Composing protocols with randomized actions

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

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

Abstract. Recently, several composition results have been established, showing that two cryptographic protocols proven secure against a Dolev-Yao attacker continue to afford the same security guarantees when composed together, provided the protocol messages are tagged with the information of which protocol they belong to. The key technical tool used to establish this guarantee is a separation result which shows that any attack on the composition can be mapped to an attack on one of the composed protocols running in isolation. We consider the composition of protocols which, in addition to using cryptographic primitives, also employ randomization within the protocol to achieve their goals. We show that if the protocols never reveal a secret with a probability greater than a given threshold, then neither does their composition, given that protocol messages are tagged with the information of which protocol they belong to.

1 Introduction

The design of correct cryptographic protocols is a highly non-trivial task, and security flaws are often subtle. Attacks on many protocols that were previously "proved" secure by hand, have been discovered. One approach that improves the confidence in the correctness of security protocols is formal analysis. In order to make the analysis amenable to automation, usually the assumption of perfect cryptography is made. In this "Dolev-Yao" framework, protocol messages are symbolic terms identified modulo an equational theory (and not bit-strings) that model cryptographic operations. Security is then proven in the presence of an omnipotent attacker that can read all messages sent on public channels by protocol participants, remember the (potentially unbounded) communication history, and (nondeterministically) inject new messages in the network addressed to particular participants while remaining anonymous. This Dolev-Yao model has shown to be very successful in identifying security flaws.

Cryptographic protocols are often proven secure in isolation. In practice, however, they may be executing concurrently or sequentially, in a modular fashion, with other protocols. For example, a number of security protocols involve a sub-protocol in which short-term secret keys are exchanged. While analyzing such protocols, often the sub-protocol is abstracted away by assuming that

⋆ Partially supported by grant NSF CNS 1314485
⋆⋆ Partially supported by grant NSF CNS 1314338

the protocol participants have successfully shared secrets. However, two cryptographic protocols proven secure independently may not remain secure if they are executed compositionally. The central problem is that these protocols may share some secret data, as in the key exchange situation described above.

Hence, a number of recent papers have identified sufficient conditions under which such protocol compositions can be proven secure — safety properties are considered in [20, 30, 28, 29, 2, 17, 18, 5, 22, 16, 19, 31, 3] and indistinguishability properties in [4, 3], while [20, 30, 28] provide a general framework for proving that protocols compose securely. Other papers [2, 29] essentially show that protocol compositions are secure if messages from one protocol cannot be confused with messages from another protocol. [11] shows that this continues to be the case even when dishonest participants do not tag their messages properly. This can be ensured if certain protocol transformations are made (see for example [17, 18, 22, 4]). Essentially, these protocol transformations require that all protocol messages are *tagged* with the protocol name and protocol instance to which they belong. The exact choice of tagging scheme depends on the desired security property; incorrect tagging can actually make a secure protocol insecure [22]. In the computational model, the problem of composing protocols securely has been studied in [9, 10].

The focus of this paper is to extend this work on secure protocol composition to protocols that employ randomization. Randomization plays a key role in the design of algorithmic solutions to problems arising in security. For example, randomization is essential in implementing cryptographic primitives such as encryption and key generation. Randomization is also used in cryptographic protocols to achieve security guarantees such as fair exchange (see [7, 23]), anonymity (see [14, 32, 25]), voter privacy in electronic voting (see [33]) and denial of service prevention (see [27]).

We study the problem of when the composition of a (randomized) sub-protocol P followed by (randomized) sub-protocol Q is secure. For non-randomized protocols, this problem was studied in [19]. Our composition framework generalizes that of [19] to handle sequential, parallel and a form of vertical composition while extending to randomized protocols. They show that if one can prove P and Q do not reveal shared secrets when run in isolation (in that case Q is assumed to generate fresh secret keys), then the sequential composition of P and Q does not reveal any secret of Q if the protocol messages are tagged with the information of which protocol they belong to. The key technical tool used to establish this guarantee is a separation result which shows that any attack on the composition can be mapped to an attack on one of P or Q. This is achieved by first showing that, as the protocol messages are tagged, messages from one protocol cannot be confused with the messages of the other protocol. Then an attack trace can be simply separated into traces of P and Q. We study the same problem for the case when P and Q are randomized protocols. Our composition framework generalizes that of [19] to handle sequential, parallel and a form of vertical composition while extending to randomized protocols. The protocols themselves are expressed in a variant of the probabilistic applied-pi calculus [26] which is

an extension of applied pi-calculus [1]. The Probabilistic applied-pi calculus is a convenient formalism to describe and analyze randomized security protocols in the Dolev-Yao model.

Contributions: Our first composition result is for the composition of one session of P and one session of Q. We show that if P (in isolation) is secure with probability at least p (i.e., the shared secrets are not leaked) and Q is secure with probability at least q, then the composed protocol is secure with probability at least pq, provided the protocol messages are tagged with the information of the protocol to which they belong. Although we exploit some techniques used in [19] to establish this result, there are important differences. First, the separation result in [19] does not carry over to the randomized setting. This is because an attack on the composition of P and Q is no longer a trace, but is instead a tree, as the protocol itself makes random choices. As a consequence, in different branches representing different resolutions of the randomized coin tosses, it is possible that the attacker may choose to send different messages (See Example 4 on Page 12). In such a case, an attack on the composition of P and Q cannot be separated into an attack on P and an attack on Q. Instead, we show that if there is an attack on the composition of P and Q then either we can extract an attack on P which succeeds with probability $> p$ or there is an attack on Q which succeeds with probability $> q$.

Another challenge manifested in the context of randomized protocols is that one must consider adversaries whose actions do not depend on the result of private coin tosses made by protocol participants, as observed in [21, 15, 8, 24, 13, 12]. In order to faithfully model the privacy of coin tosses, we mandate that an attacker always take the same actions in any two different probabilistic branches in a run of a protocol if its views of the protocol run in the two branches is exactly the same. This restriction is adopted from [21, 15, 8, 24, 13, 12], and is the first formalization of this concept within the applied-pi calculus. As demonstrated in Example 3 on Page 11, this class of attackers allows privacy guarantees, typically modeled as indistinguishability properties, to instead be modeled as reachablility properties. Considering a more restricted class of attackers imposes additional challenges in our setting, as membership in this sub-class of attackers must be maintained when mapping attack traces on composed protocols to attacks traces on the individual protocols constituting the composition.

Our second composition result concerns multiple sessions of the composed protocol. Here, we would like to show that if n sessions of P are secure with probability at least p and n sessions of Q are secure with probability at least q then n sessions of the composed protocol are secure with probability at least pq, provided the protocol messages are tagged with the information of which protocol they belong to. Indeed, a similar result is claimed in [19] for the non-randomized protocols. Unfortunately, this result is not valid even for nonrandomized protocols and we exhibit a simple example which contradicts this desired result (See Example 6 on Page 17). Essentially, the reason for this failure is that, in the claimed result, the n sessions of Q are assumed to generate fresh shared secrets in every session; but P may not be guaranteeing this freshness. Thus, messages

of one session can get confused with messages of other sessions. We establish a weaker composition result in which we assume that the messages of each session of Q are tagged with a *unique* session identifier in addition to the protocol identifier. The use of session identifiers ensures that the messages of one session cannot be confused with other sessions.

Finally, we also consider the case for protocols containing an unbounded number of sessions. For this case, we observe that a composition result is only possible when P and Q are secure with probability exactly 1. This is because if m sessions of a protocol leak a secret with probability $r > 0$ then by running mk sessions we can amplify the probability of leaking the secret. This probability approaches 1 as we increase k. We show that if an unbounded number of sessions of P are secure with probability 1 and an unbounded number of sessions of Q are secure with probability 1 then an unbounded number of sessions of the composed protocol are secure with probability 1, if the protocol messages are tagged with the information of which protocol they belong to and the messages of each session of Q are tagged with a *unique* session identifier.

