Omega = (X, \Sigma, \mu)$ where Σ is a σ-algebra on X and $\mu : \Sigma \to [0, 1]$ is a countably additive function such that $\mu(\emptyset) = 0$ and $\mu(X) \leq 1$. The set Σ is said to be the set of events and μ the (sub)-probability measure of Ω. For $F \in \Sigma$, the quantity $\mu(F)$ is said to be the probability of the event F. If $\mu(X) = 1$ then we call μ a probability measure. Given two (sub)-probability measures μ_1 and μ_2 on a measure space (S, Σ) as well as a real number $p \in [0, 1]$, the convex combination $\mu_1 +_p \mu_2$ is the (sub)-probability measure μ such that for each set $F \in \Sigma$ we have $\mu(F) = p \cdot \mu_1(F) + (1 - p) \cdot \mu_2(F)$. The set of all discrete probability distributions over S will be denoted by $\mathsf{Dist}(S)$. Given any $x \in S$, the *Dirac measure* on S, denoted δ_x, is the discrete probability measure μ such that $\mu(x) = 1$.

A discrete-time Markov chain (DTMC) is used to model systems which exhibit probabilistic behavior. Formally, a DTMC is a tuple $\mathcal{M} = (Z, z_s, \Delta)$ where Z is a countable set of *states*, z_s is the *initial* state and $\Delta : Z \hookrightarrow \mathsf{Dist}(Z)$ is the (partial) *transition function* which maps Z to a (discrete) probability distribution over Z. Informally, the process modeled by $\mathcal{M}$ evolves as follows. The process starts in the state z_s. After i execution steps, if the process is in the state z, the process moves to state z' at execution step $(i+1)$ with probability $\Delta(z)(z')$. For the rest of the paper, we will assume that for each state z, if $\Delta(z)$ is defined, then the set $\{z' \mid \Delta(z)(z') > 0\}$ is finite. An execution of $\mathcal{M}$ is a finite sequence $z_0 \to z_1 \to z_2 \to \cdots \to z_m$ such that $z_0 = z_s$ and for each $i \geq 0$, $\Delta(z_i)(z_{i+1}) > 0$. The function Δ can be extended to a probability measure on the σ-algebra genereted by the set of all executions of $\mathcal{M}$.

2.2 Partially observable Markov decision processes

POMDPs are used to model processes which exhibit both probabilistic and nondeterministic behavior, where the states of the system are only partially observable. Formally, a POMDP is a tuple $\mathcal{M} = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ where Z is a countable set of *states*, $z_s \in Z$ is the *initial* state, Act is a (countable) set of *actions*, $\Delta : Z \times \mathsf{Act} \hookrightarrow \mathsf{Dist}(Z)$ is a partial function called the *probabilistic transition relation*, $\mathcal{O}$ is a countable set of observations and $\mathsf{obs} : Z \to \mathcal{O}$ is a labeling of states with observations. Furthermore, we assume that for any action α and states z_1 and z_2 such that $\mathsf{obs}(z_1) = \mathsf{obs}(z_2)$, $\Delta(z_1, \alpha)$ is defined iff $\Delta(z_2, \alpha)$ is defined. As a matter of notation, we shall write $z \xrightarrow{\alpha} \mu$ whenever $\Delta(z, \alpha) = \mu$. A POMDP is like a DTMC except that at each state z, there is a choice amongst several possible probabilistic transitions. The choice of which probabilistic transition to *trigger* is resolved by an *attacker*. Informally, the process modeled by $\mathcal{M}$ evolves as follows. The process starts in the state z_s. After i execution steps, if the process is in the state z, then the attacker chooses an action α such that $z \xrightarrow{\alpha} \mu$ and the process moves to state z' at the $(i+1)$-st step with probability $\mu(z')$. The choice of which action to take is determined by the sequence of observations seen by the attacker.

An *execution* ρ of the POMDP $\mathcal{M}$ is a finite sequence $z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} z_2 \cdots \xrightarrow{\alpha_m} z_m$ such that $z_0 = z_s$ and for each $i \geq 0$, $z_i \xrightarrow{\alpha_{i+1}} \mu_{i+1}$ and $\mu_{i+1}(z_{i+1}) > 0$. The set of all executions of $\mathcal{M}$ will be denoted by $\mathsf{Exec}(\mathcal{M})$. If $\rho = z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} z_2 \cdots \xrightarrow{\alpha_m} z_m$ is an execution then we write $\mathsf{last}(\rho) = z_m$ and say the length of ρ, denoted $|\rho|$, is m. The probability of the event ρ in $\mathcal{M}$ is denoted $\mathsf{prob}(\rho, \mathcal{M})$. An execution ρ_1 is said to be a *one-step extension* of the execution $\rho = z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} z_2 \cdots \xrightarrow{\alpha_m} z_m$ if there exists α_{m+1} and z_{m+1} such that $\rho_1 = z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} z_2 \cdots \xrightarrow{\alpha_m} z_m \xrightarrow{\alpha_{m+1}} z_{m+1}$. In this case, we say that ρ_1 extends ρ by (α_{m+1}, z_{m+1}). An execution is called maximal if it has no one-step extension. For an execution $\rho = z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} z_2 \cdots \xrightarrow{\alpha_m} z_m$ we write $\mathsf{tr}(\rho)$ to represent the *trace* of ρ, defined as the sequence $\mathsf{obs}(z_0) \xrightarrow{\alpha_1} \mathsf{obs}(z_1) \xrightarrow{\alpha_2}$

$\mathsf{obs}(z_2)\cdots \xrightarrow{\alpha_m} \mathsf{obs}(z_m)$. The set of all traces is denoted $\mathsf{Trace}(\mathcal{M})$. Informally, a trace models the view of the attacker.

As discussed above, the choice of which transition to take in an execution is resolved by an attacker. An *attacker* $\mathcal{A} : \mathsf{Trace}(\mathcal{M}) \hookrightarrow \mathsf{Act}$ is a partial function that resolves all non-determinism and the resulting behavior can be described by a DTMC $\mathcal{M}^{\mathcal{A}} = (\mathsf{Exec}(\mathcal{M}), z_s, \Delta^{\mathcal{A}})$ where for each $\rho \in \mathsf{Exec}(\mathcal{M})$, $\Delta^{\mathcal{A}}(\rho)$ is the discrete probability distribution on $\mathsf{Exec}(\mathcal{M})$ such that $\Delta^{\mathcal{A}}(\rho)$ is defined if and only if $\Delta(\mathsf{last}(\rho), \mathcal{A}(\mathsf{tr}(\rho)))$ is defined. When the latter holds,

$$\Delta^{\mathcal{A}}(\rho)(\rho_1) = \begin{cases} \Delta(\mathsf{last}(\rho), \alpha)(z) & \text{if } \alpha = \mathcal{A}(\mathsf{tr}(\rho)),\ z = \mathsf{last}(\rho_1), \text{ and} \\ & \rho_1 \text{ extends } \rho \text{ by } (\alpha, z) \\ 0 & \text{otherwise.} \end{cases}$$

POMDPs and state-based safety properties Given a POMDP $\mathcal{M} = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$, a set $\Psi \subseteq Z$ is said to be a *state-based safety property.* An execution $\kappa \in \mathsf{Exec}(\mathcal{M}^{\mathcal{A}})$ is said to satisfy Ψ if for each state $\rho = z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} z_2 \cdots \xrightarrow{\alpha_m} z_m$ in κ is such that $z_j \in \Psi$ for all $0 \leq j \leq m$. We say $\mathcal{M}$ satisfies Ψ with probability $\geq p$ against the attacker $\mathcal{A}$ (written $\mathcal{M}^{\mathcal{A}} \models_p \Psi$) if the measure of the set $\{\kappa \mid \kappa \text{ is a maximal execution of } \mathcal{M}^{\mathcal{A}} \text{ and } \kappa \not\models \Psi\}$ in the DTMC $\mathcal{M}^{\mathcal{A}}$ is $\leq 1-p$. We say that $\mathcal{M}$ satisfies Ψ with probability $\geq p$ (written $\mathcal{M} \models_p \Psi$) if for all adversaries $\mathcal{A}$, $\mathcal{M}^{\mathcal{A}} \models_p \Psi$.

POMDPs and equivalence properties Let $\mathcal{M} = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ be a POMDP. Given an adversary $\mathcal{A}$ for $\mathcal{M}$ and a trace $t \in \mathsf{Trace}(\mathcal{M})$, let $\mathsf{prob}(t, \mathcal{M}^{\mathcal{A}})$ denote the measure of the event $\{\kappa \mid \kappa \text{ is an exeuciton of } \mathcal{M}^{\mathcal{A}} \text{ and } \mathsf{tr}(\kappa) = t\}$. Two POMDPs $\mathcal{M}_1$ and $\mathcal{M}_2$ with the same set of actions and observations are said to be trace equivalent, denoted $\mathcal{M}_1 \approx \mathcal{M}_2$, if for every attacker $\mathcal{A}$ and trace $t \in \mathsf{Trace}(\mathcal{M}_1) \cup \mathsf{Trace}(\mathcal{M}_2)$, $\mathsf{prob}(t, \mathcal{M}_1^{\mathcal{A}}) = \mathsf{prob}(t, \mathcal{M}_2^{\mathcal{A}})$.

2.3 Terms, equational theories and frames

A signature $\mathcal{F}$ contains a finite set of function symbols, each with an associated arity. We assume countably infinite and pairwise disjoint sets of special constant symbols $\mathcal{N}$ and $\mathcal{M}$, where $\mathcal{N}$ and $\mathcal{M}$ represent public and private names, respectively. Variable symbols are the union of two disjoint sets $\mathcal{X}$ and $\mathcal{X}_w$ (where $\mathcal{F} \cap (\mathcal{X} \cup \mathcal{X}_w) = \emptyset$) representing protocol and frame variables, respectively. Terms are built by the application of function symbols to variables and terms in the standard way. Given a signature $\mathcal{F}$ and $\mathcal{Y} \subseteq \mathcal{X} \cup \mathcal{X}_w$, we use $\mathcal{T}(\mathcal{F}, \mathcal{Y})$ to denote the set of terms built over $\mathcal{F}$ and $\mathcal{Y}$. The set of variables occurring in a term u is denoted by $\mathsf{vars}(u)$. A ground term is one that contains no free variables.

A substitution σ is a partial function that maps variables to terms such that the domain of σ is finite. For a substitution σ, $\mathsf{dom}(\sigma)$ will denote the domain and $\mathsf{ran}(\sigma)$ will denote the range. For a substitution σ with $\mathsf{dom}(\sigma) = \{x_1, ..., x_k\}$, we denote σ as $\{x_1 \mapsto \sigma(x_1), ..., x_k \mapsto \sigma(x_k)\}$. A substitution σ is said to be ground if every term in $\mathsf{ran}(\sigma)$ is ground and a substitution with an empty domain shall be denoted as $\emptyset$. Substitutions can be extended to terms in the usual way and we write $t\sigma$ for the term obtained by applying the substitution σ to the term t.

Our process algebra is parameterized by a non-trivial equational theory $(\mathcal{F}, E)$, where E is a set of $\mathcal{F}$-Equations. By an $\mathcal{F}$-Equation, we mean a pair $u = v$ where $u, v \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X})$ are terms that do not contain private names. Two terms u and v are said to be equal with respect to an equational theory $(\mathcal{F}, E)$, denoted $u =_E v$, if $E \vdash u = v$ in the first order theory of equality. We assume that if two terms containing names are equal, they will remain equal when the names are replaced by arbitrary terms. We often identify an equational theory $(\mathcal{F}, E)$ by E when the signature is clear from the context. Processes are executed in an environment that consists of a frame φ and a binding substitution σ. Formally, $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F})$ and $\varphi : \mathcal{X}_w \to \mathcal{T}(\mathcal{F})$.

Two frames φ_1 and φ_2 are said to be statically equivalent if $\mathsf{dom}(\varphi_1) = \mathsf{dom}(\varphi_2)$ and for all $r_1, r_2 \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X}_w)$ we have $r_1\varphi_1 =_E r_2\varphi_1$ iff $r_1\varphi_2 =_E r_2\varphi_2$. Intuitively, two frames are statically equivalent if an attacker cannot distinguish between the information they contain. A term $u \in \mathcal{T}(\mathcal{F})$ is deducible from a frame φ with recipe $r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathsf{dom}(\varphi))$ in equational theory E, denoted $\varphi \vdash_E^r u$, if $r\varphi =_E u$. We often omit r and E and write $\varphi \vdash u$ if they are clear from the context.

An equational theory E_0 is called trivial if $u =_{E_0} v$ for any terms u, v and otherwise it is said to be non-trivial. For the rest of the paper, $\mathcal{F}_b$ and $\mathcal{F}_c$ are signatures with disjoint sets of function symbols and $(\mathcal{F}_b, E_b)$ and $(\mathcal{F}_c, E_c)$ are non-trivial equational theories. The combination of these two theories will be $(\mathcal{F}, E) = (\mathcal{F}_b \cup \mathcal{F}_c, E_b \cup E_c)$.

3 Process syntax and semantics

In this section we introduce our process algebra for modeling security protocols with coin tosses. This process algebra can be seen as an extension of the one from [3]. Similar to [38], it extends the applied π-calculus by the inclusion of a new operator for probabilistic choice. Like [3], the calculus doesn't include else branches and considers a single public channel.

Process Syntax: We assume a countably infinite set of labels $\mathcal{L}$ and an equivalence relation $\sim$ on $\mathcal{L}$ that induces a countably infinite set of equivalence classes. For $l \in \mathcal{L}$, $[l]$ denotes the equivalence class of l. We use $\mathcal{L}_b$ and $\mathcal{L}_c$ to range over subsets of $\mathcal{L}$ such that $\mathcal{L}_b \cap \mathcal{L}_c = \emptyset$ and both $\mathcal{L}_b$ and $\mathcal{L}_c$ are closed under $\sim$. Each equivalence class is assumed to contain a countably infinite set of labels. Operators in our grammar will come with a unique label from $\mathcal{L}$, which together with the relation $\sim$, will be used to mask the information an attacker can obtain about actions of a process. So, when an action with label l is executed, the attacker will only be able to infer $[l]$.

The syntax of processes is introduced in Figure 1. We begin by introducing what we call basic processes, denoted by $B, B_1, B_2, ...B_n$. In the definition of basic processes, $p \in [0, 1]$, $l \in \mathcal{L}$, $x \in \mathcal{X}$ and $c_i \in \{\top, u = v\} \forall i \in \{1, ..., k\}$ where

Basic Processes

$$B \;::=\; 0 \,\big|\, \nu x^l \,\big|\, (x := u)^l \,\big|\, [c_1 \wedge ... \wedge c_k]^l \,\big|\, \mathtt{in}(x)^l \,\big|\, \mathtt{out}(u)^l \,\big|\, (B \cdot B) \,\big|\, (B +_p^l B)$$

Basic Contexts

$$D[\Box] ::= \Box \,\big|\, B \,\big|\, D[\Box] \cdot B \,\big|\, B \cdot D[\Box] \,\big|\, D[\Box] +_p^l D[\Box]$$

Contexts $[a_i \in \{\nu x, (x := u)\}]$

$$C[\Box_1, ..., \Box_m] ::= a_1^{l_1} \cdot ... \cdot a_n^{l_n} \cdot (D_1[\Box_1]|...|D_m[\Box_m])$$

Fig. 1: Process Syntax

$u, v \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X})$. In the case of the assignment rule $(x := u)^l$, we additionally require that $x \notin \mathsf{vars}(u)$. Intuitively, basic processes will be used to represent the actions of a particular protocol participant. The 0 process does nothing. The process νx^l creates a fresh name and binds it to x while $(x := u)^l$ assigns the term u to the variable x. The test process $[c_1 \wedge ... \wedge c_k]^l$ terminates if c_i is $\top$ or c_i is $u = v$ where $u =_E v$ for all $i \in \{1, ..., k\}$ and otherwise, if some c_i is $u = v$ and $u \neq_E v$, the process deadlocks. The process $\mathtt{in}(x)^l$ reads a term u from the public channel and binds it to x and the process $\mathtt{out}(u)^l$ outputs a term on the public channel. The processes $P \cdot^l Q$ sequentially executes P followed by Q whereas the process $P +_p^l Q$ behaves like P with probability p and like Q with probability $1 - p$.

We will assume a countable set of process variables $\mathcal{X}_c$, whose typical elements will be denoted by $\Box, \Box_1, ..., \Box_m$. In Figure 1, basic contexts are obtained by extending basic processes with a single process variable from $\mathcal{X}_c$. Basic contexts will be denoted by $D[\Box]$, $D_1[\Box]$, $D_2[\Box]$, ..., $D_n[\Box]$. $D_1[B_1]$ denotes the process that results from replacing every occurrence of $\Box$ in D_1 by B_1. A context is then a sequential composition of fresh variable creations and variable assignments followed by the parallel composition of a set of basic contexts. The prefix of variable creations and assignments is used to instantiate data common to one or more basic contexts. A process is nothing but a context that does not contain any process variables. We will use $C, C_1, C_2, ..., C_n$ to denote contexts and P, Q or R to denote processes. For a context $C[\Box_1, ..., \Box_m]$ and basic processes $B_1, ..., B_m$, $C[B_1, ..., B_m]$ denotes the process that results from replacing each process variable $\Box_i$ by B_i.

Definition 1. *A context* $C[\Box_1, ..., \Box_m] = a_1 \cdot ... \cdot a_n \cdot (D_1[\Box_1]|...|D_m[\Box_m])$ *is said to be well-formed if every operator has a unique label and for any labels* l_1 *and* l_2 *occurring in* D_i *and* D_j *for* $i, j \in \{1, 2, ..., m\}$, $i \neq j$ *iff* $[l_1] \neq [l_2]$.

For the remainder of this paper, contexts are assumed to be well-formed. A process that results from replacing process variables in a context by basic processes is also assumed to be well-formed. Unless otherwise stated, we will always assume that all of the labels in a basic process come from the same equivalence class.

Convention 1 *For readability, we will omit process labels when they are not relevant in a particular setting. Whenever new actions are added to a process, their labels are assumed to be fresh and not equivalent to any existing labels of that process.*

For a process Q, $\mathsf{fv}(Q)$ and $\mathsf{bv}(Q)$ denote the set of variables that have some free or bound occurrence in Q, respectively. The formal definition is standard and is presented in Appendix A for completeness. Processes containing no free variables are called ground.

Convention 2 *We restrict our attention to processes that do not contain variables with both free and bound occurrences. That is, for a process Q, $\mathsf{fv}(Q) \cap \mathsf{bv}(Q) = \emptyset$. This requirement can by met by α-renaming the bound variables in Q.*

We now give two examples that illustrate the type of protocols that can be modeled and analyzed in our process algebra.

Example 1. A mix-network [19], is a routing protocol used to hide the origin of messages that pass through it. This is achieved by routing messages through a series of proxy servers, called *mixes*, which receive encrypted traffic from multiple senders, shuffle the messages and forward them in random order. More formally, assume there are users U_1 and U_2 that hold messages m_1 and m_2 intended for recipients V_1 and V_2, respectively. In the case that a single mix sever is utilized, each user U_i prepares a message of the form

$$\mathsf{aenc}(\mathsf{aenc}(m_i, n_i, \mathsf{pk}(V_i)), n_i', \mathsf{pk}(M))$$

and sends it to the mix M. After receiving the messages, the mix decrypts each cipher text and outputs $\mathsf{aenc}(m_1, \mathsf{pk}(V_1))$ and $\mathsf{aenc}(m_2, \mathsf{pk}(V_2))$ in random order. To describe mix networks using our process calculus, we begin by modeling the the public key infrastructure using context below.

$$C[\square_0, \square_1, \square_2] = \nu k_0 \cdot \nu k_1 \cdot \nu k_2 \cdot (D_{pk} | D_0[\square_0] | D_1[\square_1] | D_2[\square_2])$$
$$D_{pk} = \mathtt{out}(\mathsf{pk}(k_0)) \cdot \mathtt{out}(\mathsf{pk}(k_1)) \cdot \mathtt{out}(\mathsf{pk}(k_2))$$
$$D_0[\square_0] = (x_0 := k_0) \cdot \square_0$$
$$D_i[\square_i] = (x_i := \mathsf{pk}(k_0)) \cdot (y_i := \mathsf{pk}(k_i)) \cdot \square_i$$

The behavior of each user and the mix can be described by the processes:

$$U_i = \nu m_i \cdot \nu n_i \cdot \nu n_i' \cdot \mathtt{out}(\mathsf{aenc}(\mathsf{aenc}(m_i, n_i, y_i), n_i', x_i))$$
$$M = \mathtt{in}(z_1) \cdot \mathtt{in}(z_2) \cdot (c_1 := \mathsf{adec}(z_1, x_0)) \cdot (c_2 := \mathsf{adec}(z_2, x_0)) \cdot$$
$$(\mathtt{out}(c_1) \cdot \mathtt{out}(c_2) +_{\frac{1}{2}} \mathtt{out}(c_2) \cdot \mathtt{out}(c_1))$$

The entire system is $C[M, U_1, U_2]$. Provided the original messages appear in random order and are not known to the attacker, the mix is designed to ensure that its outputs cannot be linked to their original creator.

Example 2. In a simple DC-net protocol, two parties Alice and Bob want to anonymously publish two confidential bits m_A and m_B, respectively. To achieve this, Alice and Bob agree on three private random bits b_0, b_1 and b_2 and output a pair of messages according to the following scheme. In our specification of the protocol, all of the private bits will be generated by Alice.

$$\begin{array}{ll} \text{If } b_0 = 0 & \text{Alice: } M_{A,0} = b_1 \oplus m_A,\ M_{A,1} = b_2 \\ & \text{Bob: } M_{B,0} = b_1,\ M_{B,1} = b_2 \oplus m_B \\ \text{If } b_0 = 1 & \text{Alice: } M_{A,0} = b_1,\ M_{A,1} = b_2 \oplus m_A \\ & \text{Bob: } M_{B,0} = b_1 \oplus m_B,\ M_{B,1} = b_2 \end{array}$$

From the protocol output, the messages m_A and m_B can be retrieved as $M_{A,0} \oplus M_{B,0}$ and $M_{A,1} \oplus M_{B,1}$. The party to which the messages belong, however, remains unconditionally private, provided the exchanged secrets are not revealed. This protocol can be modeled using the equational theory built on the signature

$$\mathcal{F}_{DC} = \{0, 1, \oplus, \mathsf{senc}, \mathsf{sdec}, \mathsf{pair}, \mathsf{fst}, \mathsf{snd}, \mathsf{val}_A, \mathsf{val}_B\}$$

with the following equations.

$$\begin{array}{l} \mathsf{sdec}(\mathsf{senc}(m,k),k) = m,\ \mathsf{fst}(\mathsf{pair}(x,y)) = x,\ \mathsf{snd}(\mathsf{pair}(x,y)) = y, \\ (x \oplus y) \oplus z = x \oplus (y \oplus z),\ x \oplus 0 = x,\ x \oplus x = 0,\ x \oplus y = y \oplus x, \\ \mathsf{val}(m, 0, b_1, b_2, r) = \mathsf{pair}(m \oplus b_1, b_2),\ \mathsf{val}(m, 1, b_1, b_2, r) = \mathsf{pair}(b_1, m \oplus b_2) \end{array}$$

The roles of Alice and Bob in this protocol are defined in our process syntax as follows.

$$\begin{array}{l} A = ((b_0 := 0) +_{\frac{1}{2}} (b_0 := 1)) \cdot ((b_1 := 0) +_{\frac{1}{2}} (b_1 := 1)) \cdot ((b_2 := 0) +_{\frac{1}{2}} (b_2 := 1) \cdot \\ \qquad \mathtt{out}(\mathsf{senc}(\mathsf{pair}(b_0, \mathsf{pair}(b_1, b_2)), k_1) \cdot \nu r \cdot \mathtt{out}(\mathsf{val}(m_A, b_0, b_1, b_2, r)) \\ B = \mathtt{in}(z) \cdot (y := \mathsf{sdec}(z, k_2)) \cdot (b_0 := \mathsf{fst}(y)) \cdot (b_1 := \mathsf{fst}(\mathsf{snd}(y))) \cdot \\ \qquad (b_2 := \mathsf{snd}(\mathsf{snd}(y)))\nu r \cdot \mathtt{out}(\mathsf{val}(m_B, b_0 \oplus 1, b_1, b_2, r)) \end{array}$$

Notice that the output of Bob depends on the value of Alice's coin flip. Because our process calculus does not contain else branches, the required functionality is embedded in the equational theory. Also notice that the communication between Alice and Bob in the above specification requires a pre-established secret keys k_1, k_2. These keys are established by first running some key exchange protocol, which can be modeled by the context $C[\Box_1, \Box_2] = \nu k \cdot (k_1 := k) \cdot (k_2 := k) \cdot (\Box_1 | \Box_2)$. If Alice holds the bit b and Bob holds the bit b', the entire protocol is $C[(m_A := b) \cdot A, (m_B := b') \cdot B]$.

Process Semantics: Given a process P, an extended process is a 3-tuple (P, φ, σ) where φ is a frame and σ is a binding substitution. Semantically, a ground process P is a POMDP $[\![P]\!] = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$, where Z is the set of all extended processes, z_s is $(P, \emptyset, \emptyset)$, $\mathsf{Act} = (\mathcal{T}(\mathcal{F} \backslash \mathcal{N}, \mathcal{X}_w) \cup \{\tau\}, \mathcal{L}/\sim)$ and Δ is a partial function that maps an extended process and an action to a distribution. We now give some additional notation needed for the definition of Δ, $\mathcal{O}$ and obs. By $\mu \cdot Q$, we mean the distribution μ_1 such that $\mu_1(P', \varphi, \sigma) = \mu(P, \varphi, \sigma)$ if P' is $P \cdot Q$ and 0 otherwise. The distributions $\mu | Q$ and $Q | \mu$ are defined analogously. The definition of Δ is given in Figure 2, where we write $(P, \varphi, \sigma) \xrightarrow{\alpha} \mu$ if $\Delta((P, \varphi, \sigma), \alpha) = \mu$. The notation $c_i \vdash \top$ is used to denote the case when c_i is $\top$ or c_i is $u = v$ where $\mathsf{vars}(u, v) \subseteq \mathsf{dom}(\sigma)$ and $u\sigma =_E v\sigma$. Note that Δ is well-defined, as basic processes are deterministic and each basic process is associated with a unique equivalence class. Given an extended process η, let $\mathsf{enabled}(\eta)$ denote the set of all $(\S, [l])$ such that for some μ, where $(P, \varphi, \sigma) \xrightarrow{(\S, [l])} \mu$,

$\S \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X}_w) \cup \{\tau\}$ and l is the label of an input or output action. Let ξ be a function from frames to frames such that if $\varphi_0 =_E \varphi_1$ then $\xi(\varphi_0) = \xi(\varphi_1)$ and $\xi(\varphi_0) =_E \varphi_0$. For a frame φ, we write $[\varphi]$ to denote the equivalence class of φ where EQ denotes the set of all such equivalence classes. For $\mathcal{O} = 2^{\mathsf{Act}} \times \mathsf{EQ}$, define obs as a function from extended processes to $\mathcal{O}$ such that for any extended process $\eta = (P, \varphi, \sigma)$, $\mathsf{obs}(\eta) = (\mathsf{enabled}(\eta), \xi(\varphi))$.

