Modular Verification of Protocol Equivalence in the Presence of Randomness

Matthew S. Bauer[1], Rohit Chadha[2], and 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 [12, 34, 8].

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 [20], onion routers [37] and Tor [32]. Randomization is also used in cryptographic protocols to achieve fair exchange [11, 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, 6].

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 [21, 14, 36, 18, 16]). In order to circumvent this problem, many authors [29, 21, 14, 36, 18, 16, 10, 17] 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 [10, 17] trace-indistinguishability of applied-pi calculus processes [2]. 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 [17], 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 [17], 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 11) 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, 5] 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 5 on Page 13). 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, 5], 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 3 on Page 11). 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 14). 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 16). 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, 3, 24, 26, 7, 31, 22, 27, 47, 5] and indistinguishability properties in [4, 5]. A recent work [10] 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, 15]. Our result is most closely related to [4, 5, 10].

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 [19] was carried out the in CSP framework [43]. In all of these works, the messages constructed by the attacker is assumed to be bounded. [17] 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, 23, 9, 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 Protocols

In this section we introduce our process algebra for modeling security protocols with coin tosses. Our formalism can be seen as an extension of the one from [4]. Similar to [38], it extends the applied π-calculus by the inclusion of a new operator for probabilistic choice. Like [4], the calculus doesn't include else branches and considers a single public channel. We will assume the reader is familiar with the models of Markov chains (DTMC) and partially observable Markov decision process (POMDP); for completeness these definitions can be found in the full version of this paper [1].

2.1 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. $\mathsf{dom}(\sigma)$ 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 will 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 $\varphi \vdash^r_E u$, 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 $(\mathcal{F}, E) = (\mathcal{F}_b \cup \mathcal{F}_c, E_b \cup E_c)$.

2.2 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

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

Basic Contexts

$$D[\Box] ::= \Box \mid B \mid D[\Box] \cdot B \mid B \cdot D[\Box] \mid D[\Box] +_p^l D[\Box]$$

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. We restrict our attention to processes that do not contain variables with both free and bound occurrences. That is, for a process Q, $\mathsf{fv}(Q) \cap \mathsf{bv}(Q) = \emptyset$. We now give an examples illustrating the type of protocols that can be modeled and analyzed in our process algebra.

Example 1. 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.

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

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 $C[(m_A := b) \cdot A, (m_B := b') \cdot B]$.

2.3 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 (partially observable Markov decision process) $(Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$, where Z is the set of all extended processes, z_s is the start state $(P, \emptyset, \emptyset)$, Act is the set of actions (pairs containing a recipe and an equivalence classes on labels), Δ is a probabilistic transition relation describing how a process evolves, $\mathcal{O}$ is a countable set of observations used to model information available to the attacker and obs is a labeling of states with observations. Informally, a process evolves as follows. After i execution steps, if the process is in state z, the attacker chooses an action α, which together with the state z defines a unique probability distribution μ given by the transition relation Δ. The process then moves to state z' in the $(i+1)$-st step with probability $\mu(z')$. The only constraint on the choice of the action α is that the same action must be chosen in all executions which are indistinguishable to the attacker. We give the formal definitions below.

The set of all actions is $\mathsf{Act} = (\mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X}_w) \cup \{\tau\}, \mathcal{L}/\sim)$. In Figure 2, we define Δ which maps an extend process and an action to a distribution on Z. There we write $(P, \varphi, \sigma) \xrightarrow{\alpha} \mu$ if $\Delta((P, \varphi, \sigma), \alpha) = \mu$. If Figure 2, $\mu \cdot Q$ denotes 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 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. An *execution* ρ of a process P is a finite sequence $z_0 \xrightarrow{\alpha_1} z_1 \cdots \xrightarrow{\alpha_m} z_m$ such that $z_0, z_1, ..., z_m \in Z$, $z_0 = z_s$ and for each $i \geq 0$, $z_i \xrightarrow{\alpha_{i+1}} \mu_{i+1}$ and $\mu_{i+1}(z_{i+1}) > 0$. The probability of the execution ρ of P, denoted $\mathsf{prob}(\rho, P)$, is $\mu_1(z_1) \times ... \times \mu_m(z_m)$.