The paper is organized as follows. In Section 2 we give relevant background information. Section 3 presents our processes algebra for randomized protocols and Section 4 gives our main composition results. Section 5 shows how this result can be extended to protocols with multiple sessions.

2 Preliminaries

We will start by discussing some standard notions from probability theory, Markov Chains and Markov Decision Processes. A process algebra for modeling security protocols with coin tosses will then be presented in Section 3. This process algebra closely resembles that of [26], which extends the applied π-calculus by the inclusion of a new operator for probabilistic choice. Following [19], our process calculus will also include several limitations necessary to achieve our results. In particular, conditionals no longer include else branches and we consider only a single public channel.

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 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 $\{z' \,|\, \Delta(z)(z') > 0\}$ is finite. An execution of $\mathcal{M}$ is a (finite or infinite) sequence $z_0 \to z_1 \to z_2 \cdots$ 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 (POMDP)s

POMDPs are used to model processes which exhibit both probabilistic and nondeterministic behavior, where the states of the system are only partially observable. Formally, an POMDP is a tuple $\mathcal{M} = (Z, z_s, Act, \Delta, \equiv)$ where Z is a countable set of *states*, $z_s \in Z$ is the *initial* state, Act is a (countable) set of *actions*, $\Delta : Z \times Act \hookrightarrow Dist(Z)$ is a partial function called the *probabilistic transition relation* and $\equiv$ is an equivalence relation on Z. Furthermore, we assume that for any action α and two states z_1 and z_2 such that $z_1 \equiv z_2$, $\Delta(z_1, \alpha)$ 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 view of the execution observed by the attacker thus far.

An *execution* ρ in the POMDP $\mathcal{M}$ is a (finite or infinite) sequence $z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} z_2 \cdots$ 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 finite executions of $\mathcal{M}$ will be denoted by $\mathsf{Exec}(\mathcal{M})$ and the set of all infinite executions will be denoted by $\mathsf{Exec}^{\infty}(\mathcal{M})$. If $\rho = z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} z_2 \cdots \xrightarrow{\alpha_m} z_m$ is a finite execution then we write $last(\rho) = z_m$ and say the length of ρ, denoted $|\rho|$ is 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 is infinite or if it is finite and 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 $tr(\rho)$ to represent the *trace* of ρ,

defined as the sequence $z_0/\equiv \xrightarrow{\alpha_1} z_1/\equiv \xrightarrow{\alpha_2} z_2/\equiv \cdots \xrightarrow{\alpha_m} z_m/\equiv$. 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. Formally, an *attacker* $\mathcal{A} : \mathsf{Trace}(\mathcal{M}) \hookrightarrow Act$ is a partial function. An attacker $\mathcal{A}$ 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 $\Delta(last(\rho), \mathcal{A}(\rho))$ is defined. If defined then

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

POMDPs and State-Based Safety properties Given a POMDP $\mathcal{M} = (Z, z_s, Act, \Delta, \equiv)$, 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 $\{\kappa \mid \kappa \text{ is an execution of } \mathcal{M}^{\mathcal{A}} \text{ and } \kappa \not\models \Psi\}$ in the DTMC $\mathcal{M}^{\mathcal{A}}$ is $> 1 - p$. We say that $\mathcal{M}$ satisfies Ψ with probability $\geq p$ (written $\mathcal{M} \models_p \Psi$) if for all adversaries $\mathcal{A}$, $\mathcal{M}^{\mathcal{A}} \models_p \Psi$.

2.3 Equational theories and frames

A signature $\mathcal{F}$ contains a finite set of function symbols, each with an associated arity. We assume a countably infinite set of special constant symbols $\mathcal{N}$, which we call names and use to represent data generated freshly during a protocol execution. Variable symbols are the union of two disjoint sets $\mathcal{X}$ and $\mathcal{X}_w$ which will be used as protocol and frame variables, respectively. It is required that variable symbols are disjoint from $\mathcal{F}$. 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 is denoted by $vars(t)$. A ground term is one that contains no free variables.

A substitution σ is a function that maps variables to terms. The set $dom(\sigma) = \{x \in \mathcal{X} \cup \mathcal{X}_w \mid \sigma(x) \neq x\}$ is said to be the *domain* of the substitution σ. For the rest of the paper, each substitution will have a finite domain. A substitution σ with domain $\{x_1, ..., x_k\}$ will be denoted as $\{x_1 \mapsto t_1, ..., x_k \mapsto t_k\}$ if $\sigma(x_i) = t_i$. The set $\{t_1, .., t_k\}$ shall be denoted by $ran(\sigma)$. A substitution σ is said to be ground if every term in $ran(\sigma)$ is ground and a substitution with an empty domain shall be denoted as $\emptyset$. A substitution can be extended to terms in the usual way. We shall 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 a $\mathcal{F}$-Equation, we mean a pair

$l = r$ where $l, r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X})$ are terms that do not contain names. Two terms s and t are said to be equal with respect to an equational theory $(\mathcal{F}, E)$, denoted $s =_E t$, if $E \vdash s = t$ in the first order theory of equality. For equational theories defined in the preceding manner, if two terms containing names are equivalent, they will remain equivalent 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, $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F})$ is a binding substitution that binds the variables of the processes and $\varphi : \mathcal{X}_w \to \mathcal{T}(\mathcal{F})$ is called a frame.

Two frames φ_1 and φ_2 are said to be statically equivalent if $dom(\varphi_1) = dom(\varphi_2)$ 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 $t \in \mathcal{T}(\mathcal{F})$ is deducible from a frame φ with recipe $r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, dom(\varphi))$ in equational theory E, denoted $\varphi \vdash^r_E t$, if $r\varphi =_E t$. We often omit r and E and write $\varphi \vdash t$ if they are clear from the context.

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

Our process syntax and semantics is similar to that of [19] with the addition of an operator for probabilistic choice. It can also been seen as a variant of [26].

Process Syntax: For technical reasons, 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 $\sim$. We assume each equivalence class contains a countably infinite number of labels. Each connective in our grammar will come with a label from $\mathcal{L}$, which will later be used to identify the process performing a protocol step after a composition takes place. The equivalence relation will be used to mask the information an attacker can obtain from the internal actions of a process, in the sense that, 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. It begins by introducing what we call basic processes, which we will denote 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 $c_i \in \{\top, s = t\} \forall i \in \{1, ..., k\}$ where $s, t \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X})$. In the case of the assignment rule $(x := t)^l$, we additionally require that $x \notin vars(t)$. Intuitively, basic processes will be used to represent the actions of a particular protocol participant. 0^l is a process that does nothing and νx^l is the process that creates a fresh name and binds it to x. The process $(x := t)^l$ assigns the term t to the variable x. The test process $[c_1 \wedge ... \wedge c_k]^l$ terminates if c_i is $\top$ or c_i is $s = t$ where $s =_E t$ for all $i \in \{1, ..., k\}$

and otherwise, if some c_i is $s = t$ and $s \neq_E t$, the process deadlocks. The process $in(x)^l$ reads a term t from the public channel and binds it to x and the process $out(t)^l$ outputs a term on the public channel. The processes $P \cdot^l Q$ sequentially executes P followed by Q whereas the process $P \oplus_p^l Q$ behaves like P with probability p and like Q with probability $1 - p$.