Definition 2. *An extended process (Q, φ, σ) preserves the secrecy of $x \in \mathsf{vars}(Q)$ in the equational theory $(\mathcal{F}, E)$, denoted $(Q, \varphi, \sigma) \models_E x$, if there is no $r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathsf{dom}(\varphi))$ such that $\varphi \vdash_E^r x\sigma$. We write $\mathsf{secret}(x)$, for $x \in \mathsf{vars}(Q)$, to represent the set of states of $[\![Q]\!]$ that preserve the secrecy of x. We also write $\mathsf{secret}(\{x_1, ..., x_n\})$ to denote $\mathsf{secret}(x_1) \cap ... \cap \mathsf{secret}(x_n)$. The braces $\{,\}$ will often be omitted for ease of notation.*

$$\text{IN}\quad \frac{r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X}_w) \quad \varphi \vdash^r u \quad x \notin \mathsf{dom}(\sigma)}{(\mathtt{in}(x)^l, \varphi, \sigma) \xrightarrow{(r,[l])} \delta_{(0,\varphi,\sigma \cup \{x \mapsto u\})}} \qquad \text{NEW}\quad \frac{x \notin \mathsf{dom}(\sigma) \quad n \text{ is a fresh name}}{(\nu x^l, \varphi, \sigma) \xrightarrow{(\tau,[l])} \delta_{(0,\varphi,\sigma \cup \{x \mapsto n\})}}$$

$$\text{OUT}\quad \frac{\mathsf{vars}(u) \subseteq \mathsf{dom}(\sigma) \quad i = |\mathsf{dom}(\varphi)| + 1}{(\mathsf{out}(u)^l, \varphi, \sigma) \xrightarrow{(\tau,[l])} \delta_{(0,\varphi \cup \{w_{(i,[l])} \mapsto u\sigma\},\sigma)}} \qquad \text{SEQ}\quad \frac{Q_0 \neq 0 \quad (Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(Q_0 \cdot Q_1, \varphi, \sigma) \xrightarrow{\alpha} \mu \cdot Q_1}$$

$$\text{TEST}\quad \frac{\forall i \in \{1, ..., n\}, c_i \vdash \top}{([c_1 \wedge ... \wedge c_n]^l, \varphi, \sigma) \xrightarrow{(\tau,[l])} \delta_{(0,\varphi,\sigma)}} \qquad \text{NULL}\quad \frac{(Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(0 \cdot Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}$$

$$\text{ASSG}\quad \frac{\mathsf{vars}(u) \subseteq \mathsf{dom}(\sigma) \quad x \notin \mathsf{dom}(\sigma)}{((x := u)^l, \varphi, \sigma) \xrightarrow{(\tau,[l])} \delta_{(0,\varphi,\sigma \cup \{x \mapsto u\sigma\})}} \qquad \text{PAR}_L\quad \frac{(Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(Q_0|Q_1, \varphi, \sigma) \xrightarrow{\alpha} \mu|Q_1}$$

$$\text{PROB}\quad \frac{}{(Q_1 +_p^l Q_2, \varphi, \sigma) \xrightarrow{(\tau,[l])} \delta_{(Q_1,\varphi,\sigma)} +_p \delta_{(Q_2,\varphi,\sigma)}} \qquad \text{PAR}_R\quad \frac{((Q_1, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(Q_0|Q_1, \varphi, \sigma) \xrightarrow{\alpha} Q_0|\mu}$$

Fig. 2: Process semantics

Notation 1 *Note that for process P and variables $x_1, ..., x_n \in \mathsf{vars}(P)$, $\mathsf{secret}(\{x_1, ..., x_n\})$ is a safety property of $[\![P]\!]$. We shall write $P \models_{E,p} \mathsf{secret}(\{x_1, ..., x_n\})$ whenever $[\![P]\!] \models_p \mathsf{secret}(\{x_1, ..., x_n\})$.*

We conclude this section by showing how the notion of trace equivalence can capture privacy properties of the DC-net and mixnet protocols described earlier in this section.

Example 3. Consider the mix network described in Example 1, called a *threshold mix*. Unfortunately, this kind of mix does not ensure sender anonymity when one considers an active attacker. It is vulnerable to what is known as an $n-1$ (flooding) attack, where the attacker simply forwards some users messages along with $n-1$ dummy messages constructed by the attacker to the mix. When the mix performs its output (flushes), the attacker can distinguish which of the n output messages was not among the $n-1$ dummy messages. The flaw in this protocol can be captured by the trace relationship $C[M, U_1, U_2] \not\approx C[M', U_1, U_2]$ where $M' = \mathtt{in}(z_1) \cdot \mathtt{in}(z_2) \cdot \nu c_1 \cdot \nu c_2 \cdot \mathsf{perm}[\mathtt{out}(c_1), \mathtt{out}(c_2)]$. Indeed, consider the attacker for $R = C[M, U_1, U_2]$ (resp. $R' = C[M', U_1, U_2]$) that generates the traces

$$\mathsf{obs}((R, \emptyset, \emptyset)) \xrightarrow{(\tau,[l_1])} \ldots \xrightarrow{(\tau,[l_k])} \mathsf{obs}((M, \varphi, \sigma)))$$
$$\mathsf{obs}((R', \emptyset, \emptyset)) \xrightarrow{(\tau,[l_1])} \ldots \xrightarrow{(\tau,[l_k])} \mathsf{obs}((M', \varphi, \sigma)))$$

where φ is defined below.

$$\begin{aligned}
&w_{4,[l_4]} \mapsto \mathsf{aenc}(\mathsf{senc}(m_2, n_1, \mathsf{pk}(k_2)), n_1', \mathsf{pk}(k_0)),\\
&w_{3,[l_3]} \mapsto \mathsf{aenc}(\mathsf{senc}(m_1, n_2, \mathsf{pk}(k_1)), n_2', \mathsf{pk}(k_0)),\\
&w_{2,[l_2]} \mapsto \mathsf{pk}(k_2), w_{1,[l_1]} \mapsto \mathsf{pk}(k_1), w_{0,[l_0]} \mapsto \mathsf{pk}(k_0)
\end{aligned}$$

and then forwards the recipes $r_1 = (\mathsf{aenc}(\mathsf{aenc}(0, n_1'', w_{1,[l_1]}), n_2'', w_{0,[l_1]})$ and $r_2 = w_{3,[l_3]}$. Every permutation of the mix's output in M corresponds to a different trace as the attacker constructable term $\mathsf{aenc}(0, n_1'', \mathsf{pk}(k_1))$ is output in a different position. In $C[M', U_1, U_2]$ however, there is a single trace, as none of the output messages can be distinguished. If the attacker instead chose $r_1 = w_{4,[l_4]}$ then there would be a single trace t such that $\mathsf{prob}(t, C[M, U_1, U_2]) = \mathsf{prob}(t, C[M', U_1, U_2]) = 1$.

Example 4. Consider the DC-net protocol defined in Example 2 which is designed to insure that an observer of the protocol's output can obtain Alice and Bob's bits but cannot distinguish the party to which each bit belongs. This property can be modeled by the equivalence

$$C[(m_A := 0) \cdot A | (m_B := 1) \cdot B] \approx C[(m_A := 1) \cdot A | (m_B := 0) \cdot B]$$

which says that any attacker for the DC-net protocol will observe identical probabilities for every sequence of protocol outputs, regardless of the bits that Alice and Bob hold in their messages.

4 Compositional equivalence of single session protocols

4.1 Disjoint Data

In the case of non-randomized protocols, it is well known that composition preserves equivalence when protocols do not share data. Recall that we have introduced a new notion of equivalence for randomized protocols wherein two protocols P, Q are equivalent if, for every attacker $\mathcal{A}$ and trace t, the event t has equal probability in the Markov chains $P^{\mathcal{A}}$ and $Q^{\mathcal{A}}$. A cornerstone of our result establishes that parallel composition is a congruence with respect to this equivalence when protocols do not share data.

Lemma 1. *Let P, P', Q, Q' be closed processes such that $\mathsf{vars}(P) \cap \mathsf{vars}(Q) = \emptyset$ and $\mathsf{vars}(P') \cap \mathsf{vars}(Q') = \emptyset$. If $P \approx P'$ and $Q \approx Q'$ then $P|Q \approx P'|Q'$.*

In the absence of probabilistic choice, Lemma 1 is obtained by transforming an attacker $\mathcal{A}$ for $P|Q$ into an attacker $\mathcal{A}'$ that "simulates" Q to P (and vice versa). When a term created by Q is forwarded to P by $\mathcal{A}$, the attacker $\mathcal{A}'$ forwards a new recipe in which the nonces in the term created by Q are replaced by fresh nonces created by the adversary. This technique is not directly applicable in the presence of randomness, where the adversary must forward a common term to P in every indistinguishable execution of $P|Q$. For example, if the outputs n_1 and $\mathsf{h}(n_2)$ from Q are forwarded by $\mathcal{A}$ to P in two indistinguishable executions, the recipe constructed by $\mathcal{A}'$ must be identical for both executions. Further complicating matters, if n_2 is latter revealed by Q, the simulation of n_1 and $\mathsf{h}(n_2)$ to P via a common term can introduce in-equivalences that were not present in the original executions. We prove the result by showing $P \approx P' \Rightarrow P|Q \approx P'|Q$ and $Q \approx Q' \Rightarrow P|Q \approx P|Q'$, which together imply Lemma 1. Each sub-goal is obtained by first inducting on the number of probabilistic choices in the common process. In the case of $P|Q \approx P'|Q$, for example, this allows one to formulate an adversary $\mathcal{A}$ for $P|Q$ (resp. $P'|Q$) as a combination of two disjoint adversaries. We then appeal to a result on POMDPs where we prove that the asynchronous product of POMDPs preserves equivalence.

4.2 Disjoint Primitives

In our composition result, we would like to argue that if two contexts $C[\square]$ and $C'[\square]$ are equivalent and two basic process B and B' are equivalent, then the processes $C[B]$ and $C'[B']$ are trace equivalent. In such a setup, contexts can instantiate data (keys) that are used by and occur free in the basic processes. This setup provides a natural way to model and reason about protocols that begin by carrying out a key exchange before transitioning into another phase of the protocol. Its worth pointing out that the combination of key exchange with anonymity protocols can indeed lead to errors. For example, it was observed in [48] that the RSA implementation of mix networks leads to a degradation in the level of anonymity provided by the mix. The formalization of this result is as follows.

Theorem 1. *Let* $C[\square_1, ..., \square_n] = \nu k_1 \cdot ... \cdot \nu k_m \cdot (D_1[\square_1]|...|D_n[\square_n])$ *(resp.* $C'[\square_1, ..., \square_n] = \nu k'_1 \cdot ... \cdot \nu k'_m \cdot (D'_1[\square_1]|...|D'_n[\square_n])$*) be a context over* $\mathcal{F}_c$ *with labels from* $\mathcal{L}_c$*. Further let* $B_1, ..., B_n$ *(resp.* $B'_1, ..., B'_n$*) be basic processes over* $\mathcal{F}_b$ *with labels from* $\mathcal{L}_b$*. For* $l_1, ..., l_n \in \mathcal{L}_b$ *and* $\sharp \notin \mathcal{F}_b \cup \mathcal{F}_c$*, assume that the following hold.*

1. $\mathsf{fv}(C) = \mathsf{fv}(C') = \emptyset$*,* $\mathsf{fv}(B_i) = \{x_i\}$ *and* $\mathsf{fv}(B'_i) = \{x'_i\}$
2. $\mathsf{vars}(C) \cap \mathsf{vars}(B_i) = \{x_i\}$ *and* $\mathsf{vars}(C') \cap \mathsf{vars}(B'_i) = \{x'_i\}$
3. $C[B_1, ..., B_n]$ *and* $C'[B'_1, ..., B'_n]$ *are ground*
4. $C[B_1, ..., B_n] \models_{E,1} \mathsf{secret}(x_1, ..., x_n)$ *and* $C'[B'_1, ..., B'_n] \models_{E,1} \mathsf{secret}(x'_1, ..., x'_n)$
5. $C[\mathtt{out}(\sharp(x_1))^{l_1}, ..., \mathtt{out}(\sharp(x_n))^{l_n}] \approx C'[\mathtt{out}(\sharp(x_1))^{l_1}, ..., \mathtt{out}(\sharp(x_n))^{l_n}]$
6. $\nu k \cdot (x_1 := k) \cdot ... \cdot (x_n := k) \cdot (B_1|...|B_n) \approx \nu k \cdot (x'_1 := k) \cdot ... \cdot (x'_n := k) \cdot (B'_1|...|B'_n)$

Then $C[B_1, ..., B_n] \approx C'[B'_1, ..., B'_n]$*.*

Proof Sketch. The result is achieved by showing that if $C[B_1, ..., B_n]$ is not trace equivalent to $C'[B'_1, ..., B'_n]$ then one of conditions 5 or 6 from Theorem 1 is violated. More specifically, we use an offending trace t under an attacker $\mathcal{A}$ for $C[B_1, ..., B_n] \not\approx C'[B'_1, ..., B'_n]$, i.e. a trace such that the measure of t in $[\![C[B_1, ..., B_n]]\!]^{\mathcal{A}}$ is different from the measure of t in $[\![C'[B'_1, ..., B'_n]]\!]^{\mathcal{A}}$, to construct a trace t' that witnesses a violation of condition 5 or 6 from Theorem 1. We can show that if $C[B_1, ..., B_n] \not\approx C'[B'_1, ..., B'_n]$ then

$$\begin{gathered} C[\mathtt{out}(\sharp(x_1)), ..., \mathtt{out}(\sharp(x_n))] | B_0 \cdot (B_1 | ... | B_n) \\ \not\approx \\ C'[\mathtt{out}(\sharp(x_1)), ..., \mathtt{out}(\sharp(x_n))] | B'_0 \cdot (B'_1 | ... | B'_n) \end{gathered} \tag{1}$$

where B_0 and B'_0 are processes that bind $\{x_1, ..., x_n\}$ and $\{x'_1, ..., x'_n\}$, respectively. This transformation is a non-trival extension of a result from [27, 3] which allows a process $P|Q$, where P and Q share common variables but are over disjoint signatures, to be transformed into an "equivalent" process $P'|Q'$ where variables are no longer shared. Variables of Q are re-initialized in Q' according to the equational equivalences they respect in an execution of $P|Q$. Unlike nondeterministic processes, where executions are sequences, executions in randomized processes form a tree where variables can receive different values in different branches of the tree. From equation 1, we can apply Lemma 1 to achieve either $C[\mathtt{out}(\sharp(x_1)), ..., \mathtt{out}(\sharp(x_n))] \not\approx C'[\mathtt{out}(\sharp(x_1)), ..., \mathtt{out}(\sharp(x_n))]$ or $B_0 \cdot (B_1 | ... | B_n) \not\approx B'_0 \cdot (B'_1 | ... | B'_n)$. In the former case, we have contradicted condition 5 of Theorem 1. If we achieve $B_0 \cdot (B_1 | ... | B_n) \not\approx B'_0 \cdot (B'_1 | ... | B'_n)$, we additionally need to transform an adversary that witnesses the in-equivalence to an adversary that witnesses the in-equivalence $\nu k \cdot (x_1 := k) \cdot ... \cdot (x_n := k) \cdot (B_1 | ... | B_n) \not\approx \nu k \cdot (x'_1 := k) \cdot ... \cdot (x'_n := k) \cdot (B'_1 | ... | B'_n)$. The presence of randomness makes this transformation tricky, as illustrated by Example 5 below.

Example 5. Define $B_1 = \mathtt{out}(\mathsf{h}(x_1))$, $B_2 = \mathtt{in}(y) \cdot (\mathtt{out}(y) +_{\frac{1}{2}} \mathtt{out}(\mathsf{h}(x_1)))$ and $B'_2 = \mathtt{in}(y) \cdot (\mathtt{out}(\mathsf{h}(x_1)) +_{\frac{1}{2}} \mathtt{out}(\mathsf{h}(x_1)))$ in the processes below.

- $P, P' = \nu k_1 \cdot (x_1 := k_1) \cdot B_1$
- $Q = \nu k_2 \cdot (x_1 := k_2) \cdot B_2$
- $Q' = \nu k_2 \cdot (x_1 := k_2) \cdot B'_2$

Consider the adversary $\mathcal{A}$ that forwards the output of P (resp. P') to Q (resp. Q') and then runs B_2 (resp. B'_2). $\mathcal{A}$ is a witness to the in-equivalence of $P|Q$ and $P'|Q'$, but it does not witness the inequivalence of $\nu k_1 \cdot (x_1 := k_1) \cdot (B_1 | B_2)$ and $\nu k_1 \cdot (x_1 := k_1) \cdot (B_1 | B'_2)$. We can, however, transform the attacker $\mathcal{A}$ to an attacker $\mathcal{A}'$ such that witnesses

$$\nu k_1 \cdot (x_1 := k_1) \cdot (B_1 | B_2) \not\approx \nu k_1 \cdot (x_1 := k_1) \cdot (B_1 | B'_2).$$

The details of this transformation can be found in Lemma 17.

Below, we demonstrate the application of Theorem 1 to reason about the security of the DC-net protocol from Example 2, where Diffie-Hellman is used for key exchange.

Example 6. Let A, B be the protocols for Alice and Bob from the DC-net protocol given in Example 2. Let $\mathcal{F}_{DH} = \{\mathsf{f}, \mathsf{g}, \mathsf{mac}\}$ be the signature for the equational theory

$$E_{DH} = \{f(g(y), x) = f(g(x), y)\}.$$

This equational theory models the Diffie-Hellman primitives, i.e. $f(x, y) = x^y \text{ mod } p$, $g(y) = \alpha^y \text{ mod } p$ for some group generator α. We use mac for a keyed hash function. Define $C[\Box_1, \Box_2] = \nu k_h \cdot D_1[\Box_1]|D_2[\Box_2]$ to be the context that models a variant of the Diffie-Hellman protocol where D_1 and D_2 are below.

$$D_1 = \nu x \cdot \mathtt{out}(\mathsf{g}(x)) \cdot \mathtt{out}(\mathsf{mac}(\mathsf{g}(x), k_h)) \cdot \mathtt{in}(z) \cdot \\ \mathtt{in}(z') \cdot [z' = \mathsf{mac}(z, k_h)] \cdot (k_1 := f(x, z)) \cdot \Box_1$$
$$D_2 = \nu y \cdot \mathtt{out}(\mathsf{g}(y)) \cdot \mathtt{out}(\mathsf{mac}(\mathsf{g}(y), k_h)) \cdot \mathtt{in}(z) \cdot \\ \mathtt{in}(z') \cdot [z' = \mathsf{mac}(z, k_h)] \cdot (k_2 := f(y, z)) \cdot \Box_2$$

We want to show the equivalence $C[(m_A := 0) \cdot A|(m_B := 1) \cdot B] \approx C[(m_A := 1) \cdot A|(m_B := 0) \cdot B]$. Using the results established in Theorem 1, the verification effort is reduced to verifying the following set of simpler properties, where $K = \nu k \cdot (k_1 := k) \cdot (k_2 := k)$.

1. $K \cdot ((m_a := 0) \cdot A|(m_b := 1) \cdot B) \approx K \cdot ((m_a := 1) \cdot A|(m_b := 0) \cdot B)$
2. $C[B_1, ..., B_n] \models_{E,1} \mathsf{secret}(x_1, ..., x_n)$ and $C'[B_1', ..., B_n'] \models_{E,1} \mathsf{secret}(x_1', ..., x_n')$

The latter properties can also be verified modularly using the results from [9].

4.3 Difficulties arising from randomization

In the setup from Theorem 1, observe that $C[\Box], C'[\Box]$ contain process (free) variables. As a result, trace equivalence cannot directly be used to equate these objects. One natural notion of equivalence between $C[\Box]$ and $C'[\Box]$ is achieved by requiring $C[B_0] \approx C'[B_0]$ under all assignments of $\Box$ to a basic process B_0. While mathematically sufficient for achieving composition, such a definition creates a non-trivial computational overhead. Instead, our result is able to guarantee safe composition when $C[B_0] \approx C'[B_0]$ for a *single* instantiatation of B_0. A natural selection for B_0 is the empty process $[\top]$. We illustrate why such a choice is insufficient in Example 7.

Example 7. Consider the contexts defined below.

$$C[\Box_1, \Box_2] = \nu k_1 \cdot \nu k_2 \cdot ((x_1 := k_1) \cdot \Box_1|(x_2 := k_1) \cdot \Box_2 +_{\frac{1}{2}} (x_2 := k_2) \cdot \Box_2)$$
$$C'[\Box_1, \Box_2] = \nu k_1 \cdot \nu k_2 \cdot ((x_1 := k_1) \cdot \Box_1|(x_2 := k_1) \cdot \Box_2 +_{\frac{1}{2}} (x_2 := k_1) \cdot \Box_2).$$

Notice that C and C' differ in that C assigns x_2 to k_1 and C' assigns x_2 to k_1. For the basic processes $B_1 = \mathtt{out}(\mathsf{h}(x_1))$ and $B_2 = \mathtt{out}(\mathsf{h}(x_2))$. We have $C[[\top], [\top]] \approx C'[[\top], [\top]]$ but $C[B_1, B_2] \not\approx C'[B_1, B_2]$. This is because for any adversary $\mathcal{A}$, $C[[\top], [\top]]^{\mathcal{A}}$ and $C'[[\top], [\top]]^{\mathcal{A}}$ both have the same set of traces. However, there is an adversary $\mathcal{A}'$ such that the only trace of $C'[B_1, B_2]^{\mathcal{A}'}$ outputs $\mathsf{h}(k_1), \mathsf{h}(k_1)$. For $C[B_1, B_2]^{\mathcal{A}'}$ there are two traces, one that outputs $\mathsf{h}(k_1), \mathsf{h}(k_2)$ and another that outputs $\mathsf{h}(k_1), \mathsf{h}(k_1)$.

The problematic behavior arising in Example 7 occurs when basic processes reveal equalities among the shared secrets from the context. Revealing these equalities may, in some cases, allow the attacker to infer the result of a private coin toss. Consequently, our composition theorem must require contexts to remain secure even when such equalities are revealed. As was the case with composition contexts, our result also relies on a notion of equivalence between basic processes B and B' that contain free variables. As was the case with equivalence between contexts, universal quantification over the free variables results in a non-trivial computational overhead. However, we are able to show that when B is not trace equivalent to B' under some instantiation the free variables, then B and B' can also be shown to be trace inequivalent when all of the free variables take the same value. This allows us to prove a stronger result by requiring a weaker condition on the equivalence between B and B'.

In Theorem 1, condition 4 requires that the composed processes do not reveal the shared secrets with probability 1. As in [9], one might also be interested in the quantitative version of this result, where the probability of revealing the shared secrets is below a given threshold. Unfortunately, condition 4 cannot be relaxed, as shown by Example 8 below.

Example 8. Consider the contexts

$$\begin{aligned} C[\square] &= \nu k_1 \cdot (x_1 := k_1) \cdot \square \cdot \mathtt{in}(y) \cdot \nu k_2 \cdot \\ &\quad\; [y = x_1] \cdot \mathtt{out}(ok) \\ C'[\square] &= \nu k_1 \cdot (x_1 := k_1) \cdot \square \cdot \mathtt{in}(y) \cdot \nu k_2 \cdot \\ &\quad\; [y = k_2] \cdot \mathtt{out}(ok) \end{aligned}$$

and the basic process $B = \nu n \cdot \mathtt{out}(x_1) +_{\frac{1}{2}} \nu n \cdot \mathtt{out}(n)$. Observe that $C[B] \models_{E,\frac{1}{2}}$ $\mathsf{secret}(x_1)$ and
$C'[B] \models_{E,\frac{1}{2}} \mathsf{secret}(x_1)$. Furthermore,

$$C[\mathtt{out}(\#(x_1))] \approx C'[\mathtt{out}(\#(x_1))]$$

and $\nu k \cdot (x_1 := k) \cdot B \approx \nu k \cdot (x_1 := k) \cdot B$. However, $C[B] \not\approx C'[B]$.

Another subtle component of Theorem 1 is condition 1, which allows each basic process to share only a single variable with the context. As demonstrated Example 9, the composition theorem does not hold when this restriction is relaxed.

Example 9. Consider the context and processes below.

$$\begin{aligned} C[\square] &= \nu k_1 \cdot \nu k_2 \cdot \nu k_3 \cdot (x_1 := k_1) \cdot \\ &\quad\; (x_2 := k_2) \cdot (x_3 := k_3) \cdot \square \\ B_1 &= \mathtt{out}(\mathsf{senc}(x_1, x_3)) \cdot \mathtt{out}(\mathsf{senc}(x_1, x_3)) \\ B_2 &= \mathtt{out}(\mathsf{senc}(x_1, x_3)) \cdot \mathtt{out}(\mathsf{senc}(x_1, x_2)) \end{aligned}$$

For $B_0 = \nu k \cdot (x_1 := k) \cdot (x_2 := k) \cdot (x_3 := k)$ we have $B_0 \cdot B_1 \approx B_0 \cdot B_2$ but $C[B_1] \not\approx C[B_2]$.

4.4 Shared primitives through tagging

Theorem 1 requires that the context and basic processes don't share cryptographic primatives. To extend the result to processes that allow components of the composition to share primitives, such as functions for encryption, decryption and hashing, we utilize a syntactic transformation of a protocol and its signature called tagging. When a protocol is tagged, a special identifier is appended to each of the messages that it outputs. On input, the protocol recursively tests all subterms of the input message to verify their tags are consistent with the protocol's tag. If this requirement is not met, the protocol deadlocks. The details of our tagging scheme, which are simailar to the ones given in [3, 27], can be found in Appendix D. In Theorem 2, we show that an attack on a composition of two tagged protocols originating from the same signature can be mapped to an attack on the composition of the protocols when the signatures are explicitly made disjoint. Given a context $C[\Box_1, ..., \Box_n]$ and basic processes $B_1, ..., B_n$ we write $\lceil C[B_1, ..., B_n] \rceil$ to denote the tagged version of $C[B_1, ..., B_n]$. Our tagging result considers the fixed equational theory where $\mathcal{F}_{\mathsf{senc}} = \{\mathsf{senc}, \mathsf{sdec}, \mathsf{h}\}$ and $E_{\mathsf{senc}} = \{\mathsf{sdec}(\mathsf{senc}(m, k), k) = m\}$. For this theory, we define a signature renaming function $\_^d$ which transforms a context C over the signature $(\mathcal{F}_{\mathsf{senc}}, E_{\mathsf{senc}})$ to a context C^d by replacing every occurrence of the function symbols $\mathsf{senc}, \mathsf{sdec}$ and h in C by $\mathsf{senc}_d, \mathsf{sdec}_d$ and h_d, respectively.