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. For a frame φ, we write $[\varphi]$ to denote the equivalence class of φ with respect to E, 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 $\eta = (P, \varphi, \sigma)$, $\mathsf{obs}(\eta) = (\mathsf{enabled}(\eta), [\varphi])$. For an execution $\rho = z_0 \xrightarrow{\alpha_1} z_1 \cdots \xrightarrow{\alpha_m} z_m$ we write $\mathsf{tr}(\rho)$ to represent the *trace* of ρ, defined as the sequence $\mathsf{obs}(z_0) \xrightarrow{\alpha_1} \mathsf{obs}(z_1) \cdots \xrightarrow{\alpha_m} \mathsf{obs}(z_m)$. The set of all traces of a process P is denoted $\mathsf{Trace}(P)$. A trace models the view of the attacker for a particular execution. An attacker for a process P is a partial function $\mathcal{A} : \mathsf{Trace}(P) \hookrightarrow \mathsf{Act}$. An attacker resolves all non-determinism, and when a process P is executed with respect to a fixed attacker $\mathcal{A}$, the evolution of the process P can be described by a DTMC $P^{\mathcal{A}}$. For process P, adversary $\mathcal{A}$ and trace t, let $\rho_1, ..., \rho_k$ be the executions of $P^{\mathcal{A}}$ such that $\mathsf{tr}(\rho_i) = t$ for all $i \in \{1, ..., k\}$. We will write $\mathsf{prob}(t, P^{\mathcal{A}})$ to denote $\sum_{i=1}^{k} \mathsf{prob}(t, P^{\mathcal{A}})$.

An extended process (Q, φ, σ) over the equational theory $(\mathcal{F}, E)$ *preserves* the secrecy of $x \in \mathsf{vars}(Q)$, written $(Q, \varphi, \sigma) \models_E x$, if there is no $r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathsf{dom}(\varphi))$ such that $\varphi \vdash^r_E x\sigma$. A process P is said to keep variables $x_1, ..., x_n$

secret with probability 1, denoted $P \models_{E,1} \mathsf{secret}(x_1, ..., x_n)$, if there is no execution of P containing a state z such that $z \not\models_E x$ for some $x \in \{x_1, ..., x_n\}$. Two processes P_0 and P_1 over the same set of actions and observations are said to be trace equivalent, denoted $P_0 \approx P_1$, if for every attacker $\mathcal{A}$ and trace $t \in \mathsf{Trace}(P_0) \cup \mathsf{Trace}(P_1)$, $\mathsf{prob}(t, P_0^{\mathcal{A}}) = \mathsf{prob}(t, P_1^{\mathcal{A}})$. Observe that for a protocol P not containing coin tosses, any two executions of P are distinguishable. Furthermore, for each attacker $\mathcal{A}$, there is only one execution of protocol P. Thus, it follows that our notion of trace equivalence coincides with the notion of trace-equivalence of for applied pi-calculus. We conclude this section by showing how the notion of trace equivalence can capture privacy properties of the DC-net protocol described earlier in this section.

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

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

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

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

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

Fig. 2: Process semantics

Example 2. Consider the DC-net protocol defined in Example 1 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.

3 Compositional equivalence of single session protocols

3.1 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.

3.2 Disjoint Primitives

In our composition result, we would like to argue that if two contexts $C[\Box]$ and $C'[\Box]$ 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 our main composition result is as follows.

Theorem 1. *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* $\mathcal{F}_c$ *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 $\mathsf{prob}(t, C[B_1, ..., B_n]^{\mathcal{A}}) \neq \mathsf{prob}(t, C'[B'_1, ..., B'_n]^{\mathcal{A}})$, 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, 4] 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 3 below.

Example 3. 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 in-equivalence 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 $\mathcal{A}'$ 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 the full version [1].

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

Example 4. Let A, B be the protocols for Alice and Bob from the DC-net protocol given in Example 1. 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. $f(x, y) = x^y \bmod p$, $g(y) = \alpha^y \bmod p$ 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.

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

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 [10].

3.3 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* instantiation of B_0. A natural selection for B_0 is the empty process $[\top]$. We illustrate why such a choice is insufficient in Example 5.

Example 5. 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 5 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. 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 in-equivalent 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 [10], 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 6 below.

Example 6. Consider the contexts

$$C[\Box] = \nu k_1 \cdot (x_1 := k_1) \cdot \Box \cdot \mathtt{in}(y) \cdot \nu k_2 \cdot [y = x_1] \cdot \mathtt{out}(ok)$$
$$C'[\Box] = \nu k_1 \cdot (x_1 := k_1) \cdot \Box \cdot \mathtt{in}(y) \cdot \nu k_2 \cdot [y = k_2] \cdot \mathtt{out}(ok)$$

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 7, the composition theorem does not hold when this restriction is relaxed.