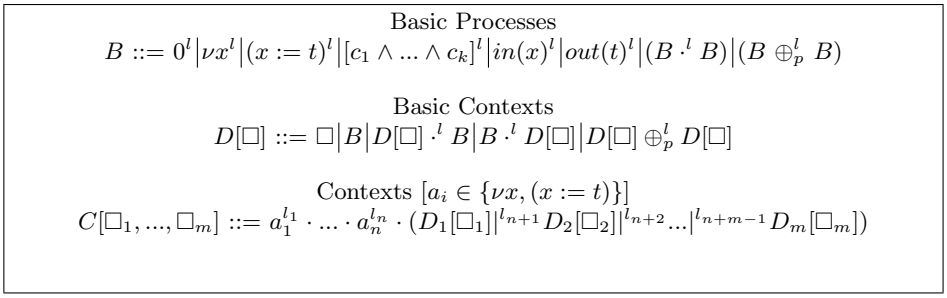

Fig. 1: Process Syntax

In Figure 1, basic processes are extended to include a special process variable $\Box$ and $\Box_1, ..., \Box_m$ are used to represent distinct processes variables. The resulting object is a basic context, which we will denote by $D[\Box]$, $D_1[\Box]$, $D_2[\Box]$, ..., $D_n[\Box]$. Notice that only a single process variable can appear in a basic context. $D_1[B_1]$ 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. In the definition of contexts, $a \in \{\nu x, (x := t)\}$. 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 the each process variable $\Box_i$ by B_i. The binding constructs in a process are assignment, input and fresh name creation. When a variable is bound in B_1, all occurrences of the variable in B_2 are bound in $B_1 \cdot B_2$. However, in $B_1 \oplus_p B_2$, a variable can occur free in B_1 and bound in B_2, or vice versa. A process containing no free variables is called ground.

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 from 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.*

The following example illustrates how protocol with randomized actions can be modeled using our process syntax.

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 k_0, k_1 and s_b 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.

$$\begin{array}{lll} \text{If } s_b = 0 & \text{Alice: } M_{A,0} = k_0 \oplus m_A, & M_{A,1} = k_1 \\ & \text{Bob: } \ M_{B,0} = k_0, & M_{B,1} = k_1 \oplus m_B \\ \text{If } s_b = 1 & \text{Alice: } M_{A,0} = k_0, & M_{A,1} = k_1 \oplus m_A \\ & \text{Bob: } \ M_{B,0} = k_0 \oplus m_B, & M_{B,1} = k_1 \end{array}$$

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 following equational theory.

$$\begin{array}{lll} \mathcal{F}_b = \{0, 1, \oplus, enc, dec, \langle, \rangle, fst, snd\} & & \\ E_b = \{dec(enc(m,k),k) = m, & x \oplus 0 = x & x \oplus x = 0 \\ \quad x \oplus y = y \oplus x & (x \oplus y) \oplus z = x \oplus (y \oplus z) & \\ \quad fst(\langle x, y\rangle) = x & snd(\langle x, y\rangle) = y\} & \end{array}$$

The role of Alice in this protocol is defined in our process syntax as

$$\begin{array}{rl} A & = A_0 \cdot (m_A := 0 \oplus_{\frac{1}{2}} m_A := 1) \cdot \\ & \quad ((s_b := 0) \cdot out(enc(s_b, k)) \cdot out(\langle k_0 \oplus m_A, k_1\rangle) \oplus_{\frac{1}{2}} \\ & \quad (s_b := 1) \cdot out(enc(s_b, k)) \cdot out(\langle k_0, k_1 \oplus m_A\rangle)) \\ A_0 & = (k_0 = 0 \oplus_{\frac{1}{2}} k_0 = 1) \cdot (k_1 = 0 \oplus_{\frac{1}{2}} k_1 = 1) \cdot out(enc(\langle k_0, k_1\rangle, k)) \end{array}$$

We now give the specification of Bob's protocol $B_1 \mid B_2$ below.

$$\begin{array}{l} B_0 = in(z_0) \cdot in(z_1) \cdot (k_0 = fst(dec(z_0, k))) \cdot \\ \qquad (k_1 = snd(dec(z_1, k))) \cdot (s_b = dec(z_1, k)) \\ B_1 = B_0 \cdot (m_B = 0 \oplus_{\frac{1}{2}} m_B = 1) \cdot out(enc(m_B, k)) \cdot [s_b = 0] \cdot out(\langle k_0, k_1 \oplus m_B\rangle) \\ B_2 = B_0 \cdot in(z_2) \cdot (m_B = dec(z_2, k)) \cdot [s_b = 1] \cdot out(\langle k_0 \oplus m_B, k_1\rangle) \end{array}$$

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 simulated using the parallel and test operators. Also notice that the communication between Alice and Bob in the above specification requires a pre-established secret key k. This key can be established by first running some key exchange protocol, which in our case, will be modeled by a context $C[\Box_1, \Box_2, \Box_3] = \nu k \cdot (\Box_1 | \Box_2 | \Box_3)$.

$$\text{INPUT}\quad \frac{r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X}) \quad \varphi \vdash^r t \quad x \notin dom(\sigma)}{(in(x)^l, \varphi, \sigma) \xrightarrow{(r,[l])} \delta_{(0,\varphi,\sigma\cup\{x\mapsto t\})}}$$

$$\text{NEW}\quad \frac{x \notin dom(\sigma) \quad n \text{ is a fresh name}}{(\nu x^l, \varphi, \sigma) \xrightarrow{(\tau,[l])} \delta_{(0,\varphi,\sigma\cup\{x\mapsto n\})}}$$

$$\text{OUTPUT}\quad \frac{vars(t) \subseteq dom(\sigma)}{(out(t)^l, \varphi, \sigma) \xrightarrow{(\tau,[l])} \delta_{(0,\varphi\cup\{w_{(|dom(\varphi)|+1,[l])}\mapsto t\sigma\},\sigma)}}$$

$$\text{TEST}\quad \frac{\forall i \in \{1,...,n\}, c_i \text{ is } \top \text{ or } c_i \text{ is } s = t \text{ where } vars(s,t) \subseteq dom(\sigma) \text{ and } s\sigma =_E t\sigma}{([c_1 \wedge ... \wedge c_n]^l, \varphi, \sigma) \xrightarrow{(\tau,[l])} \delta_{(0,\varphi,\sigma)}}$$

$$\text{ASSIGN}\quad \frac{vars(t) \subseteq dom(\sigma) \quad x \notin dom(\sigma)}{((x := t)^l, \varphi, \sigma) \xrightarrow{(\tau,[l])} \delta_{(0,\varphi,\sigma\cup\{x\mapsto t\sigma\})}}$$

$$\text{NULL}\quad \frac{(Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(0 \cdot^l Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}$$

$$\text{SEQUENCE}\quad \frac{Q_0 \neq 0 \quad (Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(Q_0 \cdot^l Q_1, \varphi, \sigma) \xrightarrow{\alpha} \mu \cdot^l Q_1}$$

$$\text{PBRANCH}\quad \frac{}{(Q_1 \oplus_p^l Q_2, \varphi, \sigma) \xrightarrow{(\tau,[l])} \delta_{(Q_1,\varphi,\sigma)} +_p \delta_{(Q_2,\varphi,\sigma)}}$$

$$\text{PARALLEL}_L\quad \frac{(Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(Q_0|^l Q_1, \varphi, \sigma) \xrightarrow{\alpha} \mu|^l Q_1}$$

$$\text{PARALLEL}_R\quad \frac{((Q_1, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(Q_0|^l Q_1, \varphi, \sigma) \xrightarrow{\alpha} Q_0|^l \mu}$$