Theorem 2. *Let* $C[\Box_1, ..., \Box_n] = \nu k_1 \cdot ... \cdot \nu k_m \cdot (D_1[\Box_1]|...|D_n[\Box_n])$ *(resp.* $C'[\Box_1, ..., \Box_n] = \nu k'_1 \cdot ... \cdot \nu k'_m \cdot (D'_1[\Box_1]|...|D'_n[\Box_n])$*) be a context over* $\mathcal{F}_{\mathsf{senc}}$ *with labels from* $\mathcal{L}_c$*. Further let* $B_1, ..., B_n$ *(resp.* $B'_1, ..., B'_n$*) be basic processes over* $\mathcal{F}_{\mathsf{senc}}$ *with labels from* $\mathcal{L}_b$*. For* $l_1, ..., l_n \in \mathcal{L}_b$ *and* $\sharp \notin \mathcal{F}_b \cup \mathcal{F}_c$*, assume that the following hold.*

1. $\mathsf{fv}(C) = \mathsf{fv}(C') = \emptyset$, $\mathsf{fv}(B_i) = \{x_i\}$ *and* $\mathsf{fv}(B'_i) = \{x'_i\}$
2. $\mathsf{vars}(C) \cap \mathsf{vars}(B_i) = \{x_i\}$ *and* $\mathsf{vars}(C') \cap \mathsf{vars}(B'_i) = \{x'_i\}$
3. $C[B_1, ..., B_n]$ *and* $C'[B'_1, ..., B'_n]$ *are ground*
4. $C[B_1, ..., B_n] \models_{E,1} \mathsf{secret}(x_1, ..., x_n)$ *and* $C'[B'_1, ..., B'_n] \models_{E,1} \mathsf{secret}(x'_1, ..., x'_n)$
5. $C[\mathtt{out}(\sharp(x_1))^{l_1}, ..., \mathtt{out}(\sharp(x_n))^{l_n}] \approx C'[\mathtt{out}(\sharp(x_1))^{l_1}, ..., \mathtt{out}(\sharp(x_n))^{l_n}]$
6. $\nu k \cdot (x_1 := k) \cdot ... \cdot (x_n := k) \cdot (B_1|...|B_n) \approx \nu k \cdot (x'_1 := k) \cdot ... \cdot (x'_n := k) \cdot (B'_1|...|B'_n)$

Then $\lceil C^c[B^b_1, ..., B^b_n] \rceil \approx \lceil (C')^c[(B'_1)^b, ..., (B'_n)^b] \rceil$.

5 Compositional equivalence for multi-session protocols

In this section, we extend our composition result to protocols that can run multiple sessions. Our focus will be on protocols that have a single occurrence of the replication operator appearing in the context. This restriction simplifies the statement of the results and proofs. However, it is possible to extend our results to protocols with a more general framework for replication. Formally, a context with replication is over the following grammar.

$$C[\Box_1, ..., \Box_m] ::= a_1^{l_1} \cdot ... \cdot a_n^{l_n} \cdot !^l (D_1[\Box_1]|...|D_m[\Box_m])$$

where $a \in \{\nu x, (x := u)\}$. The semantics of this new replication operator are given in Figure 3, where $i \in \mathbb{N}$ is used to denoted the smallest previously unused index. We will write $P(i)$ to denote that process that results from renaming each occurrence of $x \in \mathsf{vars}(P)$ to x^i for $i \in \mathbb{N}$. When $P(i)$ is relabeled freshly as in Figure 3, the new labels must all belong to the same equivalence class (that contains only those labels).

$$\text{REPL} \qquad \frac{P(i) \text{ is relabeled freshly}}{(!^l P, \varphi, \sigma) \xrightarrow{(\tau,l)} \delta_{(P(i)|!^l P, \varphi, \sigma)}}$$

Fig. 3: Replication semantics

Our semantics imposes an explicit variable renaming with each application of a replication rule. The reason for this is best illustrated through an example. Consider the process $!\mathtt{in}(x) \cdot P$ and the execution

$$(!\mathtt{in}(x) \cdot P, \emptyset, \emptyset) \rightarrow^* (\mathtt{in}(x) \cdot P | !\mathtt{in}(x) \cdot P, \varphi, \{x \mapsto t\} \cup \sigma)$$

where variable renaming does not occur. This execution corresponds to the attacker replicating $!\mathtt{in}(x) \cdot P$, running one instance of $\mathtt{in}(x) \cdot P$ and then replicating $!\mathtt{in}(x) \cdot P$ again. Note that, because x is bound at the end of the above execution, the semantics of the input action cause the process to deadlock at $\mathtt{in}(x)$. In other words, an attacker can only effective run one copy of $!\mathtt{in}(x) \cdot P$ for any process of the form $!\mathtt{in}(x) \cdot P$.

Our composition result must prevent messages from one session of a process from being confused with messages from another sessions. We achieve this by introducing an occurrence of $\nu\lambda$ directly following the replication operator. This freshly generated "session tag" will then be used to augment tags occurring in the composed processes. Recall that for any POMDPs M_1 and M_2, if $M_1 \not\approx M_2$ there exists an adversary $\mathcal{A}$ and trace t such that $\mathsf{prob}(t, [\![M_1]\!]^{\mathcal{A}}) = \mathsf{prob}(t, [\![M_2]\!]^{\mathcal{A}})$. This trace t must have finite length and subsequently M_1, M_2 can only perform a bounded number of replication actions in t. This means one can transform $M_1, M_2, \mathcal{A}, t$ to an adversary $\mathcal{A}'$, trace t' and M_1', M_2' such that $\mathsf{prob}(t', [\![M_1']\!]^{\mathcal{A}'}) = \mathsf{prob}(t', [\![M_2']\!]^{\mathcal{A}'})$ where M_1', M_2' do not contain replication. This is achieved by syntactically unrolling the replication operator $|t|$ times in M_1 (resp. M_2). The result below is a consequence of the preceding observation and Theorem 2.

Theorem 3. *Let $C[\square_1, ..., \square_n] = \nu k_1 \cdot ... \cdot \nu k_m \cdot ! \nu\lambda \cdot (D_1[\square_1] | ... | D_n[\square_n])$ (resp. $C'[\square_1, ..., \square_n] = \nu k_1' \cdot ... \cdot \nu k_m' \cdot ! \nu\lambda \cdot (D_1'[\square_1] | ... | D_n'[\square_n])$) be a context over $\mathcal{F}_{\mathsf{senc}}$ with labels from $\mathcal{L}_c$. Further let $B_1, ..., B_n$ (resp. $B_1', ..., B_n'$) be basic processes over $\mathcal{F}_{\mathsf{senc}}$ with labels from $\mathcal{L}_b$. For $l_1, ..., l_n \in \mathcal{L}_b$ and $\sharp \notin \mathcal{F}_b \cup \mathcal{F}_c$, assume that the following hold.*

1. $\mathsf{fv}(C) = \mathsf{fv}(C') = \emptyset$, $\mathsf{fv}(B_i) = \{x_i\}$ *and* $\mathsf{fv}(B_i') = \{x_i'\}$
2. $\mathsf{vars}(C) \cap \mathsf{vars}(B_i) = \{x_i\}$ *and* $\mathsf{vars}(C') \cap \mathsf{vars}(B_i') = \{x_i'\}$

3. $C[B_1, ..., B_n]$ *and* $C'[B'_1, ..., B'_n]$ *are ground*
4. $\lambda \notin \mathsf{vars}(C[B_1, ..., B_n]) \cup \mathsf{vars}(C'[B'_1, ..., B'_n])$
5. $C[B_1, ..., B_n] \models_{E,1} \mathsf{secret}(x_1, ..., x_n)$ *and* $C'[B'_1, ..., B'_n] \models_{E,1} \mathsf{secret}(x'_1, ..., x'_n)$
6. $C[\mathtt{out}(\sharp(x_1))^{l_1}, ..., \mathtt{out}(\sharp(x_n))^{l_n}] \approx C'[\mathtt{out}(\sharp(x_1))^{l_1}, ..., \mathtt{out}(\sharp(x_n))^{l_n}]$
7. $\nu k{\cdot}(x_1 := k){\cdot}...{\cdot}(x_n := k){\cdot}(B_1|...|B_n) \approx \nu k{\cdot}(x'_1 := k){\cdot}...{\cdot}(x'_n := k){\cdot}(B'_1|...|B'_n)$

Then $\lceil \nu k_1 \cdot ... \cdot \nu k_m{\cdot}!\nu\lambda \cdot (D_1^{(c,\lambda)}[B_1^{(b,\lambda)}] \mid ... \mid D_n^{(c,\lambda)}[B_n^{(b,\lambda)}])\rceil \approx \lceil \nu k'_1 \cdot ... \cdot \nu k'_m{\cdot}!\nu\lambda \cdot ((D'_1)^{(c,\lambda)}[(B'_1)^{(b,\lambda)}] \mid ... \mid (D'_n)^{(c,\lambda)}[(B'_n)^{(b,\lambda)}])\rceil$.

6 Conclusion

We have considered the problem of composition for randomized security protocols, initially analyzing protocols with a bounded number of sessions. Formally, consider trace equivalent protocols P and Q over equational theory E_a, and trace equivalent protocols P' and Q' over equational theory E_b. We showed that the composition of P and P' with Q and Q' preserves trace equivalence, provided E_a and E_b are disjoint. The same result applies to the case when both equational theories coincide and consist of symmetric encryption/decryption and hashes, provided each protocol message is tagged with a unique identifier for the protocol to which it belongs. Finally, we show that the latter result extends to protocols with an unbounded number of sessions, as long as messages from each session of the protocol are tagged with a unique session identifier. For future work, we plan to investigate protocols that allow dis-equality tests amongst messages. We also plan to investigate the composition problem when the equational theories coincide and contain other cryptographic primitives in addition to symmetric encryption/decryption and hashes.

References

1. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. *ACM SIGPLAN Notices*, 36(3):104–115, 2001.
2. Suzana Andova, Cas J. F. Cremers, Kristian Gjøsteen, Sjouke Mauw, Stig Fr. Mjølsnes, and Sasa Radomirovic. A framework for compositional verification of security protocols. *Information and Computation*, 206(2-4):425–459, 2008.
3. Myrto Arapinis, Vincent Cheval, and Stéphanie Delaune. Verifying privacy-type properties in a modular way. In *25th IEEE Computer Security Foundations Symposium (CSF'12)*, pages 95–109, 2012.
4. Myrto Arapinis, Vincent Cheval, and Stéphanie Delaune. Composing security protocols: from confidentiality to privacy. In *Proceedings of the 4th International Conference on Principles of Security and Trust (POST'15)*, volume 9036, pages 324–343, 2015.
5. Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *2010 23rd IEEE Computer Security Foundations Symposium*, pages 107–121, 2010.

6. Myrto Arapinis, Stéphanie Delaune, and Steve Kremer. From one session to many: Dynamic tags for security protocols. In *15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08)*, pages 128–142, 2008.
7. Alessandro Armando, Wihem Arsac, Tigran Avanesov, Michele Barletta, Alberto Calvi, Alessandro Cappai, Roberto Carbone, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, et al. The avantssar platform for the automated validation of trust and security of service-oriented architectures. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 267–282, 2012.
8. Mathieu Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. 12th Conference on Computer and Communications Security (CCS'05)*, pages 16–25, 2005.
9. Matthew S. Bauer, Rohit Chadha, and Mahesh Viswanathan. Composing protocols with randomized actions. In *Proceedings of the 5th International Conference on Principles of Security and Trust - Volume 9635*, pages 189–210, 2016.
10. Michael Ben-Or, Oded Goldreich, Silvio Micali, and Ronald L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.
11. Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS'05)*, pages 331–340, 2005.
12. R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, P. Pereira, and R. Segala. Task-Structured Probabilistic I/O Automata. In *Workshop on Discrete Event Systems*, 2006.
13. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd IEEE Symp. on Foundations of Computer Science (FOCS'01)*, pages 136–145, 2001.
14. Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of mutual authentication and key-exchange protocols (extended abstract). In *Proc. 3rd Theory of Cryptography Conference (TCC'06)*, pages 380–403, 2006.
15. R. Chadha, A. P. Sistla, and M. Viswanathan. Verification of randomized security protocols. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS' 17)*, 2017.
16. R. Chadha, A.P. Sistla, and M. Viswanathan. Model checking concurrent programs with nondeterminism and randomization. In *the International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 364–375, 2010.
17. K. Chatzikokolakis and C. Palamidessi. Making Random Choices Invisible to the Scheduler. *Information and Computation*, 2010, to appear.
18. David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.
19. David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.
20. L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices.* PhD thesis, Radboud University of Nijmegen, 2006.
21. Céline Chevalier, Stéphanie Delaune, and Steve Kremer. Transforming password protocols to compose. In *31st Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 204–216, 2011.
22. Yannick Chevalier and Michaël Rusinowitch. Decidability of equivalence of symbolic derivations. *Journal of Automated Reasoning*, 2010. To appear.

23. Stefan Ciobâca. *Verification and composition of security protocols with applications to electronic voting*. PhD thesis, ENS Cachan, 2012.
24. Véronique Cortier, Jérémie Delaitre, and Stéphanie Delaune. Safely composing security protocols. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, pages 352–363, 2007.
25. Véronique Cortier and Stéphanie Delaune. A method for proving observational equivalence. In *Proc. 22nd IEEE Computer Security Foundations Symposium (CSF'09)*, pages 266–276, 2009.
26. Véronique Cortier and Stéphanie Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, 2009.
27. Ştefan Ciobâcă and Véronique Cortier. Protocol composition for arbitrary primitives. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 322–336, 2010.
28. Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 13(3):423–482, 2005.
29. L. de Alfaro. The Verification of Probabilistic Systems under Memoryless Partial Information Policies is Hard. In *PROBMIV*, 1999.
30. Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.
31. Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Composition of password-based protocols. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF'08)*, pages 239–251, 2008.
32. Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.
33. Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.
34. Santiago Escobar, Catherine Meadows, and José Meseguer. *Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties*, pages 1–50. 2009.
35. Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.
36. F.D. Garcia, P. van Rossum, and A. Sokolova. Probabilistic Anonymity and Admissible Schedulers. *CoRR*, abs/0706.1019, 2007.
37. David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In *Proceedings of the First International Workshop on Information Hiding*, pages 137–150, 1996.
38. Jean Goubault-Larrecq, Catuscia Palamidessi, and Angelo Troina. A probabilistic applied pi–calculus. In *Asian Symposium on Programming Languages and Systems*, pages 175–190, 2007.
39. Carl A. Gunter, Sanjeev Khanna, Kaijun Tan, and Santosh S. Venkatesh. Dos protection for reliably authenticated broadcast. In *NDSS*, 2004.
40. Joshua D. Guttman. Authentication tests and disjoint encryption: A design method for security protocols. *Journal of Computer Security*, 12(3-4):409–433, 2004.
41. Joshua D. Guttman. Cryptographic protocol composition via the authentication tests. In *the Foundations of Software Science and Computational Structures, 12th International Conference (FOSSACS 2009)*, pages 303–317, 2009.
42. Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C. Mitchell. A modular correctness proof of ieee 802.11i and TLS. In *the 12th ACM Conference on Computer and Communications Security, (CCS 2005)*, pages 2–15, 2005.

43. Charles Antony Richard Hoare. *Communicating sequential processes*, volume 178. 1985.

44. Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: Probabilistic symbolic model checker. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 200–204, 2002.

45. Catherine Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE journal on selected areas in communications*, 21(1):44–54, 2003.

46. Catherine Meadows. *Emerging Issues and Trends in Formal Methods in Cryptographic Protocol Analysis: Twelve Years Later*, pages 475–492. 2015.

47. Sebastian Mödersheim and Luca Viganò. Sufficient conditions for vertical composition of security protocols. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, pages 435–446, 2014.

48. Birgit Pfitzmann and Andreas Pfitzmann. How to break the direct rsa-implementation of mixes. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 373–381, 1989.

49. Michael K Reiter and Aviel D Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1(1):66–92, 1998.

50. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *14th Computer Security Foundations Workshop (CSFW'01)*, pages 174–190, 2001.

51. P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia. Prêt à voter: a voter-verifiable voting system. *IEEE Transactions on Information Forensics and Security*, 4(4):662–673, 2009.

52. Steve Schneider and Abraham Sidiropoulos. Csp and anonymity. In *European Symposium on Research in Computer Security*, pages 198–218, 1996.

53. Vitaly Shmatikov. Probabilistic analysis of an anonymity system. *Journal of Computer Security*, 12(3, 4):355–377, 2004.

A Auxiliary definitions

For a basic process B, we define $\mathsf{bv}(B), \mathsf{fv}(B)$ as the set of variables that have a bound or free occurrence in B, respectively. Likewise, $\mathsf{abv}(B)$ denotes the set of variables for which every occurrence in B is bound.

$$\mathsf{bv}(B), \mathsf{abv}(B), \mathsf{fv}(B) = \begin{cases} \mathsf{bv}(B) = \mathsf{abv}(B) = \{x\}, & \text{if } B = \nu x \\ \mathsf{fv}(B) = \emptyset & \\ & \\ \mathsf{bv}(B) = \mathsf{abv}(B) = \{x\}, & \text{if } B = (x := u) \\ \mathsf{fv}(B) = \mathsf{vars}(u) & \\ & \\ \mathsf{bv}(B) = \mathsf{abv}(B) = \emptyset, & \text{if } B = [c_1 \wedge ... \wedge c_k] \\ \mathsf{fv}(B) = \mathsf{vars}(c_1) \cup ... \cup \mathsf{vars}(c_k) & \\ & \\ \mathsf{bv}(B) = \mathsf{abv}(B) = \{x\}, & \text{if } B = \mathtt{in}(x) \\ \mathsf{fv}(B) = \emptyset & \\ & \\ \mathsf{bv}(B) = \mathsf{abv}(B) = \emptyset, & \text{if } B = \mathtt{out}(u) \\ \mathsf{fv}(B) = \mathsf{vars}(u) & \\ & \\ \mathsf{bv}(B_1) \cup \mathsf{bv}(B_2), & \text{if } B = B_1 \cdot B_2 \\ \mathsf{abv}(B_1) \cap \mathsf{abv}(B_2), & \\ \mathsf{fv}(B_1) \cup \mathsf{fv}(B_2) & \\ & \\ \mathsf{bv}(B_1) \cup \mathsf{bv}(B_2), & \text{if } B = B_1 +_p B_2 \\ \mathsf{abv}(B_1) \cup \mathsf{abv}(B_2), & \\ \mathsf{fv}(B_1) \cup (\mathsf{fv}(B_2) \setminus \mathsf{abv}(B_1)) & \end{cases}$$

We can lift the preceding definition to a basic context D by requiring that $\mathsf{bv}(\Box) = \mathsf{abv}(\Box) = \mathsf{fv}(\Box) = \emptyset$ for any process variable $\Box$. Let C be a context of the form $B \cdot (D_1(\Box_1)|...|D_m(\Box_m))$ where $B = a_1 \cdot ... \cdot a_n$. Define

$$\mathsf{bv}(C) = \mathsf{bv}(B) \cup \mathsf{bv}(D_1) \cup ... \cup \mathsf{bv}(D_m)$$

and

$$\mathsf{fv}(C) = \mathsf{fv}(B) \cup ((\mathsf{fv}(D_1) \cup .. \cup \mathsf{fv}(D_m)) \setminus \mathsf{bv}(B)).$$

The definitions of bv and fv extend naturally to processes which can always be written as a context without process variables.

For some examples, it is necessary permute the order in which a set of actions $a \in \{\nu x, (x := u), [c_1 \wedge ... \wedge c_k], \mathtt{in}(x), \mathtt{out}(u)\}$ are executed. We introduce the shorthand $\mathsf{perm}[a_1, ..., a_n]$ for such an operation, which is defined inductively using the basic probabilistic choice operator as follows. If $n = 2$, then $\mathsf{perm}[a_1, a_2] = a_1 +_{\frac{1}{2}} a_2$. Inductively, if $n > 2$, then $\mathsf{perm}[a_1, ..., a_n]$ is

$$P_1 +_{\frac{1}{n}} (P_2 +_{\frac{1}{n-1}} (P_3 +_{\frac{1}{n-2}} ...(P_{n-1} +_{\frac{1}{2}} P_n)...)$$

where $P_i = a_i \cdot \mathsf{perm}[a_1, ..., a_{i-1}, a_{i+1}, ..., a_n]$.

B Proof preliminaries

We begin by recalling some definitions from [27, 23]. Let $(\mathcal{F}_b, E_b)$ and $(\mathcal{F}_c, E_c)$ be disjoint nontrivial equational theories and $E = E_b \cup E_c$. Further let $\mathcal{N}$ and $\mathcal{M}$ be sets of private and public names, respectively. We partition N into $\mathcal{N}_a \uplus \mathcal{N}_b \uplus \mathcal{N}_c$ and $\mathcal{M}$ into $\mathcal{M}_a \uplus \mathcal{M}_b \uplus \mathcal{M}_c$. From these partitions, define the d-domain, for $d \in \{b, c\}$, to be $\mathcal{D}_d = \mathcal{F}_d \cup \mathcal{N}_d \cup \mathcal{M}_d$. Similarly we define the a-domain as $\mathcal{D}_a = \mathcal{N}_a \cup \mathcal{M}_a \cup \mathcal{X} \cup \mathcal{X}_w$. For the rest of this section, $d \in \{b, c\}$ and $\overline{d}$ denotes $\{b, c\} \setminus d$. The set of all pure d-terms is $\mathcal{T}(\mathcal{D}_d, \mathcal{X})$ and the set of all pure a-terms is $\mathcal{T}(\mathcal{N}_a \cup \mathcal{M}_a, \mathcal{X} \cup \mathcal{X}_w)$. A term is pure if it is a pure e-term for some $e \in \{a, b, c\}$. For a term u,

$$\mathsf{root}(u) = \begin{cases} f & \text{if } u = f(u_1, ..., u_n) \\ u & \text{if } u \text{ is an atom} \end{cases}$$

and $\mathsf{domain}(u) = e \in \{a, b, c\}$ iff $\mathsf{root}(u) \in \mathcal{D}_e$. A *term context* is a term with holes, denoted $\_^1, ..., \_^k$. A pure d-term context is $H \in \mathcal{T}(\mathcal{D}_d, \{\_^1, ..., \_^k\})$. If H is non-empty term context such that $H \neq \_^1$, then we define $\mathsf{domain}(H) = \mathsf{domain}(\mathsf{root}(H))$. We will write $H[[u_1, ..., u_k]]$ for a term $H[u_1, ..., u_k]$ if H is a pure non-empty term context and $\mathsf{domain}(H) \neq \mathsf{domain}(u_j)$ for all $j \in \{1, ..., k\}$. The terms $u_1, ..., u_k$ are called *alien* subterms of $H[u_1, ..., u_k]$. Given a set of frame variables $\mathcal{X}_w$, let $\mathcal{X}_w^d = \{w_{i,[l]} \mid w_{i,[l]} \in \mathcal{X}_w \wedge [l] \in \mathcal{L}_d\}$.

Definition 3. *Let $\tilde{s} = s_1, ..., s_\ell$ be a sequence of terms and $\tilde{n} = n_1^b, ..., n_\ell^b, n_1^c, ..., n_\ell^c$ be a sequence of fresh private names such that $n_1^b, ..., n_\ell^b \in \mathcal{N}^b$, $n_1^c, ..., n_\ell^c \in \mathcal{N}^c$ and $n_i^d = n_j^d$ iff $s_i = s_j$ for all $d \in \{b, c\}$ and $i, j \in \{1, ..., \ell\}$. For $e \in \{b, c\}$, define the function $R_{\tilde{s},e}^{\tilde{n}} : \mathcal{T}(\mathcal{D}_b \cup \mathcal{D}_c) \to \mathcal{T}(\mathcal{D}_b \cup \mathcal{D}_c)$ as follows.*

$$R_{\tilde{s},e}^{\tilde{n}}(u) = \begin{cases} n_i^e & \textit{if } \mathsf{root}(u) \notin \mathcal{D}_e \textit{ and } u =_E s_i \textit{ for } i \in \{1, ...\ell\} \\ f(R_{\tilde{s},d}^{\tilde{n}}(u_1), ..., R_{\tilde{s},d}^{\tilde{n}}(u_k)) & \textit{if } t = f(u_1, ..., u_k) \textit{ and } f \in \mathcal{D}_d \\ u & \textit{otherwise (when } u \textit{ is an atom)} \end{cases}$$

Definition 4. *Let u be a term and $\mathsf{col}(u)$ be a function such that*

$$\mathsf{col}(u) = \begin{cases} u & \textit{if } u \textit{ is an atom} \\ v_i & \textit{if } u = f(u_1, ..., u_\ell) \textit{ is collapsable to } v_i \\ f(\mathsf{col}(u_1), ..., \mathsf{col}(u_\ell)) & \textit{otherwise} \end{cases}$$

where a term $f(u_1, ..., u_\ell)$ is collapsable to v_i if $f(\mathsf{col}(u_1), ..., \mathsf{col}(u_l)) = H[[v_1, ..., v_k]]$ and $H[[n_1, ..., n_k]] =_{E_d} n_i$ where $n_i, ..., n_k$ are fresh names such that $n_i = n_j$ iff $v_i =_E v_j$ for all $i, j \in \{1, ..., \ell\}$ and $d \in \mathsf{domain}(H)$.

Let $\mathcal{O} : \mathcal{N} \to \mathbb{N}$ be a 1 to 1 function. In case 2 of Definition 4, if $u = f(u_1, ..., u_\ell)$ is collapsible to to v_i and v_j for $i, j \in \{1, ..., \ell\}$ then we pick v_i such that $\mathcal{O}(v_i) < \mathcal{O}(v_j)$. We can easily extend col to a substitution σ by requiring $\mathsf{col}(\sigma)(x) = \mathsf{col}(\sigma(x))$ for all $x \in \mathsf{dom}(\sigma)$.