Example 7. Consider the context and processes below.

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

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 $C[B_1] \not\approx C[B_2]$.

3.4 Shared primitives through tagging

Theorem 1 requires that the context and basic processes don't share cryptographic primitives. 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 [4, 27], can be found in Appendix B. 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. $C[\mathsf{out}(\sharp(x_1))^{l_1}, ..., \mathsf{out}(\sharp(x_n))^{l_n}] \approx C'[\mathsf{out}(\sharp(x_1))^{l_1}, ..., \mathsf{out}(\sharp(x_n))^{l_n}]$
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 $\lceil C^c[B^b_1, ..., B^b_n] \rceil \approx \lceil (C')^c[(B'_1)^b, ..., (B'_n)^b] \rceil$.

4 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).

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

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* $C[\Box_1, ..., \Box_n] = \nu k_1 \cdot ... \cdot \nu k_m . !\nu\lambda \cdot (D_1[\Box_1]|...|D_n[\Box_n])$ *(resp.* $C'[\Box_1, ..., \Box_n] = \nu k_1' \cdot ... \cdot \nu k_m' . !\nu\lambda \cdot (D_1'[\Box_1]|...|D_n'[\Box_n])$*) 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. $\lambda \notin \mathsf{vars}(C[B_1, ..., B_n]) \cup \mathsf{vars}(C'[B_1', ..., B_n'])$
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 $\lceil \nu k_1 \cdot ... \cdot \nu k_m . !\nu\lambda \cdot (D_1^{(c,\lambda)}[B_1^{(b,\lambda)}] \mid ... \mid D_n^{(c,\lambda)}[B_n^{(b,\lambda)}]) \rceil \approx \lceil \nu k_1' \cdot ... \cdot \nu k_m' . !\nu\lambda \cdot ((D_1')^{(c,\lambda)}[(B_1')^{(b,\lambda)}] \mid ... \mid (D_n')^{(c,\lambda)}[(B_n')^{(b,\lambda)}]) \rceil$.

5 Conclusions and Future Work

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. `https://uofi.box.com/s/r1jwzvwqth4mq58z8ghcq1vhf1e7iz05`.
2. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. *ACM SIGPLAN Notices*, 36(3):104–115, 2001.

3. S. Andova, C. J. F. Cremers, K. Gjøsteen, S. Mauw, S. F. Mjølsnes, and S. Radomirovic. A framework for compositional verification of security protocols. *Information and Computation*, 206(2-4):425–459, 2008.
4. M. Arapinis, V. Cheval, and S. Delaune. Verifying privacy-type properties in a modular way. In *25th IEEE Computer Security Foundations Symposium (CSF'12)*, pages 95–109, 2012.
5. M. Arapinis, V. Cheval, and S. 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.
6. M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *2010 23rd IEEE Computer Security Foundations Symposium*, pages 107–121, 2010.
7. M. Arapinis, S. Delaune, and S. 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.
8. A. Armando, W. Arsac, T. Avanesov, M. Barletta, A. Calvi, A. Cappai, R. Carbone, Y. Chevalier, L. Compagna, J. 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.
9. M. 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.
10. M. S. Bauer, R. Chadha, and M. 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.
11. M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.
12. B. Blanchet, M. Abadi, and C. 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.
13. R. 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. 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.
15. R. Canetti and J. 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.
16. R. Chadha, A. 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. 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.
18. K. Chatzikokolakis and C. Palamidessi. Making Random Choices Invisible to the Scheduler. *Information and Computation*, 2010, to appear.
19. D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.