Fig. 2: Process semantics

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 $[\![P]\!] = (Z, z_s, Act, \Delta, \equiv)$, where Z is the set of all extended processes, z_s is $(P, \emptyset, \emptyset)$, $Act = (\mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X}_w) \cup \{\tau\}, \mathcal{L}/\sim)$ and Δ is a partial function from extended processes to Act. We now give some additional notation preceding our formal definitions of Δ and $\equiv$. By $\mu \cdot^l Q$, we mean the distribution μ_1 such that $\mu_1(P', \varphi, \sigma) = \mu(P, \varphi, \sigma)$ if P' is $P \cdot^l Q$ and 0 otherwise. The distributions $\mu|^l Q$ and $Q|^l \mu$ are defined analogously. The definition of Δ is given in Figure 2. Observe that we write $(P, \varphi, \sigma) \xrightarrow{\alpha} \mu$ if $\Delta((P, \varphi, \sigma), \alpha) = \mu$. Δ is well-defined, as basic processes are deterministic and each equivalence class on $\mathcal{L}$ identifies a unique basic process. Given an extended process η, let $enabled(\eta)$

denote the set of all $(\S, [l])$ such that $(P, \varphi, \sigma) \xrightarrow{(\S,[l])} \mu$, $\S \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, \mathcal{X}_w) \cup \{\tau\}$ and l is the label of an input or output action. Using this, we lift our notion of equivalence on frames from Section 2.3 to an equivalence $\equiv$ on extended processes by requiring that two extended processes $\eta = (P, \varphi, \sigma)$ and $\eta' = (P', \varphi', \sigma')$ are equivalent if $enabled(\eta) = enabled(\eta')$ and $\varphi =_E \varphi'$.

Definition 2. *An extended process (Q, φ, σ) preserves the secrecy of $x \in vars(Q)$ in the equational theory $(\mathcal{F}, E)$, denoted $(Q, \varphi, \sigma) \models_E x$, if there is no $r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}, dom(\varphi))$ such that $\varphi \vdash^r_E x\sigma$. We write $\mathsf{Secret}(x)$, for $x \in vars(Q)$, 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 $\mathsf{Secret}(x_1) \cap ... \cap \mathsf{Secret}(x_n)$. We will often omit the braces $\{,\}$ for ease of notation.*

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

Example 2. Consider the DC-net protocol defined in Example 1. The correctness property of the protocol is that, after the completion of the protocol, the origin of the participants messages can be determined with probability at most $\frac{1}{2}$. That is, an attacker can do no better than guess which position Alice's message appears in. This is the same as asserting that an attacker cannot infer the value of the secret bit s_b. In our process semantics, this can be modeled as a secrecy property as follows. Let $A' = A \cdot S$ where $S = in(z) \cdot [z = s_b] \cdot \nu s \cdot out(s)$ and define $C = \nu k \cdot (A'|B_1|B_2)$. The inclusion of S in Alice's specification requires an attacker to correctly identify the message that belongs to Alice to derive the secret value s. Therefore, if the statement $\nu k \cdot (A'|B_1|B_2) \models_{E_b, \frac{1}{2}} \mathsf{Secret}(s)$ is valid, no attacker can do better than guess which message belongs to Alice.

4 Composition results for single session protocols

We are now ready to present our first composition results. Our focus here will begin with the scenario where two principals run a key establishment protocol over the signature $\mathcal{F}_c$ after which each principal uses the established secret to communicate in a protocol over the signature $\mathcal{F}_b$. We will then show how this result can be extended to protocols operating over the same signature, provided the messages of each protocol are tagged. Before formalizing our first result, we show how a simple DC-net protocol using Diffie-Hellman (DH) for key exchange can be modeled in our composition framework. Using the results from Theorem 1, the security guarantees of each sub-protocol are achieved for the full protocol.

Example 3. Consider the processes A', B_1 and B_2, as defined in Example 2. Recall that these processes describe a simple DC-net protocol designed to guarantee the anonymity of the protocol's participants. Further recall that the sub-protocols for Alice (A') and Bob (B_1, B_2) require a pre-established shared symmetric key k. A formal specification of the DH key exchange protocol to establish

this key is given below. This process is parameterized by the signature $\mathcal{F}_c = \{g\}$ and equations $E_c = \{(g^x)^y = (g^y)^x\}$.

$$C[\Box_0, \Box_1, \Box_2] = \nu y \cdot in(a) \cdot (A_k \cdot \Box_0 | (k := a^y) \cdot \Box_1 | (k := a^y) \cdot \Box_2 | out(g^y))$$
$$A_k = \nu x \cdot out(g^x) \cdot in(b) \cdot (k := b^x)$$

Now if $C[[\top], [\top], [\top]]$ preserves the secrecy of the shared key k and $\nu k \cdot (A'|B_1|B_2)$ preserves the secrecy of k and s with probability at least $\frac{1}{2}$, then the composed protocol $C[A', B_1, B_2]$ preserves the secrecy of s with probability at least $\frac{1}{2}$. That is, if the DC-net specification does not reveal which message belongs to Alice, then neither does the DC-net protocol using DH key exchange to establish a secret communication channel between Alice and Bob.

We now give our main result.

Theorem 1. *Let* $C[\Box_1, ..., \Box_n] = \nu k_1 \cdot ... \cdot \nu k_m \cdot (D_1[\Box_1] \mid D_2[\Box_2] \mid ... \mid D_n[\Box_n])$ *be a context over* $\mathcal{F}_c$ *with labels from* $\mathcal{L}_c$, $B_1, B_2, ..., B_n$ *be basic processes over* $\mathcal{F}_b$ *with labels from* $\mathcal{L}_b$, $q_1, q_2 \in [0, 1]$ *and* $x_s \in \bigcup_{i=1}^{n} vars(B_i) \setminus vars(C)$ *such that:*

1. $fv(C) = \emptyset$ *and* $fv(B_i) \subseteq \{x_i\}$
2. $vars(C) \cap vars(B_i) \subseteq \{x_i\}$ *for* $i \in \{1, ..., n\}$
3. $C[B_1, ..., B_n]$ *is ground*
4. $C[[\top]^{l_0}, ..., [\top]^{l_n}] \models_{E_c, q_1} \mathsf{Secret}(x_1, ..., x_n)$ *where* $l_0, ..., l_n \in \mathcal{L}_b$
5. $\nu k \cdot (x_1 := k) \cdot ... \cdot (x_n := k) \cdot (B_1|...|B_n) \models_{E_b, q_2} \mathsf{Secret}(x_1, ..., x_n, x_s)$

Then $C[B_1, ..., B_n] \models_{E, q_1 q_2} \mathsf{Secret}(x_s)$.

Before describing the proof ideas behind Theorem 1, we highlight some of the proof challenges in the following example.