Lemma 2 ([23]). *For any term u, $\mathsf{col}(u) =_E u$.*

Lemma 3 ([23]). *Fix $d \in \{b, c\}$. For $f \in \mathcal{F}_d$ and terms $u_1, ..., u_\ell$ we have*

$$\mathsf{col}(f(u_1, ..., u_\ell)) = \mathsf{col}(f(\mathsf{col}(u_1), ..., \mathsf{col}(u_\ell))).$$

Lemma 4 ([23]). *(Fundamental Collapse Lemma) If $u =_E v$, then $\mathsf{col}(u) = H_1[[u_1, ..., u_k]]$ and $\mathsf{col}(v) = H_2[[u_{k+1}, ..., u_{k+\ell}]]$ are such that $\mathsf{domain}(H_1) = \mathsf{domain}(H_2)$ and $H_1[n_1, ..., n_k] =_{E_d} H_2[n_{k+1}, ..., n_{k+\ell}]$ where $d \in \mathsf{domain}(H_1)$ and $n_1, ..., n_{k+\ell}$ fresh names such that $n_i = n_j$ iff $u_i =_E u_j$ for all $1 \leq i, j \leq k + \ell$.*

For the remainder of this section, let $\tilde{s} = s_1, ..., s_\ell$ be a sequence of terms and $\tilde{n} = n^b_1, ..., n^b_\ell, n^c_1, ..., n^c_\ell$ be a sequence of fresh private names such that $n^b_1, ..., n^b_\ell \in \mathcal{N}^b$, $n^c_1, ..., n^c_\ell \in \mathcal{N}^c$ and $n^d_i = n^d_j$ iff $s_i = s_j$ for all $d \in \{b, c\}$ and $i, j \in \{1, ..., \ell\}$.

Lemma 5 ([23]). *Fix $d \in \{b, c\}$. If $u =_E v$, then $R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(u)) =_E R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(v))$.*

Lemma 6 ([23]). *Let φ be a frame and r be a recipe over φ such that $\varphi \not\vdash^{r'} s_i$ for any $i \in \{1, ..., \ell\}$ and for any sub-recipe r' of r. Then for any $d \in \{b, c\}$, $rR^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(\varphi)) =_E R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(r\varphi))$.*

Because E is stable by replacement of names for equivalent terms, we have the following.

Definition 5. *Let u_1, u_2 be terms and $n \in \mathsf{st}(u_1) \cup \mathsf{st}(u_2)$. If $u_1 =_E u_2$ and v_1, v_2 are terms such that $v_1 =_E v_2$ then $u_1\{n \mapsto v_1\} =_E u_2\{n \mapsto v_2\}$.*

Lemma 7. *Let φ and φ' be frames over $\mathcal{T}(\mathcal{D}_b \cup \mathcal{D}_c)$ such that $\mathsf{dom}(\varphi) = \mathsf{dom}(\varphi')$ and $w_{i,[l]}\varphi' = R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(w_{i,[l]}\varphi))$. If $\varphi \not\vdash \tilde{s}$ and $\varphi' \not\vdash \tilde{s}$ then $\varphi \equiv \varphi'$.*

Proof. We need to show that for any two recipes r_1 and r_2, $r_1\varphi =_E r_2\varphi$ iff $r_1\varphi' =_E r_2\varphi'$. Assume $r_1\varphi =_E r_2\varphi$. By Lemma 5, $R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(r_1\varphi)) =_E R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(r_2\varphi))$. Because $\varphi \not\vdash \tilde{s}$, we can apply Lemma 6 to both sides, yielding $r_1R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(\varphi)) =_E r_2R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(\varphi))$. By the definition of φ', we have $r_1\varphi' =_E r_2\varphi'$ as desired. The other direction follows by Definition 5. ☐

For any two executions ρ_0, ρ_1 we say that ρ_0 is a prefix of ρ_1, denoted $\rho_0 \preceq \rho_1$, if $\rho_1 = \rho_0 \xrightarrow{\alpha_0} ... \xrightarrow{\alpha_k} z_k$. The prefix operation $\preceq$ is extended to traces in the natural way. For a process P and an adversary $\mathcal{A}$, we will write $\mathsf{MExec}([\![P]\!]^{\mathcal{A}})$ to denote the subset of $\mathsf{Exec}([\![P]\!]^{\mathcal{A}})$ that contains only maximal executions. A trace t is said to be maximal for $[\![P]\!]^{\mathcal{A}}$ if $t = \mathsf{tr}(\rho)$ where $\rho \in \mathsf{MExec}([\![P]\!]^{\mathcal{A}})$. The set of all attackers will be denoted by A.

Proposition 1. *Let P, Q be processes such that $P \not\approx Q$. There exist an adversary $\mathcal{A}$ and a trace t that is maximal for $[\![P]\!]^{\mathcal{A}}$ and $[\![Q]\!]^{\mathcal{A}}$ such that $\mathsf{prob}(t, [\![P]\!]^{\mathcal{A}}) \neq \mathsf{prob}(t, [\![Q]\!]^{\mathcal{A}})$.*

Definition 6. *Let $\overline{P} = P_1, P_2$ (resp. $\overline{Q} = Q_1, Q_2$) be a sequence of processes and $\mathcal{A}$ be an attacker for P_1, P_2. Further Let $\Lambda : \mathsf{A} \to \mathsf{A}$ be a function and $\delta : \mathsf{MExec}([\![P_1]\!]^{\mathcal{A}}) \uplus \mathsf{MExec}([\![P_2]\!]^{\mathcal{A}}) \to \mathsf{MExec}([\![Q_1]\!]^{\Lambda(\mathcal{A})}) \uplus \mathsf{MExec}([\![Q_2]\!]^{\Lambda(\mathcal{A})})$ be a bijection such that the following hold.*

1. *For any* $\rho \in \mathsf{MExec}([\![P_i]\!]^{\mathcal{A}})$*,* $\mathsf{prob}(\rho, [\![P^{\mathcal{A}}]\!]) = \mathsf{prob}(\delta(\rho), [\![Q_i^{\Lambda(\mathcal{A})}]\!])$
2. *For any* $\rho_0, \rho_1 \in \mathsf{MExec}([\![P_1]\!]^{\mathcal{A}}) \uplus \mathsf{MExec}([\![P_2]\!]^{\mathcal{A}})$*,* $\mathsf{tr}(\rho_0) = \mathsf{tr}(\rho_1)$ *iff* $\mathsf{tr}(\delta(\rho_0)) = \mathsf{tr}(\delta(\rho_1))$*.*

Then $\overline{P}$ *is said to be* transposable *to* $\overline{Q}$ *by* Λ*.*

Lemma 8. *Let* (P_1, P_2) *be transposable to* (Q_1, Q_2)*. If* $P_1 \not\approx P_2$ *then* $Q_1 \not\approx Q_2$*.*

Proof. Let $\mathcal{A}$ be an adversary and t be a trace such that $\mathsf{prob}(t, P_1^{\mathcal{A}}) \neq \mathsf{prob}(t, P_2^{\mathcal{A}})$. Let $\rho_1, ..., \rho_k \in \mathsf{MExec}([\![P_1]\!]^{\mathcal{A}})$ (resp. $\rho_1', ..., \rho_l' \in \mathsf{MExec}([\![P_2]\!]^{\mathcal{A}})$) be the executions such that $\mathsf{tr}(\rho_i) = t$ (resp. $\mathsf{tr}(\rho_j') = t$) for all $i \in \{1, ..., k\}$ (resp. $j \in \{1, ..., l\}$). By property 2 of Definition 6 there exists a trace t' such that for the executions $\delta(\rho_1), ..., \delta(\rho_k) \in \mathsf{MExec}([\![Q_1]\!]^{\Lambda(\mathcal{A})})$ (resp. $\delta(\rho_1'), ..., \delta(\rho_l') \in \mathsf{MExec}([\![Q_2]\!]^{\Lambda(\mathcal{A})})$) $\mathsf{tr}(\delta(\rho_i)) = t'$ (resp. $\mathsf{tr}(\delta(\rho_j')) = t'$) for all i (resp. j). By property 1 of Definition 6, $\mathsf{prob}(t', {Q_1}^{\Lambda(\mathcal{A})}) = \mathsf{prob}(t, P_1^{\mathcal{A}}) \neq \mathsf{prob}(t, P_2^{\mathcal{A}}) = \mathsf{prob}(t', Q_2^{\Lambda(\mathcal{A})})$. □

B.1 Disjoint Variables

A process is called atomic if it can derived from the grammar:

$$A ::= \nu x^l \mid (x := u)^l \mid [c_1 \wedge ... \wedge c_k]^l \mid \mathtt{in}(x)^l \mid \mathtt{out}(u)^l$$

where $c_i \in \{\top, s = t\}$ for all $i \in \{1, ..., k\}$. We will use a to denote atomic processes. A process is called linear if it can be derived from the grammar $L ::= A \mid (L \cdot A)$ where A is an atomic process. Let P be a process and $\mathcal{A}$ be an adversary for P. For any $\rho \in \mathsf{Exec}([\![P]\!]^{\mathcal{A}})$, define the linear process $L(\rho)$ inductively as follows. For the base case $L((P, \emptyset, \emptyset)) = \epsilon$. For the inductive case, let $\rho = \rho_0 \xrightarrow{(\S, [l])} z_n'$. If l is the label of an atomic action a, then $L(\rho) = L(\rho_0) \cdot a$. Otherwise l is the label of a probabilistic choice and $L(\rho) = L(\rho_0) \cdot [T]^l$.

Proposition 2. *Let* P *be a process,* $\mathcal{A}$ *be an adversary for* P *and* $\rho \in \mathsf{Exec}([\![P]\!]^{\mathcal{A}})$*. There exists an adversary* $\mathcal{A}'$ *for* $L(\rho)$*, an execution* $\rho' \in \mathsf{Exec}([\![L(\rho)]\!]^{\mathcal{A}'})$*, a frame* φ *and a substitution* σ *such that* $\mathsf{last}(\rho) = (Q, \varphi, \sigma)$ *and* $\mathsf{last}(\rho') = (Q', \varphi, \sigma)$*.*

A process is called deterministic if it can be derived from the following grammar.

$$D \ ::= \ a \mid D +_p D \mid a \cdot D$$

The concatenation of two deterministic process, denoted $D_1 \circ D_2$, is defined below.

$$D_1 \circ D_2 = \begin{cases} a \cdot D_2 & \text{if } D_1 = a \\ D_1' \cdot D_2 +_p D_1'' \cdot D_2 & D_1 = D_1' +_p D_1'' \\ a \cdot (D_1' \circ D_2) & D_1 = a \cdot D_1' \end{cases}$$

For a basic process B, its expansion to a deterministic process, denoted $\mathsf{exp}(B)$, is defined as follows.

$$\mathsf{exp}(B) = \begin{cases} a & \text{if } B = a \\ a \cdot \mathsf{exp}(B') & \text{if } B = a \cdot B' \\ \mathsf{exp}(B_1) +_p \mathsf{exp}(B_2) & \text{if } B = B_1 +_p B_2 \\ \mathsf{exp}(B_1) \circ \mathsf{exp}(B') +_p & \text{if } B = (B_1 +_p B_2) \cdot B' \\ \mathsf{exp}(B_2) \circ \mathsf{exp}(B') & \end{cases}$$

The set of interleavings for the parallel composition of deterministic processes is given in Figure 4, and will be denoted by $\mathcal{I}(P)$ for a process P.

$$\frac{D \in \mathcal{I}(D_1|...|D_m)}{a \cdot D \in \mathcal{I}(D_1|...|D_{i-1}|a|D_{i+1}|...|D_m)} \qquad \frac{D \in \mathcal{I}(D_1|...|D_i|...|D_m)}{a \cdot D \in \mathcal{I}(D_1|...|a \cdot D_{i-1}|...|D_m)}$$

$$\frac{D' \in \mathcal{I}(D_1|...|D_i'|...|D_m) \quad D'' \in \mathcal{I}(D_1|...|D_i''|...|D_m)}{D' +_p D'' \in \mathcal{I}(D_1|...|D_i' +_p D_i''|...|D_m)}$$

Fig. 4. Interleavings for parallel composition of basic processes

Recall that a process Q takes the form $a_1 \cdot ... \cdot a_n \cdot (B_1|...|B_m)$ where $a_i \in \{\nu x, (x := t)\}$. If $D \in \mathcal{I}(\mathsf{exp}(B_1)|...|\mathsf{exp}(B_m))$, then $a_1 \cdot ... \cdot a_n \cdot D \in \mathcal{I}(P)$. Notice that $\mathcal{I}(Q)$ is a deterministic process. Further, for any adversary $\mathcal{A}$ and process Q there exists a deterministic process $D \in \mathcal{I}(Q)$ such that $\mathsf{Exec}([\![D]\!]^{\mathcal{A}}) = \mathsf{Exec}([\![Q]\!]^{\mathcal{A}})$ modulo a renaming of variables. We will write $\mathcal{I}(Q, \mathcal{A})$ to denote such a deterministic process. For a bitstring ω, if $a \in \{\nu x, \mathtt{in}(x), (x := t)\}$, let a^ω be νx^ω, $in(x^\omega)$ or $(x^\omega := t)$, respectively. For any deterministic process D and bitstring ω, let D^ω be

$$D^\omega = \begin{cases} a & \text{if } D = a \text{ and } a \notin \overline{a} \\ a^\omega & \text{if } D = a \text{ and } a \in \overline{a} \\ a \cdot D_0^\omega & \text{if } D = a \cdot D_0 \text{ and } a \notin \overline{a} \\ a^\omega \cdot D_0^\omega\{x \mapsto x^\omega\} & \text{if } D = a \cdot D_0 \text{ and } a \in \overline{a} \\ D_0^{\omega 0} +_p D_1^{\omega 1} & \text{if } D = D_0 +_p D_1 \end{cases}$$

for $\overline{a} = \{\nu x, \mathtt{in}(x), (x := t)\}$. By construction, D^ω is such that for any adversary $\mathcal{A}$ and $\rho_1, \rho_2 \in \mathsf{Exec}([\![D^\omega]\!]^{\mathcal{A}})$ if $\mathsf{last}(\rho_1) = (D_1, \varphi_1, \sigma_1)$, $\mathsf{last}(\rho_2) = (D_2, \varphi_2, \sigma_2)$ and $x \in \mathsf{dom}(\sigma_1) \cap \mathsf{dom}(\sigma_2)$ then $x\sigma_1 = x\sigma_2$. Let $\rho_1, ..., \rho_m = \mathsf{MExec}([\![D^\omega]\!]^{\mathcal{A}})$ where $\mathsf{last}(\rho_j) = (D_j, \varphi_j, \sigma_j)$ for all $j \in \{1, ..., m\}$. Define $\mathsf{bind}(D^\omega, \mathcal{A}) = \bigcup_{j=1}^{j=m} \sigma_j$. For any process Q, let $Q^b_{\mathcal{L}_b}$ be the result of replacing, in Q, every occurrence of a variable x that occurs in an atomic process a^l, where $l \in \mathcal{L}_b$, by the variable x^b. For the remainder of this section, let $i \in \{1, ..., n\}$, $S = \{x_1, ..., x_n\}$ and

$$C[\square_1, ..., \square_n] = \nu k_1 \cdot ... \cdot \nu k_m \cdot (D_1[\square_1]|D_2[\square_2]|...|D_n[\square_n])$$

$$C'[\Box_1,...,\Box_n] = \nu k'_1 \cdot ... \cdot \nu k'_m \cdot (D'_1[\Box_1]|D'_2[\Box_2]|...|D'_n[\Box_n])$$

be contexts over $\mathcal{F}_c$ with labels from $\mathcal{L}_c$, $B_1,...,B_n$ (resp. $B'_1,...,B'_n$) be basic processes over $\mathcal{F}_b$ with labels from $\mathcal{L}_b$ and assume that the conditions of Theorem 1 hold. Let $\mathcal{A}$ be an adversary for $C[B_1,...,B_n]$ and $C'[B'_1,...,B'_n]$. Define

$$S_D = \{x^\omega \mid x^\omega \in \mathsf{vars}(\mathcal{I}(C[B_1,...,B_n],\mathcal{A})^\epsilon) \wedge x \in \mathsf{vars}(S)\}$$

and

$$S'_D = \{x^\omega \mid x^\omega \in \mathsf{vars}(\mathcal{I}(C'[B'_1,...,B'_n],\mathcal{A})^\epsilon) \wedge x \in \mathsf{vars}(S)\}.$$

Let $M = \mathsf{max}(|S_D|,|S'_D|)$.

Definition 7. *Define $\Delta(C[B_1,....,B_n],\mathcal{A},M)$ to be the process*

$$[T]_1 \cdot ... \cdot [T]_s \cdot \nu k'_1 \cdot ... \cdot \nu k'_v \cdot \\ (y^b_1 := z_1) \cdot ... \cdot (y^b_v := z_v) \cdot \ (\mathcal{I}(C[B_1,...,B_n],\mathcal{A})^\epsilon)^b_{\mathcal{L}_b}$$

such that $\{y_1,...,y_v\} = S_D$, $z_j \in \{k'_1,...,k'_v\}$ and for all z_j, z_h we have $z_j = z_h$ iff $\sigma^{\mathcal{A}}(y_j) =_E \sigma^{\mathcal{A}}(y_h)$ for $\sigma^{\mathcal{A}} = \mathsf{bind}(C[B_1,...,B_n],\mathcal{A})$. We also require that s is chosen such that $s + 2v = M$.

In Definition 7, every atomic action in the prefix $[T]_1 \cdot ... \cdot [T]_s \cdot \nu k'_1 \cdot ... \cdot \nu k'_v \cdot (y^b_1 := z_1) \cdot ... \cdot (y^b_v := z_v)$ is labeled sequentially with $l_1,...,l_{2v+s} \in \mathcal{L}_b$. We can similarly define $\Delta(C'[B'_1,....,B'_n],\mathcal{A},M)$. Let $\mathsf{dis}^{\mathcal{A}} : \mathsf{A} \to \mathsf{A}$ be as follows. If $\mathcal{A}'$ is not an adversary for $\Delta(C[B_1,....,B_n],\mathcal{A},M)$ then $\mathsf{dis}^{\mathcal{A}}(\mathcal{A}')$ is undefined on all traces. Otherwise $\mathsf{dis}^{\mathcal{A}}(\mathcal{A}')$ is such that $\mathcal{A}'(t) = \alpha$ then $\mathsf{dis}^{\mathcal{A}}(\mathcal{A}')(t_0 t) = \alpha$ where $t_0 = o_0 \xrightarrow{(\tau,[l_1])} ... \xrightarrow{(\tau,[l_{2v+s}])} o_{2v+s}$. On all other traces, $\mathsf{dis}^{\mathcal{A}}(\mathcal{A}')$ is undefined. In what follows, when we write ρ reveals S for some $\rho \in \mathsf{Exec}([\![P]\!])$ and $S \subseteq \mathsf{vars}(P)$ we mean that there exists an $x \in S$ such $\varphi \vdash x\sigma$ where $\mathsf{last}(\rho) = (P',\varphi,\sigma)$. We will also assume that $B_1,...,B_n$ and $C[\Box_1,...,\Box_n]$ only share the variables $S = \{x_1,...,x_n\}$. We will write $\mathsf{vars}^b(C[B_1,...,B_n])$ and $\mathsf{vars}^c(C[B_1,...,B_n])$ to denote the sets $\bigcup_{i=1}^n \mathsf{vars}(B_i) \setminus x_i$ and $\mathsf{vars}(C[\Box_1,...,\Box_n]) \setminus S$ respectively.

Lemma 9. *Let $\mathcal{A}$ be an adversary for $P_0 = C[B_1,...,B_n]$ and $P_1 = C'[B'_1,...,B'_n]$. If $Q_0 = \Delta(P_0,\mathcal{A},M)$ and $Q_1 = \Delta(P_1,\mathcal{A},M)$ then P_0, P_1 is transposble to Q_0, Q_1 by the function $\mathsf{dis}^{\mathcal{A}}$.*

Proof. Observe that $\mathsf{Exec}([\![P_0]\!]^{\mathcal{A}}) = \mathsf{Exec}([\![\mathcal{I}(P_0,\mathcal{A})]\!]^{\mathcal{A}})$ and $\mathcal{I}(P_0,\mathcal{A})$ is the same as $\mathcal{I}(P_0,\mathcal{A})^\epsilon$ modulo renaming of variables. Likewise, $\mathsf{Exec}([\![P_1]\!]^{\mathcal{A}}) = \mathsf{Exec}([\![\mathcal{I}(P_1,\mathcal{A})]\!]^{\mathcal{A}})$ and $\mathcal{I}(P_1,\mathcal{A})$ is the same as $\mathcal{I}(P_1,\mathcal{A})^\epsilon$ modulo renaming of variables. Thus, it suffices to show $\mathcal{I}(P_0,\mathcal{A})^\epsilon, \mathcal{I}(P_0,\mathcal{A})^\epsilon$ is transposable to Q_0, Q_1 by $\mathsf{dis}^{\mathcal{A}}$. We define a bijection $\delta : \mathsf{MExec}(\mathcal{I}(P_0,\mathcal{A})^\epsilon) \uplus \mathsf{MExec}(\mathcal{I}(P_1,\mathcal{A})^\epsilon) \to \mathsf{MExec}([\![Q_0]\!]^{\mathsf{dis}^{\mathcal{A}}}) \uplus \mathsf{MExec}([\![Q_1]\!]^{\mathsf{dis}^{\mathcal{A}}})$. The definition of δ is only given on executions of $\mathsf{MExec}([\![\mathcal{I}(P_0,\mathcal{A})^\epsilon]\!]^{\mathcal{A}})$, it can be naturally extended to executions of $\mathsf{MExec}([\![\mathcal{I}(P_1,\mathcal{A})^\epsilon]\!]^{\mathcal{A}})$. Let $\mathsf{dis}^{\mathcal{A}}(\mathcal{A}) = \mathcal{A}'$ and $P'_0 = \mathcal{I}(P_0,\mathcal{A})^\epsilon$. Define a bijection

$$\delta_0 : \mathsf{MExec}([\![P'_0]\!]^{\mathcal{A}}) \to \mathsf{MExec}([\![Q_0]\!]^{\mathcal{A}'})$$

as follows. To begin, let $\rho_0 = (Q_0, \emptyset, \emptyset) \xrightarrow{(\tau,[l])} \ldots \xrightarrow{(\tau,[l])} ((R_0)^b_{\mathcal{L}_b}, \emptyset, \sigma_0)$ for $l \in \mathcal{L}_b$ and $\mathsf{dom}(\sigma) = \{k'_1, ..., k'_v, y^b_1, ..., y^b_v\}$ be an execution of $\mathsf{MExec}(\llbracket Q_0 \rrbracket^{\mathcal{A}'})$ such that $|\rho_0| = M$. Now consider any $\rho \in \mathsf{MExec}(\llbracket P'_0 \rrbracket^{\mathcal{A}})$ where $(R_0, \varphi_0, \sigma_0) \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_m} (R_m, \varphi_m, \sigma_m)$. Define $\rho' = ((R_0)^b_{\mathcal{L}_b}, \varphi'_0, \sigma'_0) \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_m} ((R_m)^b_{\mathcal{L}_b}, \varphi'_m, \sigma'_m)$ where for all $j \in \{1, ..., m\}$, φ'_j and σ'_j are as follows. If $\sigma^{\mathcal{A}} = \mathsf{bind}(P_1, \mathcal{A})$, $\tilde{s} = \{\sigma^{\mathcal{A}}(y_1), ..., \sigma^{\mathcal{A}}(y_v)\}$ and $\tilde{n} = \{\sigma_0(y^b_1), ..., \sigma_0(y^b_v)\}$:

1. $\sigma'_j = \sigma_0 \cup \{z^b \mid z \in \mathsf{dom}(\sigma) \cup \mathsf{vars}^b(P'_0)\} \cup \{z \mid z \in \mathsf{dom}(\sigma) \cup \mathsf{vars}^c(P'_0)\}$
   - $\sigma'_j(z) = R^{\tilde{n}}_{\tilde{s},b}(\mathsf{col}(\sigma_j(z)))$ for all $z \in \mathsf{dom}(\sigma'_j)$
   - $\sigma'_j(z^b) = R^{\tilde{n}}_{\tilde{s},b}(\mathsf{col}(\sigma_j(z)))$ for all $z^b \in \mathsf{dom}(\sigma'_j)$
2. $\mathsf{dom}(\varphi'_j) = \mathsf{dom}(\varphi_j)$
   - $w_{i,[l]}\varphi'_j = R^{\tilde{n}}_{\tilde{s},b}(\mathsf{col}(w_{i,[l]}\varphi_j))$

Define $\delta_0(\rho) = \rho_0\rho'$. Let $S_D = \{x^\omega \mid x^\omega \in \mathsf{vars}(P'_0)$ and $x \in S\}$ and $S'_D = S_D \cup \{(x^\omega)^b \mid (x^\omega)^b \in \mathsf{vars}(Q_0)$ and $x \in S\}$. Consider any $\rho \in \mathsf{MExec}(\llbracket P'_0 \rrbracket^{\mathcal{A}})$. By condition 4 of Theorem 1 $\rho \not\models \mathsf{secret}(S_D)$. Applying Proposition 2, the linear process $L(\rho) \not\models \mathsf{secret}(S_D)$. Furthermore, $L(\rho)$ and $L(\delta_0(\rho))$ meet the conditions of Theorem 1 from [27], allowing one to conclude that $L(\delta_0(\rho)) \not\models \mathsf{secret}(S'_D)$ and $\delta_0(\rho) \in \mathsf{MExec}(\llbracket Q_0 \rrbracket^{\mathcal{A}'})$. Proposition 2 again yields $\delta_0(\rho) \not\models \mathsf{secret}(S'_D)$. In particular, this means that $\rho \not\models \tilde{s}$ and $\delta_0(\rho) \not\models \tilde{s}$ and we can apply Lemma 7 to conclude $\varphi \equiv \varphi'$ and hence $\mathsf{obs}(\mathsf{last}(\rho)) = \mathsf{obs}(\mathsf{last}(\delta_0(\rho)))$. As noted previously, we can similarly define a bijection $\delta_1 : \mathsf{MExec}(\mathcal{I}(P_1, \mathcal{A})^\epsilon) \to \mathsf{MExec}(\llbracket Q_1 \rrbracket^{\mathcal{A}'})$ that satisfies the properties above. Let $\delta = \delta_0 \uplus \delta_1$. We have that $\mathcal{A}'$ is a valid adversary for Q_0, Q_1 and property 2 of Definition 6 holds. Clearly property 1 also holds by the definition of δ. $\square$

B.2 POMDP Result

We first fix some notation. Let $M_i = (Z_i, z^s_i, \mathsf{Act}_i, \Delta_i, \mathcal{O}_i, \mathsf{obs}_i)$ be a POMDP for $i \in \{1, 2\}$. We will assume that $Z_1 \cap Z_2 = \emptyset$ and $\mathsf{Act}_1 \cap \mathsf{Act}_2 = \emptyset$. The asynchronous product of M_1 and M_2, denoted $M_1 \otimes M_2$, is the POMDP $(Z, z^s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ where $Z = \{(z_1, z_2) \mid z_1 \in Z_1 \wedge z_2 \in Z_2\}$, $z^s = (z^s_1, z^s_2)$, $\mathsf{Act} = \mathsf{Act}_1 \cup \mathsf{Act}_2$, $\mathcal{O} = \{(o_1, o_2) \mid o_1 \in \mathcal{O}_1 \wedge o_2 \in \mathsf{obs}_2\}$, $\mathsf{obs}(z_1, z_2) = (\mathsf{obs}_1(z_1), \mathsf{obs}_2(z_2))$ and Δ is defined below.

$$\Delta((z_1, z_2), \alpha)((z'_1, z'_2)) = \begin{cases} \Delta_1(z_1, \alpha)(z'_1) & \text{if } z_2 = z'_2 \wedge \alpha \in \mathsf{Act}_1 \\ \Delta_2(z_2, \alpha)(z'_2) & \text{if } z_1 = z'_1 \wedge \alpha \in \mathsf{Act}_2 \\ 0 & \text{otherwise.} \end{cases}$$

Let $\mathcal{A}$ be an adversary for $M_1 \otimes M_2$. We will assume $\mathsf{Exec}(\llbracket (M_1 \otimes M_2) \rrbracket^{\mathcal{A}})$ is finite and every execution is of finite length. For $\rho \in \mathsf{Exec}(\llbracket (M_1 \otimes M_2) \rrbracket^{\mathcal{A}})$, we define its projection onto M_i, denoted $\pi_i(\rho)$ inductively as follows. If $\rho = (z^s_1, z^s_2)$, $\pi_i(\rho) = z^s_i$. Otherwise $\rho = \rho_0 \xrightarrow{\alpha} (z_1, z_2)$ and we distinguish two cases. If $\alpha \in \mathsf{Act}_i$ then $\pi_i(\rho) = \pi(\rho_0) \xrightarrow{\alpha} z_i$ and if $\alpha \notin \mathsf{Act}_i$ then $\pi_i(\rho) = \pi_i(\rho_0)$. The projection of a trace t of $M_1 \otimes M_2$ is defined analogously and will be denoted $\pi_i(t)$. As a shorthand, for a state $z = (z', z'')$ we will write z_1 for z' and z_2 for z''. For an observation o, o_1 and o_2 are defined similarly.