20. D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.
21. L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. PhD thesis, Radboud University of Nijmegen, 2006.
22. C. Chevalier, S. Delaune, and S. Kremer. Transforming password protocols to compose. In *31st Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 204–216, 2011.
23. Y. Chevalier and M. Rusinowitch. Decidability of equivalence of symbolic derivations. *Journal of Automated Reasoning*, 2010. To appear.
24. V. Cortier, J. Delaitre, and S. 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. V. Cortier and S. Delaune. A method for proving observational equivalence. In *Proc. 22nd IEEE Computer Security Foundations Symposium (CSF'09)*, pages 266–276, 2009.
26. V. Cortier and S. Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, 2009.
27. Ştefan Ciobâcă and V. 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. A. Datta, A. Derek, J. C. Mitchell, and D. 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. S. Delaune, S. Kremer, and M. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.
31. S. Delaune, S. Kremer, and M. D. Ryan. Composition of password-based protocols. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF'08)*, pages 239–251, 2008.
32. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.
33. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.
34. S. Escobar, C. Meadows, and J. Meseguer. *Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties*, pages 1–50. 2009.
35. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.
36. F. Garcia, P. van Rossum, and A. Sokolova. Probabilistic Anonymity and Admissible Schedulers. *CoRR*, abs/0706.1019, 2007.
37. D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding routing information. In *Proceedings of the First International Workshop on Information Hiding*, pages 137–150, 1996.
38. J. Goubault-Larrecq, C. Palamidessi, and A. Troina. A probabilistic applied pi–calculus. In *Asian Symposium on Programming Languages and Systems*, pages 175–190, 2007.
39. C. A. Gunter, S. Khanna, K. Tan, and S. S. Venkatesh. Dos protection for reliably authenticated broadcast. In *NDSS*, 2004.
40. J. D. Guttman. Authentication tests and disjoint encryption: A design method for security protocols. *Journal of Computer Security*, 12(3-4):409–433, 2004.

41. J. 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. C. He, M. Sundararajan, A. Datta, A. Derek, and J. 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. C. A. R. Hoare. *Communicating sequential processes*, volume 178. 1985.
44. M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic symbolic model checker. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 200–204, 2002.
45. C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE journal on selected areas in communications*, 21(1):44–54, 2003.
46. C. Meadows. *Emerging Issues and Trends in Formal Methods in Cryptographic Protocol Analysis: Twelve Years Later*, pages 475–492. 2015.
47. S. Mödersheim and L. 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. B. Pfitzmann and A. 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. M. K. Reiter and A. 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. S. Schneider and A. Sidiropoulos. Csp and anonymity. In *European Symposium on Research in Computer Security*, pages 198–218, 1996.
53. V. Shmatikov. Probabilistic analysis of an anonymity system. *Journal of Computer Security*, 12(3, 4):355–377, 2004.

A Bound variables

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. We define these formally below.

- If $B = \nu x$ then $\mathsf{bv}(B) = \mathsf{abv}(B) = \{x\}$ and $\mathsf{fv}(B) = \emptyset$
- If $B = (x := u)$ then $\mathsf{bv}(B) = \mathsf{abv}(B) = \{x\}$ and $\mathsf{fv}(B) = \mathsf{vars}(u)$
- If $B = [c_1 \wedge ... \wedge c_k]$ then $\mathsf{bv}(B) = \mathsf{abv}(B) = \emptyset$ and $\mathsf{fv}(B) = \mathsf{vars}(c_1) \cup ... \cup \mathsf{vars}(c_k)$
- If $B = \mathtt{in}(x)$ then $\mathsf{bv}(B) = \mathsf{abv}(B) = \{x\}$ and $\mathsf{fv}(B) = \emptyset$
- If $B = \mathtt{out}(u)$ then $\mathsf{bv}(B) = \mathsf{abv}(B) = \emptyset$ and $\mathsf{fv}(B) = \mathsf{vars}(u)$
- If $B = B_1 \cdot B_2$ then $\mathsf{bv}(B_1) \cup \mathsf{bv}(B_2)$, $\mathsf{abv}(B_1) \cap \mathsf{abv}(B_2)$ and $\mathsf{fv}(B_1) \cup \mathsf{fv}(B_2)$
- If $B = B_1 +_p B_2$ then $\mathsf{bv}(B_1) \cup \mathsf{bv}(B_2)$, $\mathsf{abv}(B_1) \cup \mathsf{abv}(B_2)$ and $\mathsf{fv}(B_1) \cup (\mathsf{fv}(B_2) \setminus \mathsf{abv}(B_1))$

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.

B Tagging

We 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:

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

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:

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

Definition 2. *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:*

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

Definition 2 can be lifted naturally to basic contexts by requiring $\lceil \Box \rceil = \Box$ for any process variable $\Box$.

Definition 3. *Let $C^d = a_1 \cdot ... \cdot a_n \cdot (D_1[\Box_1]|...|D_n(\Box_n))$ be a context over $\mathcal{F}^d_{\mathsf{senc}}$ for $d \in \{b, c\}$. The context $\lceil C^d \rceil$ is $\lceil a_1 \cdot ... \cdot a_n \rceil \cdot (\lceil D_1[\Box_1] \rceil|...|\lceil D_n(\Box_n) \rceil)$.*