Example 4. Consider the signatures $\mathcal{F}_a = \{c\}$ and $\mathcal{F}_b = \{h\}$ where c is a constant and h is a 1-ary function symbol. Let $E_a = E_b = \emptyset$ and $C[\Box_1, \Box_2] = C_1 \cdot \Box_1 | C_2 \cdot \Box_2$ be the context such that:

$$C_1 = \nu x_k \cdot (out^1(x_k) \oplus_{\frac{1}{2}} out^2(c))$$
$$C_2 = \nu y_k \cdot (out^3(y_k) \oplus_{\frac{1}{2}} out^4(c))$$

Essentially C_1 generates x_k and with probability $\frac{1}{2}$ decides to reveal it. C_2 generates y_k and with probability $\frac{1}{2}$ decides to reveal it. In both cases, when the fresh values are not revealed, a constant is output in its place. Consider the basic processes B_1 and B_2 defined as follows

$$B_1 = in^5(x) \cdot [x = x_k] \cdot \nu x_s \cdot out^6(x_s)$$
$$B_2 = in^7(y) \cdot [y = h(y_k)] \cdot \nu x_s \cdot out^8(x_s)$$

Consider the process $P = C[B_1, B_2]$ and let $C_1', C_2', \sigma, , \varphi_1, \varphi_2, \varphi_{12}, \sigma^f, \varphi_1^f, \varphi_2^f$ and φ_{12}^f be defined as follows:

$$
\begin{array}{ll}
C_1' & = out^1(x_k) \oplus_{\frac{1}{2}} out^2(c) \\
C_2' & = out^3(y_k) \oplus_{\frac{1}{2}} out^4(c) \\
\sigma & = \{x_k \mapsto k_1, y_k \mapsto k_2\} \\
\varphi_0 & = \{w_1 \to c\} \\
\varphi_1 & = \{w_1 \to k_1\} \\
\varphi_2 & = \{w_1 \to c, w_2 \to k_2\} \\
\varphi_{00} & = \{w_1 \to c, w_2 \to c\} \\
\varphi_{10} & = \{w_1 \to k_1, w_2 \to c\}
\end{array}
\qquad
\begin{array}{ll}
\varphi_{12} & = \{w_1 \to k_1, w_2 \to k_2\} \\
\varphi_{02} & = \{w_1 \to k_1, w_2 \to c\} \\
\sigma_1^f & = \{x_k \mapsto k_1, y_k \mapsto k_2, x \mapsto k_1, x_s \mapsto k_3\} \\
\sigma_2^f & = \{x_k \mapsto k_1, y_k \mapsto k_2, y \mapsto h(k_1), x_s \mapsto k_3\} \\
\varphi_1^f & = \{w_1 \to k_1, w_2 \to c, w_4 \mapsto k_3\} \\
\varphi_2^f & = \{w_1 \to c, w_2 \to k_2, w_6 \mapsto k_3\} \\
\varphi_{12}^f & = \{w_1 \to k_1, w_2 \to k_2, w_4 \mapsto k_3\}
\end{array}
$$

The execution of P shown in Figure 3 reveals x_s with probability $\frac{3}{4}$. Observe that the transitions out of the states labeling $(B_1|B_2, \varphi_1, \sigma)$ involve transitions of B_1 while the transitions out of $(B_1|B_2, \varphi_2, \sigma)$ involve transitions of B_2. If we try to fire the same transitions out of $(B_1|B_2, \varphi_2, \sigma)$ as in $(B_1|B_2, \varphi_1, \sigma)$ the process will deadlock because the attacker cannot deduce x_k in φ_2. From this, it is easy to see that the execution shown in Figure 3 cannot be written as an interleaving of one execution of $C_1|C_2$ and one execution of $B_1|B_2$. As a result, the proof technique of [19] is not immediately applicable. Nevertheless, we will be able to show that P keeps x_s secret with probability at least $\frac{1}{4}$.

In the execution of P shown in Figure 3, the attacker performs different actions depending of the result of coin toss made by C_1. When C_1 outputs a nonce, B_1 is scheduled before B_2. When C_1 outputs the constant c, B_2 is executed first. Such an attack is valid, even when considering our restricted class of adversaries. The reason is that the attacker can infer the result of the coin toss in C_1 by observing what is output.

The proof of Theorem 1 will utilize an extension of the seperation result from [19], which intuitively says that for a context C and a basic process B, if the composition $C[B]$, where C and B are over disjoint signatures and derive a set of variables with probability q, can be transformed into the composition $C'[B']$, where C' and B' represent α-renamings of C and B, such that $vars(C') \cap vars(B') = \emptyset$ and the same secret derivation guarantees are achieved. Given this result, an attacker for a composition of $C[B]$ can be transformed into an attacker for a composition of two protocols C' and B' over disjoint variables. From this attacker $\mathcal{A}$, we need to construct and an attacker $\mathcal{A}'$ for one of the sub-protocols C' or B'. Because C' and B' are simply α-renamings of C and B, $\mathcal{A}'$ is sufficient for contradicting a secrecy guarantee about one of the sub-protocols C or B. One of the challenges in constructing $\mathcal{A}'$ is that the attacker A may use terms over $\mathcal{F}_b \cup \mathcal{F}_c$. That is, it may construct inputs using terms output by both of the sub-protocols C' and B'. Our technique is to transform $\mathcal{A}$ into what we call a "pure" attacker, that constructs its inputs for actions of C' (resp B') using only terms output by actions of C' (resp B'). We define this concept formally. Given a set of labels L closed under $\sim$, let $\mathcal{X}_w^L = \{w_{i,[l]} \mid w_{i,[l]} \in \mathcal{X}_w \wedge l \in L \wedge i \in \mathbb{N}\}$

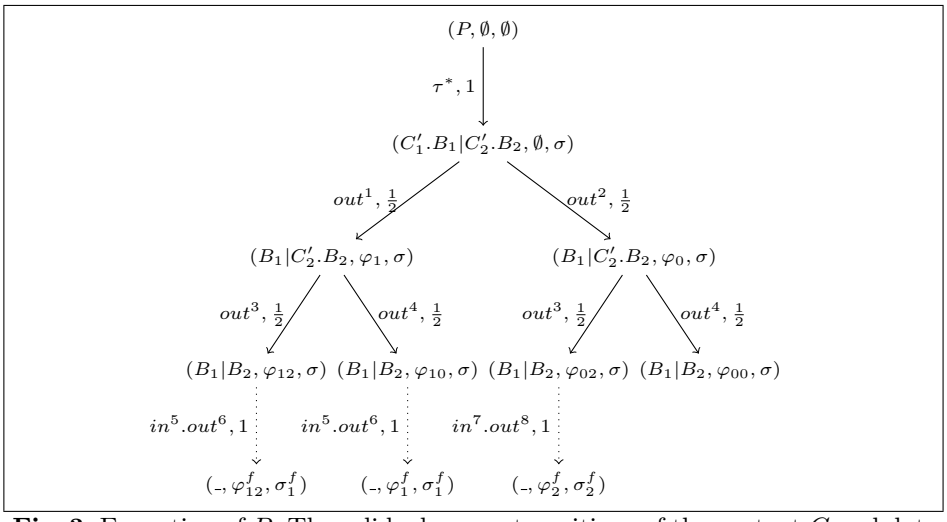

Fig. 3: Execution of P. The solid edges are transitions of the context C and dotted edges are transitions of the basic processes B_1, B_2. For convenience, the edges in the drawn execution tree may compose of more than 1 action. The recipes used in in^3 and in^5 are w_1 and $h(w_2)$ respectively. The transition probabilities also label the edges.

Definition 3. *Let L be a set of labels closed under $\sim$ and $\mathcal{F}$ be a signature. An attacker $\mathcal{A}$ for a process P is said to be pure with respect to $(L, \mathcal{F})$ if whenever $\mathcal{A}$ chooses the action $(r, [l])$ we have $r \in \mathcal{T}(\mathcal{F}, \mathcal{X}_w^L)$.*

For the kind of compositions we consider in Theorem 1, for example $C'[B']$, an attacker can be transformed into one that is pure with respect to both of the sub-protocols $C'[\top]$ and B'. This construction follows from the techniques outlined in [19].