For a state z, POMDP M, adversary $\mathcal{A}$ and $\rho \in \mathsf{Exec}([\![M]\!]^{\mathcal{A}})$ let $\mathsf{prob}^z(\rho, M^{\mathcal{A}})$ be the measure of the event ρ starting from state z in $[\![M]\!]^{\mathcal{A}}$. Additionally, for an observation o and trace t, let $\rho_1, ..., \rho_m$ be the executions of $\mathsf{Exec}([\![M]\!]^{\mathcal{A}})$ such that $\mathsf{tr}(\rho_j) = t$ and its initial state is z_j where $\mathsf{obs}(z_j) = o$ for all $j \in \{1, ..., m\}$. Define $\mathsf{prob}^o(t, M^{\mathcal{A}}) = \sum_{j=1}^m \mathsf{prob}^{z_j}(\rho_j, M^{\mathcal{A}})$. Finally, for states z, z' and action α, define $\mathsf{trans}^z((\alpha, z'), M^{\mathcal{A}})$ to be the probability of transitioning to z' from z by the action α in $[\![M]\!]^{\mathcal{A}}$.

Definition 8. *Let $\mathcal{A}$ be an adversary for $M = M_1 \otimes M_2$ and t be a trace. Define $\mathsf{proj}_i^t(\mathcal{A}, M)$ as follows. For any trace t', $\mathsf{proj}_i^t(\mathcal{A}, M)(t') = \mathcal{A}(t_0)$ if t_0 is a maximal trace such that $t_0 \preceq t$ and $\pi_i(t_0) = t'$. Otherwise $\mathsf{proj}_i^t(\mathcal{A}, M)(t')$ is undefined.*

Lemma 10. *Let $\mathcal{A}$ be an adversary for $M_1 \otimes M_2$ and t be a trace. If $\mathcal{A}_i = \mathsf{proj}_i^t(\mathcal{A}, M_1 \otimes M_2)$ then $\mathsf{prob}(t, (M_1 \otimes M_2)^{\mathcal{A}}) = \mathsf{prob}(\pi_1(t), M_1^{\mathcal{A}_1}) \times \mathsf{prob}(\pi_2(t), M_2^{\mathcal{A}_2})$.*

Proof. The proof is by induction on the length of t, where the $|t|$ is defined to be the number of actions in t. For the base case, let $|t| = o^s \xrightarrow{\alpha} o'$ and assume without loss of generality that $\alpha \in \mathsf{Act}_1$. We have $\mathsf{prob}(\pi_2(t), M_2^{\mathcal{A}_2}) = \mathsf{prob}(\mathsf{obs}(z_2^s), M_2^{\mathcal{A}_2}) = 1$ and hence $\mathsf{prob}(t, (M_1 \otimes M_2)^{\mathcal{A}}) = \mathsf{prob}(\pi_1(t), M_1^{\mathcal{A}_1})$.

For the induction step, let $t = o^s \xrightarrow{\alpha} t_0$. We can assume without loss of generality that $\alpha \in \mathsf{Act}_1$ identifies an atomic action in $M_1 \otimes M_2$. Hence $o^s = (o_1^s, o_2^s)$ and the first observation of t_0 is $o = (o_1', o_2^s)$. Let $\rho_1, ..., \rho_m \in \mathsf{Exec}([\![M_1 \otimes M_2]\!]^{\mathcal{A}})$ be the executions such that $\mathsf{tr}(\rho_j) = t$ for $j \in \{1, ..., m\}$. Further let $\rho_j = z^s \xrightarrow{\alpha} \rho_j'$ and observe that every ρ_j' must have the same initial state $z' = (z_1', z_2^s)$. We have $\mathsf{prob}(t, (M_1 \otimes M_2)^{\mathcal{A}})$ is

$$\mathsf{trans}^{z^s}((\alpha, z'), (M_1 \otimes M_2)^{\mathcal{A}}) \times \textstyle\sum_{j=1}^m \mathsf{prob}^{z'}(\rho_j', (M_1 \otimes M_2)^{\mathcal{A}}).$$

Because $t = o^s \xrightarrow{\alpha} t_0$, we have $\mathsf{tr}(\rho_j) = t_0$ and $\sum_{j=1}^m \mathsf{prob}^{z'}(\rho_j', (M_1 \otimes M_2)^{\mathcal{A}}) = \mathsf{prob}^o(t_0, (M_1 \otimes M_2)^{\mathcal{A}})$. Let M_1' be the same as M_1 with the exception that its initial state is (z_1', z_2^s) and $\mathcal{A}'$ be the adversary for $M_1' \otimes M_2$ such that $\mathcal{A}'(t) = \mathcal{A}(o^s \xrightarrow{\alpha} t)$. If $\mathcal{A}_i' = \mathsf{proj}_i^t(\mathcal{A}', M_1' \otimes M_2)$, we have $\mathsf{prob}^o(t_0, (M_1 \otimes M_2)^{\mathcal{A}})$

$$\begin{aligned} &= \mathsf{prob}(t_0, (M_1' \otimes M_2)^{\mathcal{A}'}) \\ &= \mathsf{prob}(\pi_i(t_0), (M_1')^{\mathcal{A}_1'}) \times \mathsf{prob}(\pi_2(t_0), M_2^{\mathcal{A}_2'}) \\ &= \mathsf{prob}^{o_1}(\pi_1(t_0), M_1^{\mathcal{A}_1}) \times \mathsf{prob}^{o_2}(\pi_2(t_0), M_2^{\mathcal{A}_2}) \end{aligned}$$

where the second equality follows by the induction hypothesis and the first and third equalities follows by the definition of M_1' and $\mathcal{A}'$. Let $\rho_1'', ..., \rho_l''$ be the executions of $[\![M_1]\!]^{\mathcal{A}_1}$ such that $\mathsf{tr}(\rho_k'') = \pi_1(t_0)$ for $k \in \{1, ..., l\}$. Again, for every ρ_k'', its initial state must be z_1'. Observe that

$$\mathsf{prob}^{o_1}(\pi_1(t_0), M_1^{\mathcal{A}_1}) = \textstyle\sum_{k=1}^l \mathsf{prob}^{z_1'}(\rho_k'', M_1^{\mathcal{A}}).$$

Furthermore, because $\alpha \in \mathsf{Act}_1$, it must be the case that

$$\mathsf{trans}^{z^s}((\alpha, z'), (M_1 \otimes M_2)^{\mathcal{A}}) = \mathsf{trans}^{z_1^s}((\alpha, z_1'), M_1^{\mathcal{A}_1})$$

and $\mathsf{prob}^{o_2}(\pi(t_0), M_2^{\mathcal{A}_2}) = \mathsf{prob}(\pi(t_0), M_2^{\mathcal{A}_2})$. Putting everything together we have $\mathsf{prob}(t, (M_1 \otimes M_2)^{\mathcal{A}})$ is equal to

$$\mathsf{trans}^{z_1^s}((\alpha, z_1'), M_1^{\mathcal{A}_1}) \times \textstyle\sum_{k=1}^{l} \mathsf{prob}^{z_1'}(\rho_k'', M_1^{\mathcal{A}}) \times \mathsf{prob}(\pi_2(t_0), M_2^{\mathcal{A}_2})$$

which is the same as $\mathsf{prob}(\pi_1(t_0), M_1^{\mathcal{A}_1}) \times \mathsf{prob}(\pi_2(t_0), M_2^{\mathcal{A}_2})$. □

Lemma 11. *If $M_1 \approx M_1'$ then $M_1 \otimes M \approx M_1' \otimes M$.*

Proof. Assume for a contradiction that $M_1 \otimes M \not\approx M_1' \otimes M$. By definition, there exists a trace t and adversary $\mathcal{A}$ such that $\mathsf{prob}(t, (M_1 \otimes M)^{\mathcal{A}}) \neq \mathsf{prob}(t, (M_1' \otimes M)^{\mathcal{A}})$. Let $\mathcal{A}_1 = \mathsf{proj}_1^t(\mathcal{A}, M_1 \otimes M) = \mathsf{proj}_1^t(\mathcal{A}, M_1' \otimes M)$ and $\mathcal{A}_2 = \mathsf{proj}_2^t(\mathcal{A}, M_1 \otimes M) = \mathsf{proj}_2^t(\mathcal{A}, M_1' \otimes M)$. By Lemma 10, $\mathsf{prob}(\pi_1(t), M_1^{\mathcal{A}_1}) \times \mathsf{prob}(\pi_2(t), M^{\mathcal{A}_2}) \neq \mathsf{prob}(\pi_1(t), (M_1')^{\mathcal{A}_1}) \times \mathsf{prob}(\pi_2(t), M^{\mathcal{A}_2})$ and subsequently $\mathsf{prob}(\pi_1(t), M_1^{\mathcal{A}_1}) \neq \mathsf{prob}(\pi_1(t), (M_1')^{\mathcal{A}_1})$ follows from the fact that $\mathsf{prob}(\pi_2(t), M^{\mathcal{A}_2}) \neq 0$ if $\mathsf{prob}(t, (M_1 \otimes M)^{\mathcal{A}}) \neq \mathsf{prob}(t, (M_1' \otimes M)^{\mathcal{A}})$. That is, $M_1 \not\approx M_1'$, contradiction. □

Lemma 12. *If $M_1 \approx M_1'$ and $M_2 \approx M_2'$, then $M_1 \otimes M_2 \approx M_1' \otimes M_2'$.*

Proof. Follows from Lemma 11. □

C Proof of Theorem 1

For the remainder of this section, let $i \in \{1, ..., n\}$, $S = \{x_1, ..., x_n\}$ and

$$C[\Box_1, ..., \Box_n] = \nu k_1 \cdot ... \cdot \nu k_m \cdot (D_1[\Box_1] | D_2[\Box_2] | ... | D_n[\Box_n])$$

$$C'[\Box_1, ..., \Box_n] = \nu k_1' \cdot ... \cdot \nu k_m' \cdot (D_1'[\Box_1] | D_2'[\Box_2] | ... | D_n'[\Box_n])$$

be a contexts over $\mathcal{F}_c$ with labels from $\mathcal{L}_c$, $B_1, ..., B_n$ (resp. $B_1', ..., B_n'$) be basic processes over $\mathcal{F}_b$ with labels from $\mathcal{L}_b$ and assume that the conditions of Theorem 1 hold. We will henceforth refer to the preceding conditions by †.

By convention, the labels from each B_i come from different equivalence classes, which we will denote by $\sim_i$.

Lemma 13. *Let C, C' and $B_1, ..., B_n$ (resp. $B_1', ..., B_n'$) be as above. If $C[B_1, ..., B_n] \not\approx C'[B_1', ..., B_n']$ then*

$$C[\mathtt{out}(\sharp(x_1)) \cdot B_1, ..., \mathtt{out}(\sharp(x_n)) \cdot B_n]$$
$$\not\approx$$
$$C'[\mathtt{out}(\sharp(x_1)) \cdot B_1', ..., \mathtt{out}(\sharp(x_n)) \cdot B_n'].$$

Proof. Let $P_1 = C[B_1, ..., B_n]$, $P_2 = C'[B_1', ..., B_n']$, $P_1' = C[\mathtt{out}(\sharp(x_1)) \cdot B_1, ..., \mathtt{out}(\sharp(x_n)) \cdot B_n]$ and $P_2' = C'[\mathtt{out}(\sharp(x_1)) \cdot B_1', ..., \mathtt{out}(\sharp(x_n)) \cdot B_n']$. We will assume that $\mathtt{out}(\sharp(x_1)), ..., \mathtt{out}(\sharp(x_n))$ are labeled by $l_1, ..., l_n \in \mathcal{L}_b$. Define a projection π from executions of P_1' (resp. P_2') to executions of P_1 (resp. P_2) inductively as follows. If $\rho \in \mathsf{Exec}([\![P_1']\!])$ (resp. $\rho \in \mathsf{Exec}([\![P_2']\!])$), contains no actions, then

$\pi(P_1', \emptyset, \emptyset) = (P_1, \emptyset, \emptyset)$ (resp. $\pi(P_2', \emptyset, \emptyset) = (P_2, \emptyset, \emptyset)$). Inductively, let $\rho = \rho_0 \xrightarrow{\alpha} z$ for $\alpha = (\S, [l])$. If $l = l_i$ then $\pi(\rho) = \pi(\rho_0)$. Otherwise $\pi(\rho) = \pi(\rho_0) \xrightarrow{\alpha} z'$. The projection π can be extended to traces in the following way. For a trace t, $\pi(t) = \mathsf{tr}(\pi(\rho'))$ where ρ' is any execution such that $\mathsf{tr}(\rho') = t$. This extension is well defined. From π we can define an adversary $\mathcal{A}'$ for P_1' (resp. P_2') from an adversary $\mathcal{A}$ for P_1 (resp. P_2) in the following way. For a trace t, if $\mathcal{A}(\pi(t)) = (\S, [l])$ where $l \in \sim_i$ and t doesn't contain the action $(\tau, [l_i])$, then $\mathcal{A}'(t) = (\tau, [l_i])$. Otherwise $\mathcal{A}'(t) = \mathcal{A}(\pi(t))$. Let $\rho, \rho_1, \rho_2 \in \mathsf{MExec}([\![P_k']\!]^{\mathcal{A}'})$ for $k \in \{1, 2\}$. First observe that $\mathsf{prob}(\rho, (P_k')^{\mathcal{A}'}) = \mathsf{prob}(\pi(\rho), P_k{}^{\mathcal{A}})$ by our definition of $\mathcal{A}'$. Furthermore, if $\mathsf{tr}(\pi(\rho_1)) \neq \mathsf{tr}(\pi(\rho_2))$ then $\mathsf{tr}(\rho_1) \neq \mathsf{tr}(\rho_2)$. Because $P_1 \not\approx P_2$, by Proposition 1, there is an adversary $\mathcal{A}$ and trace t that is maximal for $[\![P_1]\!]^{\mathcal{A}}$ and $[\![P_2]\!]^{\mathcal{A}}$ such that $\mathsf{prob}(t, [\![P_1]\!]^{\mathcal{A}}) \neq \mathsf{prob}(t, [\![P_2]\!]^{\mathcal{A}})$. Let $t_1, ..., t_k$ be the traces of $(P_1')^{\mathcal{A}'}, (P_2')^{\mathcal{A}'}$ such that $\pi(t_j) = t$ for all $j \in \{1, ..., k\}$. We have

$$\mathsf{prob}(t, [\![P_1]\!]^{\mathcal{A}}) = \sum_{j=1}^{k} \mathsf{prob}(t_j, [\![P_1']\!]^{\mathcal{A}'})$$

and

$$\mathsf{prob}(t, [\![P_2]\!]^{\mathcal{A}}) = \sum_{j=1}^{k} \mathsf{prob}(t_j, [\![P_2']\!]^{\mathcal{A}'}).$$

By the preceding observations,

$$\sum_{j=1}^{k} \mathsf{prob}(t_j, [\![P_1']\!]^{\mathcal{A}'}) \neq \sum_{j=1}^{k} \mathsf{prob}(t_j, [\![P_2']\!]^{\mathcal{A}'})$$

and thus there must be some j such that $\mathsf{prob}(t_j, [\![(P_1']\!]^{\mathcal{A}'}) \neq \mathsf{prob}(t_j, [\![P_2']\!]^{\mathcal{A}'})$. That is, $P_1' \not\approx P_2'$. □

Lemma 14. *Let $\varphi_0, \varphi_0', \varphi_1, \varphi_1'$ be frames and $\mathcal{N}_0, \mathcal{N}_1$ be sets of names such that $\mathcal{N}_0 \cap \mathcal{N}_1 = \emptyset$ and the following hold.*

1. *$\forall u \in \mathsf{ran}(\varphi_0) \cup \mathsf{ran}(\varphi_0')$ and $\forall w_{i,[l]} \in \mathsf{dom}(\varphi_0) \cup \mathsf{dom}(\varphi_0')$:*
   - *$u \in \mathcal{T}(\mathcal{F} \cup \mathcal{N}_0 \cup \mathcal{M})$ and $l \in \mathcal{L}_b$*
2. *$\forall u \in \mathsf{ran}(\varphi_1) \cup \mathsf{ran}(\varphi_1')$ and $\forall w_{i,[l]} \in \mathsf{dom}(\varphi_1) \cup \mathsf{dom}(\varphi_1')$:*
   - *$u \in \mathcal{T}(\mathcal{F} \cup \mathcal{N}_1 \cup \mathcal{M})$ and $l \in \mathcal{L}_c$*
3. *$\varphi = \varphi_0 \cup \varphi_1$ and $\varphi' = \varphi_0' \cup \varphi_1'$*

Then $\varphi \equiv \varphi'$ iff $\varphi_0 \equiv \varphi_1$ and $\varphi_0' \equiv \varphi_1'$.

Proof. "⇒" We show the contrapositive. Assume without loss of generality that $\varphi_0 \not\equiv \varphi_0'$. That is, $\mathsf{dom}(\varphi) \neq \mathsf{dom}(\varphi')$ or there exist recipes r_1, r_2 such that $r_1\varphi_0 =_E r_2\varphi_0$ and $r_1\varphi_0' \neq_E r_2\varphi_0'$ (or vice versa, in which case the result follows by a similar argument). If $\mathsf{dom}(\varphi) \neq \mathsf{dom}(\varphi')$ then the result is immediate. Otherwise the free variables in r_1, r_2 must come from $\mathsf{dom}(\varphi_0) = \mathsf{dom}(\varphi_0')$ and thus they have the form $w_{i,[l]}$ for $l \in \mathcal{L}_b$. By definition, for all such $w_{i,[l]}$, we have

$w_{i,[l]}\varphi_0 = w_{i,[l]}\varphi$ and $w_{i,[l]}\varphi'_0 = w_{i,[l]}\varphi'$. That is, $r_1\varphi =_E r_2\varphi$ and $r_1\varphi' \neq_E r_2\varphi'$, which means that $\varphi \not\equiv \varphi'$.

"$\Leftarrow$" We show the contrapositive. Assume that $\varphi \not\equiv \varphi'$. Then there exists recipes r_a, r_b such that $r_a\varphi =_E r_b\varphi$ and $r_a\varphi' \neq_E r_b\varphi'$ (or vice versa, in which case the result follows by a similar argument). If $\mathsf{dom}(\varphi) \neq \mathsf{dom}(\varphi')$ or $r_a, r_b \in \mathsf{dom}(\varphi)$ then the result is immediate. In what follows, we will assume that for all $w_{i,[l_1]}, w_{j,[l_2]} \in \mathsf{dom}(\varphi)$, we have $w_{i,[l_1]}\varphi =_E w_{j,[l_2]}\varphi$ iff $w_{i,[l_1]}\varphi' =_E w_{j,[l_2]}\varphi'$. Let υ be a function on terms such that $\upsilon(t_1) = \upsilon(t_2)$ iff $t_1 =_E t_2$ and $\upsilon(t_1) \in [t_1]$. Let $\tilde{\varphi}$ (resp. $\tilde{\varphi}'$) be the frame that results from replacing every $u \in \mathsf{ran}(\varphi)$ (resp. $u \in \mathsf{ran}(\varphi')$) by $\upsilon(u)$. Clearly, for any recipe r, $r\varphi =_E r\tilde{\varphi}$ and $r\varphi' =_E r\tilde{\varphi}'$. Let $\mathsf{M}^0 : \mathcal{N}_0 \to M_0$ and $\mathsf{M}^1 : \mathcal{N}_1 \to M_1$ be bijections such that $M_0 \cap M_1 = \emptyset$, $M_0, M_1 \subset \mathcal{M}$ and no names in M_0, M_1 occur in φ, φ'. For some name $k \in \mathcal{N}_0$ (resp. $k \in \mathcal{N}_1$), we will write M^0_k (resp. M^1_k) to denote the value of k under such a bijection. Given a term u, we will write $\mathsf{M}^0(u)$ (resp. $\mathsf{M}^1(u)$) to denote the term that results from replacing in u, every $k \in \mathcal{N}_0$ (resp. $k \in \mathcal{N}_1$) by M^0_k (resp. M^1_k). Let r^0_a, r^0_b (resp. r^1_b, r^1_b) be the recipes that result from replacing every subrecipe $u = w_{i,[l]}$ in r^0_a, r^0_b (resp. r^1_b, r^1_b) where $l \in \mathcal{L}_c$ (resp. $l \in \mathcal{L}_b$) by the recipe $\mathsf{M}^1(u\varphi)$ (resp. $\mathsf{M}^0(u\varphi)$). By Definition 5, for any recipe r, we have $r\tilde{\varphi} =_E r^0\tilde{\varphi} =_E r^1\tilde{\varphi}$ and $r\tilde{\varphi}' =_E r^0\tilde{\varphi}' =_E r^1\tilde{\varphi}'$. Given this, it must be the case that either

$$r^0_a\tilde{\varphi} =_E r^0_b\tilde{\varphi} \text{ and } r^0_a\tilde{\varphi}' \neq_E r^0_b\tilde{\varphi}'$$

or

$$r^1_a\tilde{\varphi} =_E r^1_b\tilde{\varphi} \text{ and } r^1_a\tilde{\varphi}' \neq_E r^1_b\tilde{\varphi}'.$$

Because the free variables of r^0_a, r^0_b are over $\mathsf{dom}(\varphi_0)$ and the free variables of r^1_a, r^1_b are over $\mathsf{dom}(\varphi_1)$ we can conclude, by a similar argument as the previous direction that either $\varphi_0 \not\equiv \varphi'_0$ or $\varphi_1 \not\equiv \varphi'_1$. □

Definition 9. *Let L be a set of labels closed under $\sim$. An adversary $\mathcal{A}$ for a process R is said to be pure with respect to L if whenever $\mathcal{A}$ chooses an action $(r, [l])$, we have $r \in (\mathcal{F}, \mathcal{X}^L_w)$.*

Lemma 15. *Let P, P' be processes such that* $\mathsf{vars}(P) \cap \mathsf{vars}(Q) = \mathsf{vars}(P') \cap \mathsf{vars}(Q) = \emptyset$. *If $P \approx P'$ then $P|Q \approx P'|Q$.*

Proof. The proof is by induction on the number of probabilistic choices in Q. For the base case, let Q be a process that doesn't contain probabilistic choice and assume for a contradiction that $P|Q \not\approx P'|Q$. Define $S = \{k \mid \nu k \text{ occurs in } Q\}$ to be the set of all variable names that can be bound by the ν operator in Q. Further let $\mathsf{M} \subset \mathcal{M}$ be a set of names not occurring in P, P' and Q such that there exists a bijection from M to S. We will write M_k to denote the element of M mapped to by $k \in S$ under this bijection. Define Q' as the process that results from replacing, in Q, every occurrence of an atomic action νk by $(k := \mathsf{M}_k)$. We have that if $P|Q \not\approx P'|Q$ then $P|Q' \not\approx P'|Q'$, which is a straightforward consequence of Definition 5.

By convention, we can assume that P, P' are labeled by $\mathcal{L}_b$ and Q is labeled by $\mathcal{L}_c$. Let $\mathcal{A}$ be an adversary and t be a trace such that $\mathsf{prob}(t, (P|Q')^{\mathcal{A}}) \neq$

$\mathsf{prob}(t, (P'|Q')^{\mathcal{A}})$. From $\mathcal{A}$, we define an adversary $\mathcal{A}'$ for $P|Q'$ and $P'|Q'$ that is pure with respect to $\mathcal{L}_b$ as follows. Let $\rho \in \mathsf{Exec}([\![P|Q']\!]^{\mathcal{A}})$ be such that $\mathsf{tr}(\rho) = t$ and $\mathsf{last}(\rho) = (R, \varphi, \sigma)$. Define mappings $\Theta_\rho : \mathsf{dom}(\sigma) \cap \mathsf{vars}(Q') \to \mathcal{T}(\mathcal{F} \cup \mathcal{M}, \mathsf{dom}(\varphi))$ and $\Phi_\rho : \mathsf{dom}(\varphi) \to \mathcal{T}(\mathcal{F} \cup \mathcal{M}, \mathsf{dom}(\varphi))$ by induction on the length of ρ. If ρ contains no actions then $\mathsf{dom}(\sigma) = \mathsf{dom}(\varphi) = \emptyset$. Inductively let $\rho = \rho_0 \xrightarrow{\alpha} z$. We distinguish 4 cases.

case 1: α does not execute an output action or an input/assignment action that binds a variable in $\mathsf{dom}(\sigma) \cap \mathsf{vars}(Q')$. Define $\Theta_\rho = \Theta_{\rho_0}$ and $\Phi_\rho = \Phi_{\rho_0}$.

case 2: α executes output action of the form $\mathtt{out}(u)$ where $u \in \mathcal{T}(\mathcal{F}, \mathsf{dom}(\sigma))$. Define $\Theta_\rho = \Theta_{\rho_0}$. The output action must bind a frame variable $w_{j,[l]}$ to the value $u\sigma$. If $l \in \mathcal{L}_c$ then $\Phi_\rho(w_{j,[l]}) = u\Theta_{\rho_0}$. Otherwise, if $l \in \mathcal{L}_b$ then $\Phi_\rho(w_{j,[l]}) = w_{j,[l]}$.

case 3: α executes a action of the form $\mathtt{in}(x)$ from Q'. The action α must be of the form $(r, [l])$. Define $\Theta_\rho(x) = r\Phi_{\rho_0}$ and $\Phi_\rho = \Phi_{\rho_0}$.

case 4: α executes a action of the form $(x := u)$ from Q' where $u \in \mathcal{T}(\mathcal{F}, \mathsf{dom}(\sigma))$. Define $\Theta_\rho(x) = u\Theta_{\rho_0}$ and $\Phi_\rho = \Phi_{\rho_0}$.

We extend Φ_ρ to actions by requiring that $\Phi_\rho(\tau, [l]) = (\tau, [l])$ and $\Phi_\rho(r, [l]) = (r\Phi_\rho, [l])$. Now if

$$t' = \mathsf{obs}(z_0) \xrightarrow{\alpha_0} \dots \xrightarrow{\alpha_{k-1}} \mathsf{obs}(z_k)$$

is a prefix of t and $\mathcal{A}(t') = \alpha_k$ let

$$\mathcal{A}'(\mathsf{obs}(z_0) \xrightarrow{V'_\rho(\alpha_0)} \dots \xrightarrow{V'_\rho(\alpha_{k-1})} \mathsf{obs}(z_k)) = V'_\rho(\alpha_k).$$

In all other cases, $\mathcal{A}'$ is undefined. To see that $\mathcal{A}'$ is well defined, one can show that $\rho \in \mathsf{Exec}([\![P|Q']\!]^{\mathcal{A}})$ iff $\rho \in \mathsf{Exec}([\![P|Q']\!]^{\mathcal{A}'})$ and $\mathsf{prob}(\rho, (P|Q')^{\mathcal{A}}) = \mathsf{prob}(\rho, (P|Q')^{\mathcal{A}'})$ by a simple induction on the length of ρ. By a similar argument, we have $\rho' \in \mathsf{Exec}([\![P'|Q']\!]^{\mathcal{A}})$ iff $\rho' \in \mathsf{Exec}([\![P'|Q']\!]^{\mathcal{A}'})$ and $\mathsf{prob}(\rho', (P'|Q')^{\mathcal{A}}) = \mathsf{prob}(\rho,' (P'|Q')^{\mathcal{A}'})$. The preceding facts imply that $\mathcal{A}'$ is an adversary for $P|Q'$ and $P'|Q'$ such that $\mathsf{prob}(t, (P|Q')^{\mathcal{A}}) = \mathsf{prob}(t, (P|Q')^{\mathcal{A}'})$ and $\mathsf{prob}(t, (P'|Q')^{\mathcal{A}}) = \mathsf{prob}(t, (P'|Q')^{\mathcal{A}'})$. That is $\mathsf{prob}(t', (P|Q')^{\mathcal{A}'}) \neq \mathsf{prob}(t', (P'|Q')^{\mathcal{A}'})$. By our construction, $\mathcal{A}'$ doesn't use an recipes output by Q'. By Lemma 14 and Lemma 11, we can conclude that $P \not\approx P'$, contradiction.

For the induction step, assume for a contradiction that $P|Q \not\approx P'|Q$. Let Q' be the result of replacing, in Q, some occurrence of a subprocess $Q_0 +_p Q_1$ by $\mathtt{out}(0)^{l_0} \cdot Q_0 +_p \mathtt{out}(1)^{l_1} \cdot Q_1$ where $0, 1$ are fresh unary constant symbols. Define a projection π from executions of $P|Q'$ (resp. $P'|Q'$) to executions of $P|Q$ (resp. $P'|Q$) inductively as follows. If $\rho \in \mathsf{Exec}([\![P|Q']\!])$ (resp. $\rho \in \mathsf{Exec}([\![P'|Q']\!])$) contains no actions, then $\pi(P|Q', \emptyset, \emptyset) = (P|Q, \emptyset, \emptyset)$ (resp. $\pi(P'|Q', \emptyset, \emptyset) = (P'|Q, \emptyset, \emptyset)$). Inductively, let $\rho = \rho_0 \xrightarrow{\alpha} z$ for $\alpha = (\S, [l])$. If $l \in \{l_0, l_1\}$ then $\pi(\rho) = \pi(\rho_0)$. Otherwise $\pi(\rho) = \pi(\rho_0) \xrightarrow{\alpha} z$. The projection π can be extended to traces by requiring that $\pi(t) = \mathsf{tr}(\pi(\rho))$. From π we can define an adversary $\mathcal{A}'$ for $P|Q'$ (resp. $P'|Q'$) from an adversary $\mathcal{A}$ for $P|Q$ (resp. $P'|Q$) in the following way. For a trace t, if $\mathcal{A}(\pi(t)) = (\S, [l])$ where l occurs in Q_0 (resp. Q_1) and t doesn't contain the action $(\tau, [l_0])$ (resp. $(\tau, [l_1])$), then $\mathcal{A}'(t) = (\tau, [l_0])$ (resp. $\mathcal{A}'(t) = (\tau, [l_1])$). Otherwise $\mathcal{A}'(t) = \mathcal{A}(\pi(t))$. Clearly we have $\mathsf{prob}(\rho, (P|Q)^{\mathcal{A}}) = \mathsf{prob}(\pi(\rho), (P|Q')^{\mathcal{A}'})$ and $\mathsf{prob}(\rho, (P'|Q)^{\mathcal{A}}) = \mathsf{prob}(\pi(\rho), (P'|Q')^{\mathcal{A}'})$.

Furthermore, for any $\rho_1, \rho_2 \in \mathsf{Exec}([\![P|Q']\!]^{\mathcal{A}'})$ (resp. $\rho_1, \rho_2 \in \mathsf{Exec}([\![P'|Q']\!]^{\mathcal{A}'})$) if $\mathsf{tr}(\rho_1) \neq \mathsf{tr}(\rho_2)$ then $\mathsf{tr}(\pi(\rho_1)) \neq \mathsf{tr}(\pi(\rho_2))$. Because $P|Q \not\approx P'|Q$ there is an adversary $\mathcal{A}$ and trace t such that $\mathsf{prob}(t, (P|Q)^{\mathcal{A}}) \neq \mathsf{prob}(t, (P'|Q)^{\mathcal{A}})$. Let $t_1, ..., t_v$ be the traces of $(P|Q')^{\mathcal{A}'}, (P'|Q')^{\mathcal{A}'}$ such that $\mathsf{tr}(t_j) = t$ for all $j \in \{1, ..., v\}$. By the preceding observations, there is some j such that $\mathsf{prob}(t_j, (P|Q')^{\mathcal{A}'}) \neq \mathsf{prob}(t_j, (P'|Q')^{\mathcal{A}'})$ and hence $P|Q' \not\approx P'|Q'$. Let $\rho_0, \rho_1 \in \mathsf{Exec}([\![P|Q']\!]^{\mathcal{A}'})$ (resp. $\rho_0, \rho_1 \in \mathsf{Exec}([\![P'|Q']\!]^{\mathcal{A}'})$) be executions such that ρ_0 contains the action $(\tau, [l_0])$ and ρ_1 contains the action $(\tau, [l_1])$. We have $\mathsf{tr}(\rho_0) \neq \mathsf{tr}(\rho_1)$. From this, we can conclude that $P|Q'_0 \not\approx P'|Q'_0$ or $P|Q'_1 \not\approx P'|Q'_1$ where Q'_0 (resp. Q'_1) is the process that results from replacing, in Q, some occurrence of the subprocess $Q_0 +_p Q_1$ by Q_0 (resp. Q_1). By the induction hypothesis, we have that$P|Q'_0 \approx P'|Q'_0$ and $P|Q'_1 \approx P'|Q'_1$, contradiction. $\square$

Lemma 16. *Let P, P', Q, Q' be processes such that* $\mathsf{vars}(P) \cap \mathsf{vars}(Q) = \emptyset$ *and* $\mathsf{vars}(P') \cap \mathsf{vars}(Q') = \emptyset$*. If $P \approx P'$ and $Q \approx Q'$ then $P|Q \approx P'|Q'$.*

Proof. By Lemma 15, we have $P|Q \approx P'|Q$ and $P'|Q \approx P'|Q'$ from which the result follows. $\square$

Lemma 17. *Let P, P', Q, Q' be processes such that* $\mathsf{vars}(P) \cap \mathsf{vars}(Q) = \{x\}$ *and* $\mathsf{vars}(P') \cap \mathsf{vars}(Q') = \{x\}$*. If $\nu x \cdot (P|Q) \approx \nu x \cdot (P'|Q')$ then*

$$\nu x_1 \cdot P\{x \mapsto x_1\} | \nu x_2 \cdot Q\{x \mapsto x_2\}$$
$$\approx$$
$$\nu x_1 \cdot P'\{x \mapsto x_1\} | \nu x_2 \cdot Q'\{x \mapsto x_2\}.$$

Proof. We begin by showing that $\nu x \cdot P \approx \nu x \cdot P'$ and $\nu x \cdot Q \approx \nu x \cdot Q'$. We only give the agruement for $\nu x \cdot P \approx \nu x \cdot P'$ as the case of $\nu x \cdot Q \approx \nu x \cdot Q'$ is similar. Assume for a contradiction that $\nu x \cdot P \not\approx \nu x \cdot P'$. Let $t = \mathsf{obs}(z_0) \xrightarrow{\alpha_1} ... \xrightarrow{\alpha_m} \mathsf{obs}(z_m)$ be a trace such that $\mathsf{obs}(z_k) = (\mathsf{enabled}(z_k), \upsilon(\varphi_k))$. Define $t|Q$ to be the trace $o'_0 \xrightarrow{\alpha_1} ... \xrightarrow{\alpha_m} o'_m$ such that $o'_k = (\mathsf{enabled}(z_k) \cup \mathsf{enabled}(Q, \emptyset, \emptyset), \upsilon(\varphi_k))$. Because $\nu x \cdot P \not\approx \nu x \cdot P'$, there exists an adversary $\mathcal{A}$ and trace t such that $\mathsf{prob}(t, (\nu x \cdot P)^{\mathcal{A}}) \neq \mathsf{prob}(t, (\nu x \cdot P')^{\mathcal{A}})$. Define $\mathcal{A}'$ to be the adversary for $\nu x \cdot (P|Q)$ and $\nu x \cdot (P'|Q')$ such that $\mathcal{A}'(t|Q) = \mathcal{A}(t)$. For any execution $\rho \in \mathsf{Exec}([\![P]\!]^{\mathcal{A}})$ (resp. $\rho \in \mathsf{Exec}([\![P']\!]^{\mathcal{A}})$) of the form $(\nu x \cdot P, \emptyset, \emptyset) \xrightarrow{\alpha_1} ... \xrightarrow{\alpha_k} (P_0, \varphi_0, \sigma_0)$ let $\rho|Q$ be the execution $(\nu x \cdot (P|Q), \emptyset, \emptyset) \xrightarrow{\alpha_1} ... \xrightarrow{\alpha_k} (P_0|Q, \varphi_0, \sigma_0)$. For any executions $\rho_1, \rho_2 \in \mathsf{Exec}([\![\nu x \cdot P]\!]^{\mathcal{A}})$ we have $\rho_1|Q, \rho_2|Q \in \mathsf{Exec}([\![\nu x \cdot (P|Q)]\!]^{\mathcal{A}'})$ where the measure of ρ_1 is the same as the measure of $\rho_1|Q$ and $\mathsf{tr}(\rho_1) = \mathsf{tr}(\rho_2)$ iff $\mathsf{tr}(\rho_1|Q) = \mathsf{tr}(\rho_2|Q)$. That is, $\mathsf{prob}(t|Q, (\nu x \cdot (P|Q))^{\mathcal{A}'}) = \mathsf{prob}(t, (\nu x \cdot (P|Q))^{\mathcal{A}})$. By a similar argument, we can conclude that $\mathsf{prob}(t|Q', (\nu x \cdot (P'|Q'))^{\mathcal{A}'}) = \mathsf{prob}(t, (\nu x \cdot (P'|Q'))^{\mathcal{A}})$. That is, $\mathsf{prob}(t|Q, (\nu x \cdot (P|Q))^{\mathcal{A}'}) \neq \mathsf{prob}(t|Q', (\nu x \cdot (P'|Q'))^{\mathcal{A}'})$ and hence $\nu x \cdot (P|Q) \not\approx \nu x \cdot (P'|Q')$, contradiction. Given that $\nu x \cdot P \approx \nu x \cdot P'$ and $\nu x \cdot Q \approx \nu x \cdot Q'$ the result follows by Lemma 16. $\square$

Lemma 18. *Assuming †, we have $C[B_1, ..., B_n] \approx C'[B_1, ..., B_n]$.*

Proof. Assume for a contradiction that $C[B_1, ..., B_n] \not\approx C'[B_1, ..., B_n]$. Let $P_1 = C[\mathtt{out}(\sharp(x_1)) \cdot B_1, ..., \mathtt{out}(\sharp(x_n)) \cdot B_n]$ and $P_2 = C'[\mathtt{out}(\sharp(x_1)) \cdot B_1, ..., \mathtt{out}(\sharp(x_n)) \cdot$

$B_n]$. By Lemma 13, $P_1 \not\approx P_2$. Let $\mathcal{A}$ be an adversary such that $\mathsf{prob}(t, P_1^{\mathcal{A}}) \neq \mathsf{prob}(t, P_2^{\mathcal{A}})$ for some trace t. By Lemma 9, there exists some $M \in \mathbb{N}$ such that P_1, P_2 are transposable to $\Delta(P_1, \mathcal{A}, M), \Delta(P_2, \mathcal{A}, M)$. By Lemma 8, this yields $\Delta(P_1, \mathcal{A}, M) \not\approx \Delta(P_2, \mathcal{A}, M)$. Let t' be a trace and $\mathcal{A}'$ be an adversary such that $\mathsf{prob}(t', \Delta(P_1, \mathcal{A}, M)^{\mathcal{A}'}) \neq \mathsf{prob}(t', \Delta(P_2, \mathcal{A}, M)^{\mathcal{A}'})$. Further, let $\rho_1, \rho_2 \in \mathsf{Exec}([\![\Delta(P_1, \mathcal{A}, M)]\!]^{\mathcal{A}'})$ be such that $\mathsf{tr}(\rho_1) = \mathsf{tr}(\rho_2) = t'$. If t' contains an action with a label from $\sim_i$, then the first such action in ρ_1 (resp. ρ_2) is an output of a term of the form $\sharp((x_i^{\pi_1})^b \sigma_1)$ (resp. $\sharp((x_i^{\pi_2})^b \sigma_2)$) where $\mathsf{last}(\rho_k) = (R_k, \varphi_k, \sigma_k)$ for $k \in \{1, 2\}$. We will write $\sharp^1(x_i)$ (resp. $\sharp^2(x_i)$) to denote the first (and only) output of ρ_1 (resp. ρ_2) with a label from $\sim_i$. Observe that if ρ_1 contains actions with labels from $\sim_i$ and $\sim_j$ then $\sharp^1(x_i) =_E \sharp^1(x_j)$ iff $\sharp^2(x_i) =_E \sharp^2(x_j)$. If this was not the case, then ρ_1 and ρ_2 would not have the same trace. Notice that the above observation also holds when $\rho_2 \in \mathsf{Exec}([\![\Delta(P_2, \mathcal{A}, M)]\!]^{\mathcal{A}'})$.

Let $B_0 := \nu k_0 \cdot ... \cdot \nu k_n \cdot (x_1 := z_1) \cdot ... \cdot (x_1 := z_n)$ be a process such that $z_i \in \{k_0, ..., k_n\}$ and the following hold. If t' doesn't contain an action from $\sim_i$ then $z_i = k_0$ and otherwise $z_i = z_j$ iff $\sharp^1(x_i) = \sharp^1(x_j)$. By definition, $\Delta(P_1, \mathcal{A}, M)$ and $\Delta(P_2, \mathcal{A}, M)$, both contain a prefix of the form $\nu k'_1 \cdot ... \cdot \nu k'_v \cdot (y_1^b := z_1) \cdot ... \cdot (y_v^b := z_v)$. Furthermore, each prefix has the same length. Let $B'_0 = [\top] \cdot ... \cdot [\top] \cdot B_0$ such that $|B'_0| = 2v$. We will assume that the actions of B'_0 are labeled sequentially by $l_1, ..., l_{2v} \in \mathcal{L}_b$. Using the adversary $\mathcal{A}'$ and the trace t', we construct an adversary $\mathcal{A}''$ for $C[\mathtt{out}(\sharp(x_1)), ..., \mathtt{out}(\sharp(x_n))]|B'$ and $C'[\mathtt{out}(\sharp(x_1)), ..., \mathtt{out}(\sharp(x_n))]|B'$ where $B' = B'_0 \cdot (B_1|...|B_n)$ as follows. If t'' is a prefix of t' then $\mathcal{A}''(t'') = \mathcal{A}'(t'')$. Otherwise, $\mathcal{A}''$ is undefined. By our construction, we have

$$\begin{gathered}\mathsf{prob}(t', (C[\mathtt{out}(\sharp(x_1)), ..., \mathtt{out}(\sharp(x_n))]|B')^{\mathcal{A}''}) \\ \neq \\ \mathsf{prob}(t', (C'[\mathtt{out}(\sharp(x_1)), ..., \mathtt{out}(\sharp(x_n))]|B')^{\mathcal{A}''})\end{gathered}$$

which means $C[\mathtt{out}(\sharp(x_1)), ..., \mathtt{out}(\sharp(x_n))]|B' \not\approx C'[\mathtt{out}(\sharp(x_1)), ..., \mathtt{out}(\sharp(x_n))]|B'$. Applying Lemma 15, we get

$$C[\mathtt{out}(\sharp(x_1)), ..., \mathtt{out}(\sharp(x_n))] \not\approx C'[\mathtt{out}(\sharp(x_1)), ..., \mathtt{out}(\sharp(x_n))]$$

which contradictions condition 5 of Theorem 1. □

Lemma 19. $C'[B_1, ..., B_n] \approx C'[B'_1, ..., B'_n]$.

Proof. By a similar arguement as the one used in Lemma 18, there exists a process $B_0 := \nu k_0 \cdot ... \cdot \nu k_n \cdot (x_1 := z_1) \cdot ... \cdot (x_1 := z_n)$ where $z_i \in \{k_0, ..., k_n\}$ for all $i \in \{1, ..., n\}$ such that

$$B_0 \cdot (B_1|...|B_n) \not\approx B_0 \cdot (B'_1|...|B'_n).$$

By Lemma 17, we can conclude that

$$\begin{gathered}\nu k \cdot (x_1 := k) \cdot .... \cdot (x_n := k) \cdot (B_1|...|B_n) \\ \not\approx \\ \nu k \cdot (x'_1 := k) \cdot .... \cdot (x'_n := k) \cdot (B'_1|...|B'_n)\end{gathered}$$

which contradictions condition 6 of Theorem 1. □

Theorem 1 is a consequence of Lemma 18 and Lemma 19.

D Proof of Theorem 2

Let C be a context and B be a basic process, both over the equational theory $(\mathcal{F}_{\mathsf{senc}}, E_{\mathsf{senc}})$ where $\mathcal{F}_{\mathsf{senc}} = \{\mathsf{senc}, \mathsf{sdec}, \mathsf{h}\}$ and $E_{\mathsf{senc}} = \{\mathsf{sdec}(\mathsf{senc}(m,k),k) = m\}$. To securely compose C and B, the terms occurring in each protocol must be tagged by function symbols from disjoint equational theories. The tagging of two protocols will be done in two steps. To begin, a signature renaming function $\_^d$ will be applied to each of C and B with distinct values of $d \in \{b, c\}$. The function $\_^d$ transforms a context C over the signature $(\mathcal{F}_{\mathsf{senc}}, E_{\mathsf{senc}})$ to a context C^d by replacing every occurrence of the function symbols $\mathsf{senc}, \mathsf{sdec}$ and h in C by $\mathsf{senc}_d, \mathsf{sdec}_d$ and h_d, respectively. The resulting context C^d is over the signature $(\mathcal{F}^d_{\mathsf{senc}}, E^d_{\mathsf{senc}})$, for $\mathcal{F}^d_{\mathsf{senc}} = \{\mathsf{senc}_d, \mathsf{sdec}_d, \mathsf{h}_d\}$ and $E^d_{\mathsf{senc}} = \{\mathsf{sdec}_d(\mathsf{senc}_d(m,k),k) = m\}$. Given C^c and B^b over the disjoint signatures $\mathcal{F}^c_{\mathsf{senc}}$ and $\mathcal{F}^b_{\mathsf{senc}}$, the tagging function $\lceil \_ \rceil$ is then applied to C^c and B^b, generating the the tagged versions of C and B.

We now give the formal definition of the tagging function $\lceil \_ \rceil$. Let $\mathcal{F}^d_{\mathsf{tag}} = \{\mathsf{tag}_d, \mathsf{untag}_d\}$ and $E^d_{\mathsf{tag}} = \{\mathsf{untag}_d(\mathsf{tag}_d(x)) = x\}$. Further, $\mathcal{F}_{\mathsf{tag}} = \mathcal{F}^b_{\mathsf{tag}} \cup \mathcal{F}^c_{\mathsf{tag}}$ and $E_{\mathsf{tag}} = E^b_{\mathsf{tag}} \cup E^c_{\mathsf{tag}}$. The function $\mathcal{H} : \mathcal{T}(\mathcal{F}^d_{\mathsf{senc}}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}^d_{\mathsf{tag}}, \mathcal{X})$ is defined below.

$$\begin{aligned}
\mathcal{H}(\mathsf{senc}_d(u_1,u_2)) &= \mathsf{senc}(\mathsf{tag}_d(\mathcal{H}(u_1)), \mathcal{H}(u_2))\\
\mathcal{H}(\mathsf{sdec}_d(u_1,u_2)) &= \mathsf{untag}_d(\mathsf{sdec}(\mathcal{H}(u_1), \mathcal{H}(u_2)))\\
\mathcal{H}(\mathsf{h}_d(u)) &= \mathsf{h}(\mathsf{tag}_d(\mathcal{H}(u)))\\
\mathcal{H}(u) &= u, \text{ if } u \text{ is a name or variable}
\end{aligned}$$

The function tests^d below maps terms from $\mathcal{T}(\mathcal{F}_{enc} \cup \mathcal{F}^d_{tag}, \mathcal{X})$ to a conjunction of equalities, as defined below.

$$\begin{aligned}
\mathsf{tests}^d(\mathsf{senc}(u_1,u_2)) &= \mathsf{tests}^d(u_1) \wedge \mathsf{tests}^d(u_2)\\
\mathsf{tests}^d(\mathsf{sdec}(u_1,u_2)) &= \mathsf{tests}^d(u_1) \wedge \mathsf{tests}^d(u_2)\\
\mathsf{tests}^d(\mathsf{h}(u)) &= \mathsf{tests}^d(u)\\
\mathsf{tests}^d(\mathsf{tag}_d(u)) &= \mathsf{tests}^d(u)\\
\mathsf{tests}^d(\mathsf{untag}_d(u)) &= \mathsf{tag}_d(\mathsf{untag}_d(u)) = \mathsf{tag}_d(u) \wedge\\
&\quad\ \mathsf{tests}^d(u)\\
\mathsf{tests}^d(u) &= \top, \text{ if } u \text{ is a name or variable}
\end{aligned}$$

For a term u, observe that $\mathsf{tests}^d(u) = c_1 \wedge ... \wedge c_n$ where c_i is $\top$ or $v_1 = v_2$ for ground terms $v_1, v_2 \in \mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}$. We say that $\mathsf{tests}^d(u)$ *passes* if c_i is $\top$ or $v_1 =_{E_{\mathsf{senc}} \cup E_{\mathsf{tag}}} v_2$ for all $i \in \{1, ..., n\}$. Using the preceding notions, we define $\lceil \_ \rceil$ as follows.

Definition 10. *Let B^d be basic process over $\mathcal{F}^d_{\mathsf{senc}}$ for $d \in \{b, c\}$. The basic process $\lceil B^d \rceil$ is defined as follows.*

$$\begin{aligned}
\lceil \square \rceil &= \square \\
\lceil \nu x \rceil &= [\top] \cdot \nu x \\
\lceil \mathtt{in}(x) \rceil &= [\top] \cdot \mathtt{in}(x) \\
\lceil \mathtt{out}(u) \rceil &= [\mathsf{tests}^d(\mathcal{H}(u))] \cdot \mathtt{out}(\mathcal{H}(u)) \\
\lceil (x := u) \rceil &= [\mathsf{tests}^d(\mathcal{H}(u))] \cdot (x := \mathcal{H}(u)) \\
\lceil [u = v] \rceil &= [\mathsf{tests}^d(\mathcal{H}(u)) \wedge \mathsf{tests}^d(\mathcal{H}(v))] \cdot \\
&\quad [\mathcal{H}(u) = \mathcal{H}(v)] \\
\lceil B_1 \cdot B_2 \rceil &= \lceil B_1 \rceil \cdot \lceil B_2 \rceil \\
\lceil B_1 +_p B_2 \rceil &= [\top] \cdot \lceil B_1 \rceil +_p [\top] \cdot \lceil B_2 \rceil
\end{aligned}$$

Definition 10 can be lifted naturally to basic contexts by requiring $\lceil \square \rceil = \square$ for any process variable $\square$.