We are now ready to describe the crux of our composition theorem. The fundamental technical challenge of this result is to show that, for some context $C[\Box]$, process B and set of secrets S, if $C[[\top]] \models_{q_1} \mathsf{Secret}(S)$ and $B \models_{q_2} \mathsf{Secret}(S)$, then $C[B] \models_{q_1 q_2} \mathsf{Secret}(S)$. In light of the preceding results, one must transform a pure attacker for a composed protocol $C[B]$ that reveals some secret values with probability $\geq 1 - q_1 q_2$ into an attacker for one of the sub-protocols C or B that derives the secret values with probability at least $1 - q_1$ or $1 - q_2$, respectively. This essentially boils down to proving that an attacker for the asynchronous product of two POMDP's M_1 and M_2, denoted $M_1 \otimes M_2$, can be transformed into an attacker for either M_1 or M_2. This is achieved by transforming the attacker $\mathcal{A}$ for $M_1 \otimes M_2$ into an attacker $\mathcal{A}'$ with the following two properties:

- $\mathcal{A}'$ executes all of the actions from M_1 before executing any actions from M_2.

- For any two executions $\rho, \rho' \in \mathsf{Exec}((M_1 \otimes M_2)^{\mathcal{A}'})$, if the projection of ρ and ρ' onto their components from M_i (for $i \in \{1,2\}$) produces two equivalent traces and $\mathcal{A}'$ picks an action from M_i, then $\mathcal{A}'(tr(\rho)) = \mathcal{A}'(tr(\rho'))$.
- $\mathcal{A}'$ derives the secret values in S with probability greater than or equal to that of the attacker $\mathcal{A}$.

The details of this construction are quite involved, and can be found in the accompanying technical report [6]. As a result of Theorem 1, one can reason about protocols composed sequentially by taking a context with a single basic context where a single hole appears at the end. The same is true for protocols composed in parallel, as given by Corollary 1. In this setting, one considers a context built over two basic contexts. One basic context contains only a hole, while the other contains no holes.

Corollary 1. *Let C be a basic process over $\mathcal{F}_c$ with labels from $\mathcal{L}_c$ and B be a basic processes over $\mathcal{F}_b$ with labels from $\mathcal{L}_b$ and $q_1, q_2 \in [0,1]$ such that:*

1. $vars(C) \cap vars(B) = \emptyset$
2. $C \models_{E_c,q_1} \mathsf{Secret}(x_c)$ *for* $x_c \in vars(C)$
3. $B \models_{E_b,q_2} \mathsf{Secret}(x_b)$ *for* $x_b \in vars(B)$

Then $(C|B) \models_{E,q_1 q_2} \mathsf{Secret}(x_b, x_c)$.

It is important to point out that the security guarantees of the composed process may in fact be stronger than what we can prove utilizing Theorem 1. This is because we always assume the worst case in that context assigns the same secret values to each basic process. As a result, our composition result will in some cases lead to only an under-approximation on the probability that a set of variables is kept secret, as shown by the following example:

Example 5. Consider the signatures $\mathcal{F}_b = \{h\}$ and $\mathcal{F}_c = \{\}$ with empty equational theories and the context defined as follows:

$$C[\Box_1, \Box_2] = \nu k_1 \cdot \nu k_2 \cdot (([x_1 := k_1] \oplus_{\frac{1}{2}} [x_1 := k_2]) \cdot \Box_1 \mid [x_2 := k_2] \cdot \Box_2)$$

Essentially, the context generates shared secrets x_1 and x_2 for two sub-protocols $\Box_1$ and $\Box_2$ to be run in parallel. For the sub-protocol $\Box_1$, it sets the secret x_1 to k_1 with probability $\frac{1}{2}$ and to k_2 with probability $\frac{1}{2}$. In the second sub-protocol, the shared secret x_2 is set to k_2. Now consider the sub-protocols B_1 and B_2 defined as follows:

$$B_1 = out(h(x_1)) \oplus_{\frac{1}{2}} 0$$
$$B_2 = in(z) \cdot [z = h(x_2)] \cdot \nu x_s \cdot out(x_s)$$

B_1 outputs $h(x_1)$ with probability $\frac{1}{2}$ and with probability $\frac{1}{2}$ does nothing. B_2 checks if the attacker can construct $h(x_2)$ before revealing x_s. It is easy to see that $C[B_1, B_2]$ reveals x_s with probability $\frac{1}{4}$. This is because the attacker can construct $h(x_2)$ when x_1 and x_2 are equal (which happens with probability $\frac{1}{2}$)

and when B_1 reveals $h(x_1)$ (which also happens with probability $\frac{1}{2}$). In-fact, we can easily show that $C[B_1, B_2]$ keeps x_s secret with probability exactly $\frac{3}{4}$. However, Theorem 1 can only show $C[B_1, B_2]$ keeps x_s secret with probability $\frac{1}{2}$, since in our composition results, we assume that x_1 and x_2 get the same secret name.

It is often necessary for protocols to share basic cryptographic primitives, such as functions for encryption, decryption and hashing. We extend our composition result to such protocols. The key ingredient for composition in this context is tagging, a syntactic transformation of a protocol and its signature, designed to ensure secure composition. Essentially, tagging a protocol appends a special identifier to each of the messages that it outputs. When the protocol performs an input, it will recursively test all subterms in the input message to verify their tags are consistent with the protocol's tag. One limitation with tagging is that its correctness largely depends on the signature in question. As in [19], we will limit the class of cryptographic primitives we consider to symmetric encryption and a hash function, with the understanding that our results can be extended to primitives for asymmetric encryption.

Let C be a context and B be a basic process, both over the equational theory $(\mathcal{F}_{enc}, E_{enc})$ where $\mathcal{F}_{enc} = \{enc, dec, h\}$ and $E_{enc} = \{dec(enc(m, rn, k), k) = m\}$. 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 $(\mathcal{F}_{enc}, E_{enc})$ to a context C^d by replacing every occurrence of the function symbols enc, dec and h in C by enc_d, dec_d and h_d, respectively. The resulting context C^d is over the signature $(\mathcal{F}^d_{enc}, E^d_{enc})$, for $\mathcal{F}^d_{enc} = \{enc_d, dec_d, h_d\}$ and $E^d_{enc} = \{dec_d(enc_d(m, rn, k), k) = m\}$. Given C^c and B^b over the disjoint signatures $\mathcal{F}^c_{enc}$ and $\mathcal{F}^b_{enc}$, the tagging function $\lceil \_ \rceil$ is then applied to C^c and B^b, generating the the tagged versions of C and B. We omit the details of this function but note that our tagging scheme is similar to that of [4]. The full details can be found in the technical report [6].

Essentially the tagging scheme enforces the requirement that, whenever a protocol manipulates a term, that term should be tagged with the identifier of the protocol. This is achieving by prefixing every atomic action in a tagged protocol with a conjunction of tests such that, 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. Tagging is a means to enforce the disjointness condition on the context and basic process signatures in Theorem 1. In particular, we can 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 this, we can prove an analogous result to Theorem 1 for tagged protocols.

Theorem 2. *Let* $C[\Box_1, ..., \Box_n] = \nu k_1 \cdot ... \cdot \nu k_m \cdot (D_1[\Box_1] \mid D_2[\Box_2] \mid ... \mid D_n[\Box_n])$ *be a context over* $\mathcal{F}_{enc}$ *with labels from* $\mathcal{L}_c$*,* $B_1, B_2, ..., B_n$ *be basic processes over*

$\mathcal{F}_{enc}$ with labels from $\mathcal{L}_b$, $q_1, q_2 \in [0,1]$ and $x_s \in \bigcup_{i=1}^{n} vars(B_i) \setminus vars(C)$ such that:

- $fv(C) = \emptyset$ *and* $fv(B_i) \subseteq \{x_i\}$
- $vars(C) \cap vars(B_i) \subseteq \{x_i\}$ *for* $i \in \{1, ..., n\}$
- $C[B_1, ..., B_n]$ *is ground*
- $C[[\top]^{l_0}, ..., [\top]^{l_n}] \models_{E_{enc}, q_1} \mathsf{Secret}(x_1, ..., x_n)$ *where* $l_0, ..., l_n \in \mathcal{L}_b$
- $\nu k \cdot (x_1 := k) \cdot ... \cdot (x_n := k) \cdot (B_1 | ... | B_n) \models_{E_{enc}, q_2} \mathsf{Secret}(x_1, ..., x_n, x_s)$.