Definition 11. *Let $C^d = a_1 \cdot \ldots \cdot a_n \cdot (D_1[\square_1] | \ldots | D_n(\square_n))$ be a context over $\mathcal{F}^d_{\mathsf{senc}}$ for $d \in \{b, c\}$. The context $\lceil C^d \rceil$ is $\lceil a_1 \cdot \ldots \cdot a_n \rceil \cdot (\lceil D_1[\square_1] \rceil | \ldots | \lceil D_n(\square_n) \rceil)$.*

The following example demonstrates the behavior of processes that are tagged using our scheme. Whenever a protocol manipulates a term, that term should be tagged with the identifier of the protocol. To enforce this, every observable action in a tagged protocol is prefixed with a conjunction of tests. If the terms manipulated by the atomic action meet the aforementioned requirement, the tests will pass. Otherwise, the tests will fail, and further protocol actions will be blocked. In this way, messages from one protocol cannot be confused with messages from another protocol.

Example 10. Let $P = \nu n \cdot \nu m \cdot \mathtt{out}(\mathsf{senc}(m, n, k))$ and $Q = \mathtt{in}(x) \cdot \mathtt{out}(\mathsf{sdec}(x, k))$ where the processes P and Q share a key k. We have $\lceil P^b \rceil = [\top] \cdot \nu n \cdot [\top] \cdot \nu m \cdot [\top] \cdot \mathtt{out}(\mathsf{senc}(\mathsf{tag}_b(m), n, k))$ and $\lceil Q^c \rceil = [\top] \cdot \mathtt{in}(x) \cdot [\mathsf{tag}_c(\mathsf{untag}_c(\mathsf{sdec}(x, k))) = \mathsf{sdec}(x, k)] \cdot \mathtt{out}(\mathsf{untag}_c(\mathsf{sdec}(x, k)))$. If the output of $\lceil P^b \rceil$ is forwarded to $\lceil Q^c \rceil$ then the test

$$\begin{gathered}
\mathsf{tag}_c(\mathsf{untag}_c(\mathsf{sdec}(\mathsf{senc}(\mathsf{tag}_b(m), n, k), k)) \\
= \\
\mathsf{sdec}(\mathsf{senc}(\mathsf{tag}_b(m), n, k), k)
\end{gathered}$$

reduces to $\mathsf{tag}_c(\mathsf{untag}_c(\mathsf{tag}_b(m)) = \mathsf{tag}_b(m)$ but ultimately blocks because $c \neq b$.

For ease of notation, let $E = E_{\mathsf{senc}} \cup E_{\mathsf{tag}}$. Let $d \in \{b, c\}$ be a symbolic identifier used for tagging processes. We will write d' to denote the new identifier c' if $d = c$ and b' if $d = b$. Using these identifiers, we define new equational theories for tagged processes as follows. Let

$$\mathcal{F}^{d'}_{\mathsf{tag}} = \{\mathsf{tag}_{d'}, \mathsf{untag}_{d'}\}$$

and

$$E^{d'}_{\mathsf{tag}} = \{\mathsf{untag}_{d'}(\mathsf{tag}_{d'}(x)) = x\}.$$

To achieve Theorem 2, we will map an attack on a tagged process over the equational theory E to an attack on a process over disjoint signatures in the extended equational theory E_0 defined below. Let

$$\mathcal{F}^0 = \mathcal{F}^b_{\mathsf{senc}} \cup \mathcal{F}^c_{\mathsf{senc}} \cup \mathcal{F}^a_{\mathsf{senc}} \cup \mathcal{F}^{b'}_{\mathsf{tag}} \cup \mathcal{F}^{c'}_{\mathsf{tag}}$$

and

$$E^0 = E^b_{\mathsf{senc}} \cup E^c_{\mathsf{senc}} \cup E^a_{\mathsf{senc}} \cup E^{b'}_{\mathsf{tag}} \cup E^{c'}_{\mathsf{tag}}.$$

We now define a function $\lfloor \_ \rfloor : \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}^0, \mathcal{X})$. This function is an adaptation of the one from [27].

$$\begin{array}{ll}
\lfloor \mathsf{tag}_d(u) \rfloor & = \mathsf{tag}_{d'}(\lfloor u \rfloor) \\
\lfloor \mathsf{untag}_d(u) \rfloor & = \mathsf{untag}_{d'}(\lfloor u \rfloor) \\
\lfloor \mathsf{senc}(u_1, u_2) \rfloor & = \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor), \lfloor u_2 \rfloor) \text{ if} \\
 & \quad \lfloor u_1 \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor)) \\
 & = \mathsf{senc}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor) \text{ otherwise} \\
\lfloor \mathsf{sdec}(u_1, u_2) \rfloor & = \mathsf{tag}_{d'}(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)) \text{ if} \\
 & \quad \lfloor u_1 \rfloor =_{E^0} \mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor) \\
 & = \mathsf{sdec}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor) \text{ otherwise} \\
\lfloor \mathsf{h}(u) \rfloor & = \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor u \rfloor)) \text{ if} \\
 & \quad \lfloor u \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u \rfloor)) \\
 & = \mathsf{h}_a(\lfloor u \rfloor) \text{ otherwise} \\
\lfloor u \rfloor & = u \text{ for a name or a variable}
\end{array}$$

Let u be a term and $\overrightarrow{E}$ be an orientation of the equations of E from left to right. We will write $u \to_E v$ to denote that u rewrites to v and $u \to^*_E v$ if u rewrites to v in 0 or more steps. The normal form of u will be denoted $u\downarrow$. A term u is in *head normal form* if any sequence of rewrites on u cannot happen at the root.

Lemma 20. *Let $u, v \in \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X})$. If $u \to_E v$ then $\lfloor u \rfloor =_{E^0} \lfloor v \rfloor$.*

Proof. The proof is by induction on the structure of u. For the base case, let u be a name or variable. If $u \to_E v$ then $u = v$ and the goal is immediate. For the induction step, we proceed by cases.

case 1: $u = \mathsf{tag}_d(u_1)$. By the definition of E, u does not rewrite in the root symbol. That is, if $u \to_E v$ then $v = \mathsf{tag}_d(v_1)$ where $u_1 \to_E v_1$. By definition, $\lfloor u \rfloor = \mathsf{tag}_{d'}(\lfloor u_1 \rfloor)$ and $\lfloor v \rfloor = \mathsf{tag}_{d'}(\lfloor v_1 \rfloor)$. By the I.H. we have $\lfloor u_1 \rfloor =_{E^0} \lfloor v_1 \rfloor$ and the case follows.

case 2: $u = \mathsf{untag}_d(u_1)$. We consider two cases. First assume that $u_1 = \mathsf{tag}_d(u_2)$ and $u \to_E u_2$. We have the following.

$$\begin{array}{rl}
\lfloor u \rfloor = & \mathsf{untag}_{d'}(\lfloor u_1 \rfloor) \\
= & \mathsf{untag}_{d'}(\mathsf{tag}_{d'}(\lfloor u_2 \rfloor)) \\
=_{E^0} & \lfloor u_2 \rfloor
\end{array}$$

Otherwise $u \to_E v$ where $v = \mathsf{untag}_d(u_1')$. By the I.H. $\lfloor u_1 \rfloor =_{E^0} \lfloor u_1' \rfloor$ and the result follows.

case 3: $u = \mathsf{senc}(u_1, u_2)$. By the definition of E, u does not rewrite in the root symbol. That is, $v = \mathsf{senc}(v_1, v_2)$ where $u_1 \to_E v_1$ and $u_2 = v_2$ (or $u_1 = v_2$ and $u_2 \to_E v_2$, which follows by a similar argument). By the I.H. $\lfloor u_1 \rfloor =_{E^0} \lfloor v_1 \rfloor$. We consider two subcases.

In the first, $\lfloor u_1 \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$. Because $\lfloor u_1 \rfloor =_{E^0} \lfloor v_1 \rfloor$, we have $\lfloor v_1 \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor v_1 \rfloor))$. By definition, we know $\lfloor u \rfloor = \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor), \lfloor u_2 \rfloor)$ and $\lfloor v \rfloor = \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor v_1 \rfloor), \lfloor u_2 \rfloor)$ and thus $\lfloor u \rfloor =_{E^0} \lfloor v \rfloor$.

In the second case, $\lfloor u_1 \rfloor \neq_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$. Because $\lfloor u_1 \rfloor =_{E^0} \lfloor v_1 \rfloor$, we have $\lfloor v_1 \rfloor \neq_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor v_1 \rfloor))$. That is, $\lfloor u \rfloor = \mathsf{senc}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)$ and $\lfloor v \rfloor = \mathsf{senc}_a(\lfloor v_1 \rfloor, \lfloor u_2 \rfloor)$ and thus $\lfloor u \rfloor =_{E^0} \lfloor v \rfloor$.

case 4: $u = \mathsf{sdec}(u_1, u_2)$. We consider two subcases.

subcase 4.1: $u_1 = \mathsf{senc}(u_3, u_2)$ and $u \to_E u_3$. First assume that the following equation holds.

$$\lfloor u_3 \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_3 \rfloor)) \tag{2}$$

Because $u_1 = \mathsf{senc}(u_3, u_2)$, by the I.H. we have $\lfloor u_1 \rfloor =_{E^0} \lfloor \mathsf{senc}(u_3, u_2) \rfloor = \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_3 \rfloor), \lfloor u_2 \rfloor)$ where the latter equality follows by equation 2. From the proceeding facts, we have $\mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)$

$$\begin{aligned}
&=_{E^0} \mathsf{senc}_d(\mathsf{sdec}_d(\mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_3 \rfloor), \lfloor u_2 \rfloor), \lfloor u_2 \rfloor), \lfloor u_2 \rfloor) \\
&=_{E^0} \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_3 \rfloor), \lfloor u_2 \rfloor) \\
&=_{E^0} \lfloor u_1 \rfloor
\end{aligned}$$

The result is a consequence of the following derivation.

$$\begin{aligned}
\lfloor u \rfloor = \quad & \mathsf{tag}_{d'}(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)) \\
=_{E^0} \; & \mathsf{tag}_{d'}(\mathsf{sdec}_d(\mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_3 \rfloor), \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)) \\
=_{E^0} \; & \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_3 \rfloor)) \\
=_{E^0} \; & \lfloor u_3 \rfloor
\end{aligned}$$

Now assume that equation 2 does not hold. Because $u_1 \to_E \mathsf{senc}(u_3, u_2)$, by the I.H. we have $\lfloor u_1 \rfloor =_{E^0} \lfloor \mathsf{senc}(u_3, u_2) \rfloor = \mathsf{senc}_a(\lfloor u_3 \rfloor, \lfloor u_2 \rfloor)$, where the latter equality follows from the fact that equation 2 does not hold. By the definition of E_0, $\mathsf{senc}_a(\lfloor u_3 \rfloor, \lfloor u_2 \rfloor)$ does not rewrite in the root symbol and $\lfloor u_1 \rfloor \downarrow = \mathsf{senc}_a(u_3', u_2')$. We also have $\mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)) \downarrow = \mathsf{senc}_d(\mathsf{sdec}_d(\mathsf{senc}_a(u_3', u_4'), u_2')$. Notice that $\lfloor u_1 \rfloor \downarrow$ and $\mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)$ have normal forms of different size. Because E_0 is a convergent rewrite system, this means that $\lfloor u_1 \rfloor \neq_{E^0} \mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)$ and thus $\lfloor u \rfloor = \mathsf{sdec}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)$. The case follows by the derivation below.

$$\begin{aligned}
\lfloor u \rfloor = \quad & \mathsf{sdec}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)) \\
=_{E^0} \; & \mathsf{sdec}_a(\mathsf{senc}_a(\lfloor u_3 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)) \\
=_{E^0} \; & \lfloor u_3 \rfloor
\end{aligned}$$

case 4.2: $u \to_E v$ where $v = \mathsf{sdec}(u_1', u_2)$ and $u_1 \to_E u_1'$. The case when $v = \mathsf{sdec}(u_1, u_2')$ and $u_2 \to_E u_2'$ follows by a similar argument. By the I.H. $\lfloor u_1 \rfloor = \lfloor u_1' \rfloor$ and the case is straightforward.

case 5: $u = \mathsf{h}(u_1)$. Follows by a similar argument as case 2. □

Lemma 21. *Let $u, v \in \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X})$ be terms in normal form. Then $u \neq v$ implies $\lfloor u \rfloor \neq_{E^0} \lfloor v \rfloor$.*

Proof. Define a function $\lfloor \_ \rfloor_1 : \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}^0, \mathcal{X})$ which is identical to $\lfloor \_ \rfloor$ on all cases with the exception that $\lfloor \mathsf{sdec}(u_1, u_2) \rfloor_1 = \mathsf{sdec}_a(\lfloor u_1 \rfloor_1, \lfloor u_2 \rfloor_1)$. Formally,

$$
\begin{aligned}
\lfloor \mathsf{tag}_d(u) \rfloor_1 &= \mathsf{tag}_{d'}(\lfloor u \rfloor_1) \\
\lfloor \mathsf{untag}_d(u) \rfloor_1 &= \mathsf{untag}_{d'}(\lfloor u \rfloor_1) \\
\lfloor \mathsf{senc}(u_1, u_2) \rfloor_1 &= \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor_1), \lfloor u_2 \rfloor_1) \text{ if} \\
&\quad\ \lfloor u_1 \rfloor_1 =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor_1)) \\
&= \mathsf{senc}_a(\lfloor u_1 \rfloor_1, \lfloor u_2 \rfloor_1) \text{ otherwise} \\
\lfloor \mathsf{sdec}(u_1, u_2) \rfloor_1 &= \mathsf{sdec}_a(\lfloor u_1 \rfloor_1, \lfloor u_2 \rfloor_1) \\
\lfloor \mathsf{h}(u) \rfloor_1 &= \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor u \rfloor_1)) \text{ if} \\
&\quad\ \lfloor u \rfloor_1 =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u \rfloor_1)) \\
&= \mathsf{h}_a(\lfloor u \rfloor_1) \text{ otherwise} \\
\lfloor u \rfloor_1 &= u \text{ for a name or a variable.}
\end{aligned}
$$

We show the following.

(i) $\lfloor u \rfloor = \lfloor u \rfloor_1$ and $\lfloor v \rfloor = \lfloor v \rfloor_1$
(ii) $\lfloor u \rfloor$, $\lfloor v \rfloor$ are in head normal form
(iii) $u \neq v$ implies $\lfloor u \rfloor \neq_{E^0} \lfloor v \rfloor$

by induction on $\mathsf{max}(|u|, |v|)$. We can assume without loss of generality that $|v| \leq |u|$ and hence need only show items (i) and (ii) hold for the term u. For the base case, when u is a name or a variable, we have $u = \lfloor u \rfloor = \lfloor u \rfloor_1$ from which items (i) and (ii) are obvious. We also have $v = \lfloor v \rfloor$ and clearly $\lfloor u \rfloor = u \neq_{E^0} v = \lfloor v \rfloor$. For the induction step, we proceed by a case analysis on u.

case 1: $u = \mathsf{tag}_d(u_1)$.

(i) We have $\lfloor u \rfloor = \mathsf{tag}_{d'}(\lfloor u_1 \rfloor)$ and $\lfloor u \rfloor_1 = \mathsf{tag}_{d'}(\lfloor u_1 \rfloor_1)$. By the I.H. $\lfloor u_1 \rfloor =_{E^0} \lfloor u_1 \rfloor_1$ and thus $\lfloor u \rfloor =_{E^0} \lfloor u \rfloor_1$.

(ii) By definition $\lfloor u \rfloor = \mathsf{tag}_{d'}(\lfloor u_1 \rfloor)$ and clearly $\lfloor u \rfloor$ is in head normal form.

(iii) We again do a case analysis on v. Assume $\lfloor u \rfloor =_E \lfloor u \rfloor$. By items (i) and (ii), the only interesting cases are when $v = \mathsf{tag}_d(v_1)$ or $v = \mathsf{sdec}(v_1, v_2)$. When $v = \mathsf{tag}_d(v_1)$, it must be the case that $u_1 \neq v_1$ because $u \neq v$. We have $\lfloor u \rfloor = \mathsf{tag}_{d'}(\lfloor u_1 \rfloor)$ and $\lfloor v \rfloor = \mathsf{tag}_{d'}(\lfloor v_1 \rfloor)$. By the I.H. $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$ and it follows that $\lfloor u \rfloor \neq_{E^0} \lfloor v \rfloor$.

When $v = \mathsf{sdec}(v_1, v_2)$, by item (i) we have $\lfloor v \rfloor = \lfloor v \rfloor_1$ and thus $\lfloor v \rfloor = \mathsf{sdec}_a(\lfloor v_1 \rfloor, \lfloor v_2 \rfloor)$. That is, $\lfloor u \rfloor{\downarrow} = \mathsf{tag}_{d'}(u_1')$ and $\lfloor v \rfloor{\downarrow} = \mathsf{sdec}_a(v_1', v_2')$ and clearly $\lfloor u \rfloor \neq_{E^0} \lfloor v \rfloor$.

case 2: $u = \mathsf{untag}_d(u_1)$.

(i) We have $\lfloor u \rfloor = \mathsf{untag}_{d'}(\lfloor u_1 \rfloor)$ and $\lfloor u \rfloor_1 = \mathsf{untag}_{d'}(\lfloor u_1 \rfloor_1)$. By the I.H. $\lfloor u_1 \rfloor =_{E^0} \lfloor u_1 \rfloor_1$ and thus $\lfloor u \rfloor =_{E^0} \lfloor u \rfloor_1$.

(ii) We again do a case analysis of u_1. First note that $u_1 \neq \mathsf{tag}_d(u_2)$ for any u_2. Otherwise we would have $u = \mathsf{untag}_d(\mathsf{tag}_d(u_2)) \rightarrow_E u_2$, contradicting the fact that u is in normal form. Now consider the case when $u_1 = \mathsf{sdec}(u_2, u_3)$. By the I.H. $\lfloor u_1 \rfloor = \lfloor u_1 \rfloor_1 = \mathsf{sdec}_a(\lfloor u_2 \rfloor, \lfloor u_3 \rfloor)$. Then we have $\lfloor u \rfloor = \mathsf{untag}_{d'}(\mathsf{sdec}_a(\lfloor u_2 \rfloor, \lfloor u_3 \rfloor))$ which clearly does not rewrite in the root symbol. The remaining cases are straightforward.

(iii) By item (ii), $\lfloor u \rfloor = \mathsf{untag}_{d'}(\lfloor u_1 \rfloor)$ is in head normal form. We do a case analysis of v, with the only interesting case being when $v = \mathsf{untag}_d(v_1)$. Again by item (ii), we have $\lfloor v \rfloor = \mathsf{untag}_{d'}(\lfloor v_1 \rfloor)$ is in head normal form. It then suffices to show $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$. Because $u \neq v$, $u_1 \neq v_1$ and the I.H. yields $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$.

case 3: $u = \mathsf{senc}(u_1, u_2)$.

(i) By the I.H. $\lfloor u_1 \rfloor = \lfloor u_1 \rfloor_1$ and $\lfloor u_2 \rfloor = \lfloor u_2 \rfloor_1$. From this we have $\lfloor u_1 \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$ iff $\lfloor u_1 \rfloor_1 =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor_1))$ and it follows that $\lfloor u \rfloor = \lfloor u \rfloor_1$.

(ii) By definition, $\lfloor u \rfloor = \mathsf{senc}_e(u_3', u_4')$ for some terms u_3', u_4' and $e \in \{a, b, c\}$. Clearly, $\mathsf{senc}_e(u_3', u_4')$ is in head normal form.

(iii) We do a case analysis on v. By item (ii), the only interesting case is when $v = \mathsf{senc}(v_1, v_2)$. In this case $u_1 \neq v_1$ or $u_2 \neq v_2$. By the I.H. $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$ or $\lfloor u_2 \rfloor \neq_{E^0} \lfloor v_2 \rfloor$. Clearly if $\lfloor u \rfloor = \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor), \lfloor u_2 \rfloor)$ and $\lfloor v \rfloor = \mathsf{senc}_a(\lfloor v_1 \rfloor, \lfloor v_2 \rfloor)$ (or vice-versa) then $\lfloor u \rfloor \neq_{E^0} \lfloor v \rfloor$. If $\lfloor u \rfloor = \mathsf{senc}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)$ and $\lfloor v \rfloor = \mathsf{senc}_a(\lfloor v_1 \rfloor, \lfloor v_2 \rfloor)$ then the result follows from the fact that $\lfloor u \rfloor, \lfloor v \rfloor$ are in head normal form and $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$ or $\lfloor u_2 \rfloor \neq_{E^0} \lfloor v_2 \rfloor$. When $\lfloor u \rfloor = \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor), \lfloor u_2 \rfloor)$ and $\lfloor v \rfloor = \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor v_1 \rfloor), \lfloor v_2 \rfloor)$ the result follows by a similar argument.

case 4: $u = \mathsf{sdec}(u_1, u_2)$.

(i) We do a case analysis on u_1. By the I.H. $\lfloor u_1 \rfloor$ is in head normal form and thus the only interesting case is when $u_1 = \mathsf{senc}(u_3, u_4)$. Assume for a contradiction that $\lfloor u_1 \rfloor =_{E^0} \mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)$. By definition, $\lfloor u_1 \rfloor = \mathsf{senc}_d(u_3', \lfloor u_4 \rfloor)$ for some term u_3'. In particular, this means that $\lfloor u_4 \rfloor =_{E^0} \lfloor u_2 \rfloor$. By the I.H. (contrapositive of item (iii)) $u_2 = u_4$. Then we have $u = \mathsf{sdec}(\mathsf{senc}(u_3, u_2), u_2) \rightarrow u_3$, which contradicts the fact that u is in normal form. That is, $\lfloor u_1 \rfloor \neq_{E^0} \mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)$ and $\lfloor u \rfloor = \mathsf{sdec}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)$. By the I.H. $\lfloor u_1 \rfloor = \lfloor u_1 \rfloor_1$ and $\lfloor u_2 \rfloor = \lfloor u_2 \rfloor_1$ from which it follows that $\lfloor u \rfloor = \lfloor u \rfloor_1$.

(ii) By item (i), $\lfloor u \rfloor = \lfloor u \rfloor_1$ and we have $\lfloor u \rfloor = \mathsf{sdec}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)$. Notice that if $\lfloor u \rfloor$ is not in head normal form then $\lfloor u_1 \rfloor =_{E^0} \mathsf{senc}_a(\lfloor u_3 \rfloor, \lfloor u_2 \rfloor)$ for some u_3. By the definition of $\lfloor \_ \rfloor$ and the fact that $\lfloor u_1 \rfloor$ is in head normal form (by the I.H.), it must be the case that $u_1 = \mathsf{senc}(u_3, u_4)$. Because u is in normal form, $u_1 \neq \mathsf{senc}(u_3, u_2)$ for any u_3. Otherwise we would have $u = \mathsf{sdec}(\mathsf{senc}(u_3, u_2), u_2) \rightarrow_E u_3$. That is, $\lfloor u_1 \rfloor \neq_{E^0} \mathsf{senc}_a(\lfloor u_3 \rfloor, \lfloor u_2 \rfloor)$ for any u_3 and the result follows.

(iii) By items (i) and (ii), $\lfloor u \rfloor = \lfloor u \rfloor_1 = \mathsf{sdec}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)$ is in head normal form. We do a case analysis on v, with the only interesting case being when $v = \mathsf{sdec}(v_1, v_2)$. Then again by items (i) and (ii), we have $\lfloor v \rfloor = \lfloor v \rfloor_1 = \mathsf{sdec}_a(\lfloor v_1 \rfloor, \lfloor v_2 \rfloor)$ is in head normal form. Because $u \neq v$ and either $u_1 \neq v_1$ or

$u_2 \neq v_2$. By the I.H. either $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$ or $\lfloor u_2 \rfloor \neq_{E^0} \lfloor v_2 \rfloor$ from which the result follows.

case 5: $u = \mathsf{h}(u_1)$.

(i) By the I.H. $\lfloor u_1 \rfloor = \lfloor u_1 \rfloor_1$. From this we have $\lfloor u_1 \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$ iff $\lfloor u_1 \rfloor_1 =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor_1))$ and it follows that $\lfloor u \rfloor = \lfloor u \rfloor_1$.

(ii) By definition, $\lfloor u \rfloor = \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$ or $\lfloor u \rfloor = \mathsf{h}_d(\lfloor u_1 \rfloor)$. Clearly, both $\mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$ and $\mathsf{h}_d(\lfloor u_1 \rfloor)$ are in head normal form.