Then $\lceil C^c[B_1^b, ..., B_n^b] \rceil \models_{E_{enc} \cup E_{tag}, q_1 q_2} \mathsf{Secret}(x_s)$.

5 Replication

In this section, we extend our composition result to protocols that can run multiple sessions.[3] We will begin by considering processes that contain only a bounded version of the replication operator. The bounded replication operator has an explicit bound that limits the number of times a process can replicate. We limit ourselves to processes that contain only a single occurrence of this replication operator. This restriction is not limiting for the applications we consider and it will simplify the proofs. It is, however, possible to extends our results to a more general setting in which a process can contain multiple occurrences of the replication operator.

We will start by showing showing that if the protocols $C = \nu k_1 \cdot ... \cdot \nu k_m \cdot !_n(C[\Box_1] | ... | C[\Box_l])$ and $!_n(B_1 | ... | B_l)$ are proven secure with probability at least p and q, respectively, then the composition $\nu k_1 \cdot ... \cdot \nu k_m . !_n(C[B_1] | ... | C[B_l])$ is secure with probability at least pq, provided the protocol messages are tagged with both a protocol identifier and a unique session identifier. A similar result (with the absence of the session identifier), has been claimed in [19] for nonrandomized protocols (with p and q both being 1). However, we discovered a simple counterexample, which works for the case of two sessions. Essentially the reason for this attack is that protocol messages from one session can be confused with messages from the other session.

Example 6. Consider the signatures $\mathcal{F}_b = \{h, c\}$ and $\mathcal{F}_a = \{\}$ where c is a constant, h is a 1-ary function symbol and $E = E_a \cup E_b = \emptyset$. We will consider two sessions of the composed protocol.

Let P be the process defined as:

$$P = \nu k_1 \cdot \nu k_2 . !_2(P_1 \mid P_2)$$

where $P_1 = (x_k := k_1)$ and $P_2 = (y_k := k_2)$. Let Q be the process defined as:

$$\begin{aligned} Q &= !_2(\nu k \cdot ((x_k := k) \cdot Q_1 \mid (y_k := k) \cdot Q_2)) \\ Q_1 &= (in(y) \cdot ([y = c] \cdot out^l(h(x_k)) \mid [y = h(x_k)] \cdot \nu x_s \cdot out^{l'}(x_s)) \\ Q_2 &= 0. \end{aligned}$$

[3] n sessions of P will be denoted by $!_n P$.

Clearly, P keeps x_k and y_k secret with probability 1 and Q keeps x_k, y_k and x_s secret with probability 1. Theorem 3 from [19] would imply that x_s is kept secret by $W = \nu k_1 \cdot \nu k_2 \cdot !_2(P_1 \cdot Q_1 \mid P_2 \cdot Q_2)$ in both sessions of the protocol. However, we can show that this is not the case. The reason is as follows. In both sessions of the composed protocol, x_k gets the same value. In the first session of the composed protocol, when y is input by Q_1, attacker sends the constant c. Thereafter, the attacker learns $h(x_k)$ because Q_1 outputs it. In the second session of the composed protocol, the attacker sends $h(x_k)$ to Q_1; the check $[y = h(x_k)]$ succeeds and the attacker learns x_s in this session.

In process calculus terms, this attack can be realized by the execution:

$$(W, \emptyset, \emptyset) \rightarrow^* (W', \emptyset, \sigma') \rightarrow^* (W'', \varphi'', \sigma'')$$

where

$$\begin{aligned} W' &= (Q_1^1 | Q_1^2) \\ \sigma' &= \{k_1 \mapsto n_1, k_2 \mapsto n_2, x_k^1 \mapsto n_1, y_k^1 \mapsto n_2, \\ &\quad x_k^2 \mapsto n_1, y_k^2 \mapsto n_2\} \\ W'' &= 0 \\ \sigma'' &= \sigma' \cup \{y^1 \mapsto c, y^2 \mapsto h(n_1), x_s^2 \mapsto n_3\} \\ \varphi'' &= \{w_l \mapsto h(n_1), w_{l'} \mapsto n_3\} \end{aligned}$$

Note above that we have used superscripts on variables x_k, y_k, y and x_s in the substitutions to indicate their values in different sessions. Essentially in this execution in $(W', \emptyset, \sigma')$, P is finished in both sessions and assigned x_k and y_k the same values in both sessions. The role Q_2 is also finished in both sessions. Q_1^1 is the first session of Q_1 and Q_1^2 is the second session of Q_1. Now in Q_1^1, the attacker inputs c for y resulting in Q_1^1 leaking $h(n_1)$. In Q_1^2, the attacker can input $h(n_1)$ and learn the value of x_s generated.

Formally, a context containing bounded replication is defined as

$$C[\Box_1, ..., \Box_m] ::= a_1^{l_1} \cdot ... \cdot a_n^{l_n} \cdot !_n^l (D_1[\Box_1] |^{l_{n+1}} D_2[\Box_2] |^{l_{n+2}} ... |^{l_{n+m-1}} D_m[\Box_m])$$

where $a \in \{\nu x, (x := t)\}$ and $n \geq 2$ is a natural number. The semantics for this bounded replication operator is given in Figure 4, where $i, j \in \mathbb{N}$ are used to denoted the smallest previously unused indices. We will use $P(i)$ to denote that process that results from renaming each occurrence of $x \in vars(P)$ to x^i for $i \in \mathbb{N}$. When $P(i)$ or $P(j)$ is relabeled freshly as in Figure 4, the new labels must all belong to the same equivalence class (that contains only those labels). The notation x^* denotes the infinite set $\{x^0, x^1, x^2, ...\}$.

$$\text{B-REPLICATION} \quad \frac{n > 2 \quad l' \text{ is a fresh label} \quad P(i) \text{ is relabeled freshly}}{(!_n^l P, \varphi, \sigma) \xrightarrow{(\tau, [l])} \delta_{(P(i) |^{l'} !_{n-1}^l P, \varphi, \sigma)}}$$

$$\text{B-REPLICATION}_{n=2} \quad \frac{l' \text{ is a fresh label} \quad P(i), P(j) \text{ are relabeled freshly}}{(!_2^l P, \varphi, \sigma) \xrightarrow{(\tau, [l])} \delta_{(P(i) |^{l'} P(j), \varphi, \sigma)}}$$

Fig. 4: Bounded 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 $!_m in(x) \cdot P$ and the execution

$$(!_m in(x) \cdot P, \emptyset, \emptyset) \rightarrow^* (in(x) \cdot P | !_{m-1} in(x) \cdot P, \varphi, \{x \mapsto t\} \cup \sigma)$$

where variable renaming does not occur. This execution corresponds to the attacker replicating $!_m in(x) \cdot P$, running one instance of $in(x) \cdot P$ and then replicating $!_m in(x) \cdot P$ 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 $in(x)$. In other words, an attacker can only effective run one copy of $!_m in(x) \cdot P$ for any process of the form $!_m in(x) \cdot P$. It is also convenient to consider this restricted version of α-renaming in view of secrecy. In particular, if a variable is α-named arbitrarily with each application of "B-REPLICATION", then the definition of $!^l_n P$ keeping $x \in vars(P)$ secret becomes unclear, or at least more complicated.