(iii) We do a case analysis on v. By item (ii), the only interesting case is when $v = \mathsf{h}(v_1)$. Because $u \neq v$, it must be the case that $u_1 \neq v_1$ and by the I.H. we have $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$. Clearly if $\lfloor u \rfloor = \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$ and $\lfloor v \rfloor = \mathsf{h}_a(\lfloor u_1 \rfloor)$ (or vice-versa) then $\lfloor u \rfloor \neq_{E^0} \lfloor v \rfloor$. If $\lfloor u \rfloor = \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$ and $\lfloor v \rfloor = \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor v_1 \rfloor))$ or $\lfloor u \rfloor = \mathsf{h}_a(\lfloor u_1 \rfloor)$ and $\lfloor v \rfloor = \mathsf{h}_a(\lfloor v_1 \rfloor)$ then the result follows from the fact that $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$. □

Lemma 22. *Let $u, v \in \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X})$. We have $u =_E v$ iff $\lfloor u \rfloor =_{E^0} \lfloor v \rfloor$.*

Proof. The proof follows from Lemma 20 and Lemma 21. □

Let φ be a frame and σ be a substitution over $\mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}})$. Define $\lfloor \varphi \rfloor = \{\lfloor w\varphi \rfloor \mid w \in \mathsf{dom}(\varphi)\}$ and $\lfloor \sigma \rfloor = \{\lfloor x\sigma \rfloor \mid x \in \mathsf{dom}(\sigma)\}$. Further define a function $\Diamond_\varphi : \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X}_w) \to \mathcal{T}(\mathcal{F}_0, \mathcal{X}_w)$ as follows. This function is an adaptation of the one from [27].

$$
\begin{array}{ll}
\Diamond_\varphi(\mathsf{tag}_d(r)) & = \mathsf{tag}_{d'}(\Diamond_\varphi(r)) \\
\Diamond_\varphi(\mathsf{untag}_d(r)) & = \mathsf{untag}_{d'}(\Diamond_\varphi(r)) \\
\Diamond_\varphi(\mathsf{senc}(r_1, r_2)) & = \mathsf{senc}_d(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1)), \Diamond_\varphi(r_2)) \text{ if} \\
 & \quad \Diamond_\varphi(r_1)\lfloor\varphi\rfloor =_{E^0} \\
 & \quad \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1)))\lfloor\varphi\rfloor \\
 & = \mathsf{senc}_a(\Diamond_\varphi(r_1), \Diamond_\varphi(r_2)) \text{ otherwise} \\
\Diamond_\varphi(\mathsf{sdec}(r_1, r_2)) & = \mathsf{tag}_{d'}(\mathsf{sdec}_d(\Diamond_\varphi(r_1), \Diamond_\varphi(r_2))) \text{ if} \\
 & \quad \Diamond_\varphi(r_1)\lfloor\varphi\rfloor =_{E^0} \\
 & \quad \mathsf{senc}_d(\mathsf{sdec}(\Diamond_\varphi(r_1), \Diamond_\varphi(r_2)), \Diamond_\varphi(r_2))\lfloor\varphi\rfloor \\
 & = \mathsf{sdec}_a(\Diamond_\varphi(r_1), \Diamond_\varphi(r_2)) \\
\Diamond_\varphi(\mathsf{h}(r)) & = \mathsf{h}_d(\mathsf{untag}_{d'}(\Diamond_\varphi(r))) \text{ if} \\
 & \quad \Diamond_\varphi(r)\lfloor\varphi\rfloor =_{E^0} \\
 & \quad \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\Diamond_\varphi(r))\lfloor\varphi\rfloor \\
 & = \mathsf{h}_a(\Diamond_\varphi(r)) \text{ otherwise} \\
\Diamond_\varphi(w) & = w \text{ for } w \in \mathsf{dom}(\varphi)
\end{array}
$$

Lemma 23. *Let φ be a frame over $\mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}})$ and $r \in \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X}_w)$. Then $\lfloor r\varphi \rfloor =_{E^0} \Diamond_\varphi(r)\lfloor\varphi\rfloor$.*

Proof. The proof is by induction on the structure of r. For the base case, when r is a frame variable w, we have $\Diamond_\varphi(w)\lfloor\varphi\rfloor = w\lfloor\varphi\rfloor = \lfloor w\varphi \rfloor$. For the induction step, we proceed by cases.

case 1: $r = \mathsf{tag}_d(r_1)$. We have $\lfloor \mathsf{tag}_d(r_1)\varphi \rfloor = \mathsf{tag}_{d'}(\lfloor r_1\varphi \rfloor)$ and $\Diamond_\varphi(\mathsf{tag}_d(r_1))\lfloor\varphi\rfloor = \mathsf{tag}_{d'}(\Diamond_\varphi(r_1)\lfloor\varphi\rfloor)$. By the I.H. $\lfloor r_1\varphi\rfloor =_{E^0} \Diamond_\varphi(r_1)\lfloor\varphi\rfloor$ and the case follows.

case 2: $r = \mathsf{h}(r_1)$. If $\Diamond_\varphi(\mathsf{h}(r_1)) = \mathsf{h}_d(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1)))$ then $\Diamond_\varphi(r_1)\lfloor\varphi\rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1)))\lfloor\varphi\rfloor$. By the I.H. $\Diamond_\varphi(r_1)\lfloor\varphi\rfloor =_{E^0} \lfloor r_1\varphi\rfloor$ and we have $\lfloor r_1\varphi\rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor r_1\varphi\rfloor))$. By definition, this means that $\lfloor \mathsf{h}(r_1)\varphi\rfloor = \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor r_1\varphi\rfloor))$ and we have the following.

$$\begin{aligned}\lfloor \mathsf{h}(r_1)\varphi\rfloor &= \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor r_1\varphi\rfloor))\\ &=_{E^0} \mathsf{h}_d(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1)\lfloor\varphi\rfloor))\\ &= \Diamond_\varphi(\mathsf{h}(r_1))\lfloor\varphi\rfloor\end{aligned}$$

If $\Diamond_\varphi(\mathsf{h}(r_1)) = \mathsf{h}_a(\Diamond_\varphi(r_1))$ then we can show by a similar argument that $\lfloor \mathsf{h}(r_1)\varphi\rfloor = \mathsf{h}_a(\lfloor r_1\varphi\rfloor)$ and the case again follows by the I.H.

The remaining cases are similar. □

Lemma 24. *Let φ_1, φ_2 be frames over $\mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}})$. If $\lfloor\varphi_1\rfloor \equiv_{E^0} \lfloor\varphi_2\rfloor$ then $\varphi_1 \equiv_E \varphi_2$.*

Proof. We show the contrapositive. Assume $\varphi_1 \not\equiv_E \varphi_2$. By definition, there exist recipes r and r' such that $r\varphi_1 \neq_E r'\varphi_1$ and $r\varphi_2 =_E r'\varphi_2$ (or vice-versa). We show

$$\Diamond_{\varphi_1}(r)\lfloor\varphi_1\rfloor \neq_E \Diamond_{\varphi_1}(r')\lfloor\varphi_1\rfloor$$

and

$$\Diamond_{\varphi_1}(r)\lfloor\varphi_2\rfloor =_E \Diamond_{\varphi_1}(r')\lfloor\varphi_2\rfloor.$$

By Lemma 22, if $r\varphi_1 \neq_E r'\varphi_1$ then $\lfloor r\varphi_1\rfloor \neq_{E^0} \rfloor r'\varphi_1\rfloor$. Using Lemma 23, we have $\Diamond_\varphi(r)\lfloor\varphi_1\rfloor \neq_{E^0} \Diamond_\varphi(r')\lfloor\varphi_1\rfloor$. By a similar argument, we can derive $\Diamond_{\varphi_1}(r)\lfloor\varphi_2\rfloor =_E \Diamond_{\varphi_1}(r')\lfloor\varphi_2\rfloor$ and hence $\lfloor\varphi_1\rfloor \not\equiv_{E^0} \lfloor\varphi_2\rfloor$. □

Lemma 25. *Let φ, φ' be frames over $\mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}})$ such that $\lfloor\varphi\rfloor \equiv_{E^0} \lfloor\varphi'\rfloor$. Then $\Diamond_\varphi(r) = \Diamond_{\varphi'}(r)$.*

Proof. The proof is by induction on the structure of r. For the base case, when r is an frame variable w, we have $\Diamond_\varphi(w) = \Diamond_{\varphi'}(w) = w$. For the induction step, we proceed by cases.

case 1: $r = \mathsf{tag}_d(r_1)$. By definition, $\Diamond_\varphi(r) = \mathsf{tag}_{d'}(\Diamond_\varphi(r_1))$ and $\Diamond_{\varphi'}(r) = \mathsf{tag}_{d'}(\Diamond_{\varphi'}(r_1))$. From the I.H. $\Diamond_\varphi(r_1) = \Diamond_{\varphi'}(r_1)$ and it follows that $\Diamond_\varphi(r) = \Diamond_{\varphi'}(r)$.

case 2: $r = \mathsf{h}(r_1)$. By the I.H. we can define $r'_1 = \Diamond_\varphi(r_1) = \Diamond_{\varphi'}(r_1)$ and $r'_2 = \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1))) = \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\Diamond_{\varphi'}(r_1)))$. Because $\lfloor\varphi\rfloor \equiv_{E^0} \lfloor\varphi'\rfloor$, we have $r'_1\lfloor\varphi\rfloor =_{E^0} r'_2\lfloor\varphi\rfloor$ iff $r'_1\lfloor\varphi'\rfloor =_{E^0} r'_2\lfloor\varphi'\rfloor$. This means that if $\Diamond_\varphi(\mathsf{h}(r_1)) = \mathsf{h}_d(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1)))$ then $\Diamond_{\varphi'}(\mathsf{h}(r_1)) = \mathsf{h}_d(\mathsf{untag}_{d'}(\Diamond_{\varphi'}(r_1)))$ and if $\Diamond_\varphi(\mathsf{h}(r_1)) = \mathsf{h}_a(\Diamond_\varphi(r_1))$ then $\Diamond_{\varphi'}(\mathsf{h}(r_1)) = \mathsf{h}_a(\Diamond_{\varphi'}(r_1))$. In either case, the result follows by the I.H.

The remaining cases are similar. □

Proposition 3. *Let $u \in \mathcal{T}(\mathcal{F}^d_{\mathsf{senc}}, \mathcal{X})$ and $u_1 \in \mathsf{st}(u)$. If $\mathsf{tests}^d(\mathcal{H}(u)\sigma)$ passes then $\mathsf{tests}^d(\mathcal{H}(u_1)\sigma)$ passes.*

Lemma 26. *Let $u \in \mathcal{T}(\mathcal{F}^d_{\mathsf{senc}}, \mathcal{X})$ and σ be a substitution over $\mathcal{F}^d_{\mathsf{senc}}$. If $\mathsf{tests}^d(\mathcal{H}(u)\sigma)$ passes then $\lfloor \mathcal{H}(u)\sigma \rfloor =_{E^0} u\lfloor \sigma \rfloor$.*

Proof. The proof is by induction on the structure of u. For the base case, when u is a name or variable we have $\lfloor \mathcal{H}(u)\sigma \rfloor = \lfloor u\sigma \rfloor = u\lfloor \sigma \rfloor$. For the induction step, we do a case analysis. By Proposition 3, we can apply the I.H. on any subterm of u.

case 1: $u = \mathsf{h}_d(u_1)$. By definition, $\mathcal{H}(u) = \mathsf{h}(\mathsf{tag}_d(\mathcal{H}(u_1)))$ and $\lfloor \mathcal{H}(u)\sigma \rfloor = \lfloor \mathsf{h}(\mathsf{tag}_d(\mathcal{H}(u_1)\sigma)) \rfloor$. Let $v = \mathsf{tag}_d(\mathcal{H}(u_1)\sigma)$. We have

$$\begin{aligned} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor v \rfloor)) &= \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor \mathsf{tag}_d(\mathcal{H}(u_1)\sigma) \rfloor)) \\ &= \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\mathsf{tag}_{d'}(\lfloor \mathcal{H}(u_1)\sigma \rfloor))) \\ &=_{E^0} \mathsf{tag}_{d'}(\lfloor \mathcal{H}(u_1)\sigma \rfloor) \\ &= \lfloor v \rfloor. \end{aligned}$$

The result is a consequence of the following derivation.

$$\begin{aligned} \lfloor \mathcal{H}(u)\sigma \rfloor &= \lfloor \mathsf{h}(\mathsf{tag}_d(\mathcal{H}(u_1)\sigma)) \rfloor \\ &= \mathsf{h}_d(\mathsf{untag}_{d'}(\mathsf{tag}_{d'}(\lfloor \mathcal{H}(u_1)\sigma \rfloor))) \\ &=_{E^0} \mathsf{h}_d(\lfloor \mathcal{H}(u_1)\sigma \rfloor) \\ &=_{E^0} \mathsf{h}_d(u_1\lfloor \sigma \rfloor) && \text{(I.H.)} \\ &= \mathsf{h}_d(u_1)\lfloor \sigma \rfloor \end{aligned}$$

case 2: $u = \mathsf{senc}_d(u_1, u_2)$. By definition, $\mathcal{H}(u) = \mathsf{senc}(\mathsf{tag}_d(\mathcal{H}(u_1)), \mathcal{H}(u_2))$ and $\lfloor \mathcal{H}(u)\sigma \rfloor = \lfloor \mathsf{senc}(\mathsf{tag}_d(\mathcal{H}(u_1)\sigma), \mathcal{H}(u_2)\sigma) \rfloor$. Let $v = \mathsf{tag}_d(\mathcal{H}(u_1)\sigma)$. Identically to case 1, we can show that $\mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor v \rfloor)) = \lfloor v \rfloor$. The result is a consequence of the following derivation.

$$\begin{aligned} \lfloor \mathcal{H}(u)\sigma \rfloor &= \lfloor \mathsf{senc}(\mathsf{tag}_d(\mathcal{H}(u_1)\sigma), \mathcal{H}(u_2)\sigma) \rfloor \\ &= \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor \mathsf{tag}_d(\mathcal{H}(u_1)\sigma) \rfloor), \lfloor \mathcal{H}(u_2)\sigma \rfloor) \\ &= \mathsf{senc}_d(\mathsf{untag}_{d'}(\mathsf{tag}_{d'}(\lfloor \mathcal{H}(u_1)\sigma \rfloor)), \lfloor \mathcal{H}(u_2)\sigma \rfloor) \\ &=_{E^0} \mathsf{senc}_d(\lfloor \mathcal{H}(u_1)\sigma \rfloor, \lfloor \mathcal{H}(u_2)\sigma \rfloor) \\ &=_{E^0} \mathsf{senc}_d(u_1\lfloor \sigma \rfloor, u_2\lfloor \sigma \rfloor) && \text{(I.H.)} \\ &= \mathsf{senc}_d(u_1, u_2)\lfloor \sigma \rfloor \end{aligned}$$

case 3: $u = \mathsf{sdec}_d(u_1, u_2)$. By definition $\mathcal{H}(u)\sigma = \mathsf{untag}_d(\mathsf{sdec}(\mathcal{H}(u_1)\sigma, \mathcal{H}(u_2)\sigma))$. Because $\mathsf{tests}^d(\mathcal{H}(u)\sigma)$ passes, it must be the case that

$$\mathsf{tag}_d(\mathsf{untag}_d(\mathsf{sdec}(\mathcal{H}(u_1)\sigma, \mathcal{H}(u_2)\sigma))) =_{E^0} \mathsf{sdec}(\mathcal{H}(u_1)\sigma, \mathcal{H}(u_2)\sigma).$$

The preceding equality only holds when

$$\mathcal{H}(u_1)\sigma =_{E^0} \mathsf{senc}(\mathsf{tag}_d(v), \mathcal{H}(u_2)\sigma) \qquad (3)$$

for some term v. Therefore, by the definition of $\lfloor \_ \rfloor$, we have

$$\begin{aligned} \lfloor \mathcal{H}(u_1)\sigma \rfloor &=_{E^0} \mathsf{senc}_d(\mathsf{untag}_{d'}(\mathsf{tag}_{d'}(v)), \lfloor \mathcal{H}(u_2)\sigma \rfloor) \\ &=_{E^0} \mathsf{senc}_d(\lfloor v\sigma \rfloor, \lfloor \mathcal{H}(u_2)\sigma \rfloor) \end{aligned}$$

from which it follows that $\mathsf{senc}_d(\mathsf{sdec}_d(\lfloor \mathcal{H}(u_1)\sigma \rfloor, \lfloor \mathcal{H}(u_2)\sigma \rfloor), \lfloor \mathcal{H}(u_2)\sigma \rfloor)$

$$\begin{aligned}
&=_{E^0} \mathsf{senc}_d(\mathsf{sdec}_d(\mathsf{senc}_d(v, \lfloor \mathcal{H}(u_2)\sigma \rfloor), \lfloor \mathcal{H}(u_2)\sigma \rfloor), \lfloor \mathcal{H}(u_2)\sigma \rfloor)\\
&=_{E^0} \mathsf{senc}_d(\lfloor v\sigma \rfloor, \lfloor \mathcal{H}(u_2)\sigma \rfloor)\\
&=_{E^0} \lfloor \mathcal{H}(u_1)\sigma \rfloor
\end{aligned}$$

The result is consequence of the following.

$$\begin{aligned}
\lfloor \mathcal{H}(u)\sigma \rfloor &= \lfloor \mathsf{untag}_d(\mathsf{sdec}(\mathcal{H}(u_1)\sigma, \mathcal{H}(u_2)\sigma)) \rfloor\\
&= \mathsf{untag}_{d'}(\mathsf{tag}_{d'}(\mathsf{sdec}_d(\lfloor \mathcal{H}(u_1)\sigma \rfloor, \lfloor \mathcal{H}(u_2)\sigma \rfloor)))\\
&=_{E^0} \mathsf{sdec}_d(\lfloor \mathcal{H}(u_1)\sigma \rfloor, \lfloor \mathcal{H}(u_2)\sigma \rfloor)\\
&=_{E^0} \mathsf{sdec}_d(u_1\lfloor \sigma \rfloor, u_2\lfloor \sigma \rfloor) && \text{(I.H.)}\\
&= \mathsf{sdec}_d(u_1, u_2)\lfloor \sigma \rfloor
\end{aligned}$$

□

Define a function Θ on traces inductively as follows. If $|t| = 0$ then $\Theta(t) = t$. Otherwise $t = t' \xrightarrow{(\S,[l])} \mathsf{obs}(P, \varphi, \sigma)$. Define

$$\Theta(t) = \Theta(t') \xrightarrow{\tau,[l]} \mathsf{last}(\Theta(t')) \xrightarrow{(\Diamond_\varphi(\S),[l])} \mathsf{obs}(P', \varphi', \sigma')$$

where $\Diamond_\varphi(\tau) = \tau$ if $\S = \tau$ and otherwise $\Diamond_\varphi(r)$ if $\S$ is a recipe r. Let P be a process over $\mathcal{F}^b_{\mathsf{senc}} \cup \mathcal{F}^c_{\mathsf{senc}}$. Define a function $\Diamond_P : \mathsf{A} \to \mathsf{A}$ as follows. If $\mathcal{A}$ is not an adversary for $\lceil P \rceil$ then $\Diamond_P(\mathcal{A})$ is undefined. Otherwise

$$\Diamond_P(\mathcal{A})(t) = \begin{cases} \mathcal{A}(\Theta(t)) & \text{if } \Theta(t) \in \mathsf{Trace}([\![\lceil P \rceil^{\mathcal{A}}]\!]) \\ \text{undefined} & \text{otherwise} \end{cases}$$

For substitutions σ, σ' and an equational theory E, we write $\sigma \cong_E \sigma'$ if $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $x\sigma =_E x\sigma'$ for all $x \in \mathsf{dom}(\sigma)$.

Lemma 27. *Let P, Q be linear processes over $\mathcal{F}_{\mathsf{senc}}$ and W be an arbitrary interleaving of P^b and Q^c. If*

$$t = \mathsf{obs}(\lceil W \rceil, \emptyset, \emptyset) \xrightarrow{(\tau,[l_1])}\xrightarrow{\alpha_1} \ldots \xrightarrow{(\tau,[l_k])}\xrightarrow{\alpha_k} \mathsf{obs}(\lceil W_k \rceil, \varphi_k, \sigma_k)$$

is a trace of $[\![\lceil W \rceil^{\mathcal{A}}]\!]$ then

$$t' = \mathsf{obs}(W, \emptyset, \emptyset) \xrightarrow{\alpha'_1} \ldots \xrightarrow{\alpha'_k} \mathsf{obs}(W'_k, \varphi'_k, \sigma'_k)$$

is a trace of $[\![W^{\Diamond_W(\mathcal{A})}]\!]$ such that $\varphi'_k \equiv_{E^0} \lfloor \varphi_k \rfloor$ and $\sigma'_k \equiv_{E^0} \lfloor \sigma_k \rfloor$.

Proof. The proof is by induction on k. The base case, $k = 0$, is trivial. For the induction step, let $t' = t'' \xrightarrow{\alpha'_k} \mathsf{obs}(W', \varphi'_k, \sigma'_k)$ where $\mathsf{last}(t'') = \mathsf{obs}(W'', \varphi'_{k-1}, \sigma'_{k-1})$. By the I.H. t'' is a trace of $[\![W^{\Diamond_W(\mathcal{A})}]\!]$ such that $\varphi'_{k-1} \equiv_{E^0} \lfloor \varphi_{k-1} \rfloor$ and $\sigma'_{k-1} \equiv_{E^0} \lfloor \sigma_{k-1} \rfloor$. We proceed by cases.

case 1: α_{2k} is an input action of the form $\mathtt{in}(x)$. By definition, α'_k is also an input action of the form $\mathtt{in}(x)$ and clearly t' is a trace of $[\![W^{\Diamond(\mathcal{A})}]\!]$. We have $\varphi_k = \varphi_{k-1}$ and $\varphi'_k = \varphi'_{k-1}$ and therefore $\varphi'_k = \varphi'_{k-1} \equiv_{E_0} \lfloor \varphi_{k-1} \rfloor = \lfloor \varphi_k \rfloor$. By definition $\sigma_k = \sigma_{k-1} \cup \{x \mapsto r\varphi_{k-1}\}$ and $\sigma'_k = \sigma'_{k-1} \cup \{x \mapsto \Diamond_{\varphi_{k-1}}(r)\varphi'_{k-1}\}$. By the I.H. it suffices to show $x\sigma'_k =_{E^0} \lfloor x\sigma_k \rfloor$, which is a consequence of the derivation below.

$$\begin{aligned}\lfloor x\sigma_k \rfloor &=_{E^0} \lfloor r\varphi_{k-1} \rfloor && \text{(Lemma 22)}\\ &=_{E^0} \Diamond_{\varphi_{k-1}}(r)\lfloor \varphi_{k-1} \rfloor && \text{(Lemma 23)}\\ &=_{E^0} \Diamond_{\varphi_{k-1}}(r)\varphi'_{k-1} && \text{(I.H.)}\\ &= \quad x\sigma'_k\end{aligned}$$

case 2: α_{2k} corresponds to executing an assignment of the form $x := \mathcal{H}(u)$ for some $u \in \mathcal{T}(\mathcal{F}^d_{\mathsf{senc}}, \mathcal{X})$. By definition α'_k is also an assignment of the form $x := u$ and clearly t' is a trace of $[\![W^{\Diamond(\mathcal{A})}]\!]$. Because t is a trace of $[\![\lceil W \rceil^{\mathcal{A}}]\!]$, it must be the case that $\mathsf{tests}^d(\mathcal{H}(u))$ passes. We also have $\varphi_k = \varphi_{k-1}$ and $\varphi'_k = \varphi'_{k-1}$ and therefore $\varphi'_k = \varphi'_{k-1} \equiv_{E^0} \lfloor \varphi_{k-1} \rfloor = \lfloor \varphi_k \rfloor$. By definition $\sigma_k = \sigma_{k-1} \cup \{x \mapsto \mathcal{H}(u)\sigma_{k-1}\}$ and $\sigma'_k = \sigma'_{k-1} \cup \{x \mapsto u\sigma'_{k-1}\}$. By the I.H. it suffices to show $x\sigma'_k =_{E_0} \lfloor x\sigma_k \rfloor$, which is a consequence of the derivation below.

$$\begin{aligned}\lfloor x\sigma_k \rfloor &=_{E^0} \lfloor \mathcal{H}(u)\sigma_{k-1} \rfloor && \text{(Lemma 22)}\\ &=_{E^0} u\lfloor \sigma_{k-1} \rfloor && \text{(Lemma 26)}\\ &=_{E^0} u\sigma'_{k-1} && \text{(I.H.)}\\ &=_{E^0} x\sigma'_k\end{aligned}$$

case 3: α_{2k} corresponds to executing a test of the form $[\mathcal{H}(u_1) = \mathcal{H}(u_2)]$ where $u_1, u_2 \in \mathcal{T}(\mathcal{F}^d_{\mathsf{senc}}, \mathcal{X})$. By definition α'_k is also a test of the form $[u_1 = u_2]$. Because t is a trace of $[\![\lceil W \rceil^{\mathcal{A}}]\!]$, it must be the case that $\mathsf{tests}^d(\mathcal{H}(u_1))$ and $\mathsf{tests}^d(\mathcal{H}(u_1))$ both pass. We also have $\varphi_k = \varphi_{k-1}$ and $\varphi'_k = \varphi'_{k-1}$ and therefore $\varphi'_k = \varphi'_{k-1} \equiv_{E^0} \lfloor \varphi_{k-1} \rfloor = \lfloor \varphi_k \rfloor$. We can similarly conclude that $\sigma'_k \equiv_{E^0} \lfloor \sigma_k \rfloor$. We know that $\mathcal{H}(u_1)\sigma_k =_E \mathcal{H}(u_2)\sigma_k$, from which we have the following derivation.

$$\begin{aligned}\lfloor \mathcal{H}(u_1)\sigma_k \rfloor &=_{E^0} \lfloor \mathcal{H}(u_2)\sigma_k \rfloor && \text{(Lemma 22)}\\ u_1\lfloor \sigma_k \rfloor &=_{E^0} u_2\lfloor \sigma_k \rfloor && \text{(Lemma 26)}\\ u_1\sigma'_k &=_{E^0} u_2\sigma'_k\end{aligned}$$

That is, t' is a trace of $[\![W^{\Diamond_W(\mathcal{A})}]\!]$

case 4: α_{2k} is an output of the form $\mathtt{out}(\mathcal{H}(u))$ where $u \in \mathcal{T}(\mathcal{F}^d_{\mathsf{senc}}, \mathcal{X})$. By definition α'_k is also an output of the form $\mathtt{out}(u)$. Because t is a trace of $[\![\lceil W \rceil^{\mathcal{A}}]\!]$, it must be the case that $\mathsf{tests}^d(\mathcal{H}(u))$ passes. We also have $\sigma_k = \sigma_{k-1}$ and $\sigma'_k = \sigma'_{k-1}$ and therefore $\sigma'_k = \sigma'_{k-1} \equiv_{E^0} \lfloor \sigma_{k-1} \rfloor = \lfloor \sigma_k \rfloor$. By definition $\varphi_k = \varphi_{k-1} \cup \{w \mapsto \mathcal{H}(u)\sigma_k\}$ and $\varphi'_k = \varphi'_{k-1} \cup \{w \mapsto u\sigma'_k\}$. By the I.H. it suffices to show $w\varphi'_k =_{E_0} \lfloor w\varphi_k \rfloor$, which is a consequence of the derivation below.

$$\begin{aligned}\lfloor w\varphi_k \rfloor &=_{E^0} \lfloor \mathcal{H}(u)\sigma_k \rfloor && \text{(Lemma 22)}\\ &=_{E^0} u\lfloor \sigma_k \rfloor && \text{(Lemma 26)}\\ &=_{E^0} u\sigma'_k\\ &= \quad w\varphi'_k\end{aligned}$$

case 5: α_{2k} corresponds to a new name creation of the form νn. This case is straightforward from the fact that α'_k is also a new name creation of the form νn and $\lfloor n \rfloor = n$.

Let $C[\Box_1, ..., \Box_n] = \nu k_1 \cdot ... \cdot \nu k_m \cdot (D_1[\Box_1]|...|D_n[\Box_n])$ (resp. $C'[\Box_1, ..., \Box_n] = \nu k'_1 \cdot ... \cdot \nu k'_m \cdot (D'_1[\Box_1]|...|D'_n[\Box_n])$) be a context over $\mathcal{F}_{\mathsf{senc}}$ with labels from $\mathcal{L}_c$.

Further let $B_1, ..., B_n$ (resp. $B'_1, ..., B'_n$) be basic processes over $\mathcal{F}_{\mathsf{senc}}$ with labels from $\mathcal{L}_b$. We will write ‡ when the conditions of Theorem 2 hold for the preceding processes. The following is a consequence of Lemmas 24, 25 and 27.

Proposition 4. *Assuming ‡, then*

$$(\lceil C^c[B_1^b, ..., B_n^b]\rceil,\ \lceil (C')^c[(B'_1)^b, ..., (B'_n)^b]\rceil)$$

is transposable to

$$(C^c[B_1^b, ..., B_n^b],\ (C')^c[(B'_1)^b, ..., (B'_n)^b]).$$