As mentioned in Example 6, our composition result must prevent messages from one session of a process with bounded replication from being confused with messages from another sessions. We achieve this in the following way. Our composed processes will contain an occurrence of $\nu\lambda$ directly following the occurrence of a bounded replication operator. This freshly generated "session tag" will then be used to augment tags occurring in the composed processes. We have the following result.

Theorem 3. *Let* $C[\Box_1, ..., \Box_n] = \nu k_1 \cdot ... \cdot \nu k_m . !_u \nu\lambda \cdot (D_1[\Box_1] \mid D_2[\Box_2] \mid ... \mid D_n[\Box_n])$ *be a context over* $\mathcal{F}_{enc}$ *with labels from* $\mathcal{L}_c$, $B_1, B_2, ..., B_n$ *be basic processes over* $\mathcal{F}_{enc}$ *with labels from* $\mathcal{L}_b$, $q_1, q_2 \in [0,1]$ *and* $x_s \in \bigcup_{i=1}^n vars(B_i) \setminus vars(C)$ *such that:*

- $fv(C) = \emptyset$ *and* $fv(B_i) \subseteq \{x_i\}$
- $vars(C) \cap vars(B_i) \subseteq \{x_i\}$ *for* $i \in \{1, ..., n\}$
- $\lambda \notin vars(P) \cup vars(Q)$
- $C[B_1, ..., B_n]$ *is ground*
- $C[[\top]^{l_0}, ..., [\top]^{l_n}] \models_{E_{enc}, q_1} \mathsf{Secret}(x_1, ..., x_n)$ *where* $l_0, ..., l_n \in \mathcal{L}_b$
- $\nu k \cdot (x_1 := k) \cdot ... \cdot (x_n := k) \cdot !_m (B_1 | ... | B_n) \models_{E_{enc}, q_2} \mathsf{Secret}(x_1, ..., x_n, x_s^*)$

Then $\lceil \nu k_1 \cdot ... \cdot \nu k_m \cdot !_u \nu\lambda \cdot (D_1^c[B_1^{(b,\lambda)}] \mid D_2^c[B_2^{(b,\lambda)}] \mid ... \mid D_n^c[B_n^{(b,\lambda)}]) \rceil \models_{E_{enc} \cup E_{tag}, q_1 q_2}$ $\mathsf{Secret}(x_s^*)$.

As a final result, we will show how protocols containing unbounded replication can be composed. That is, we will consider processes over the following grammar.

$$C[\Box_1, ..., \Box_m] ::= a_1^{l_1} \cdot ... \cdot a_n^{l_n} \cdot !^l (D_1[\Box_1] |^{l_{n+1}} D_2[\Box_2] |^{l_{n+2}} ... |^{l_{n+m-1}} D_m[\Box_m])$$

where $a \in \{\nu x, (x := t)\}$. The semantics of this new replication operator are given in Figure 5, where again, $i \in \mathbb{N}$ is the smallest previously unused index.

$$\text{REPLICATION}_N \quad \frac{l' \text{ is a fresh label} \quad P(i) \text{ is relabeled freshly}}{(!^l P, \varphi, \sigma) \xrightarrow{(r,l)} \delta_{(P(i)|^{l'}!^l P, \varphi, \sigma)}}$$

Fig. 5: Replication semantics

As before, when $P(i)$ is relabeled freshly, the new labels must all belong to the same equivalence class.

As previously eluded to, we cannot state a result in the style of Theorem 3 with non-trivial probabilities. This is because, in the unbounded setting, a attacker can always amplify the probability of deriving a secret by running an attack on more sessions of a protocol. Such a restriction makes our result for unbounded processes almost identical to that of Theorem 6 from [19]. Our result, however, has two main advantages. It elimiates the still applicable attack of Example 6 while considering a richer class of processes.

Theorem 4. *Let* $C[\Box_1, ..., \Box_n] = \nu k_1 \cdot ... \cdot \nu k_m . !\nu\lambda \cdot (D_1[\Box_1] \mid D_2[\Box_2] \mid ... \mid D_n[\Box_n])$ *be a context over* $\mathcal{F}_{enc}$ *with labels from* $\mathcal{L}_c$, $B_1, B_2, ..., B_n$ *be basic processes over* $\mathcal{F}_{enc}$ *with labels from* $\mathcal{L}_b$ *and* $x_s \in \bigcup_{i=1}^{n} vars(B_i) \setminus vars(C)$ *such that:*

- $fv(C) = \emptyset$ *and* $fv(B_i) \subseteq \{x_i\}$
- $vars(C) \cap vars(B_i) \subseteq \{x_i\}$ *for* $i \in \{1, ..., n\}$
- $\lambda \notin vars(P) \cup vars(Q)$
- $C[B_1, ..., B_n]$ *is ground*
- $C[[\top]^{l_0}, ..., [\top]^{l_n}] \models_{E_{enc},1} \mathsf{Secret}(x_1, ..., x_n)$ *where* $l_0, ..., l_n \in \mathcal{L}_b$
- $\nu k \cdot (x_1 := k) \cdot ... \cdot (x_n := k) \cdot !_m(B_1|...|B_n) \models_{E_{enc},1} \mathsf{Secret}(x_1, ..., x_n, x_s^*)$

Then $\lceil \nu k_1 \cdot ... \cdot \nu k_m . !\nu\lambda \cdot (D_1^c[B_1^{(b,\lambda)}] \mid D_2^c[B_2^{(b,\lambda)}] \mid ... \mid D_n^c[B_n^{(b,\lambda)}]) \rceil \models_{E_{enc} \cup E_{tag},1} \mathsf{Secret}(x_s^*)$.

6 Conclusions

We have studied the problem of securely composing two randomized security protocols. For one session, we show that if P is secure with probability p and Q is secure with probability q then the composed protocol is secure with probability at least pq if the protocol messages are tagged with the information of which protocol they belong to. The same result applies to multiple sessions except that in addition the protocol messages of Q also need to be tagged with session identifiers. The focus of this work has been secrecy properties. In terms of future work, we plan to investigate when composition of randomized security protocols preserve indistinguishability properties.

References

1. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *28th ACM Symp. on Principles of Programming Languages (POPL'01)*, pages 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. Composing security protocols: from confidentiality to privacy. *preprint avaliable at: http://arxiv.org/pdf/1407.5444v3.pdf*.
4. 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, Cambridge Massachusetts, USA, 2012. IEEE Computer Society Press.
5. 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)*, volume 5330 of *Lecture Notes in Artificial Intelligence*, pages 128–142. Springer, 2008.
6. Matthew S. Bauer, Rohit Chadha, and Mahesh Viswanathan. Composing Protocol with Randomized Actions. Technical report, University of Illinois at Urbana-Champaign, Department of Computer Science, 2016.
7. 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.
8. 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.
9. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In Moni Naor, editor, *42nd IEEE Symp. on Foundations of Computer Science (FOCS'01)*, pages 136–145. IEEE Comp. Soc. Press, 2001.
10. 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)*, volume 3876 of *Lecture Notes in Computer Science*, pages 380–403. Springer, 2006.
11. Marco Carbone and Joshua D. Guttman. Sessions and separability in security protocols. In *Proceedings of the Principles of Security and Trust - Second International Conference, POST 2013*, pages 267–286, 2013.
12. 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.
13. K. Chatzikokolakis and C. Palamidessi. Making Random Choices Invisible to the Scheduler. *Information and Computation*, 2010, to appear.
14. David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptology*, 1(1):65–75, 1988.
15. L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. PhD thesis, Radboud University of Nijmegen, 2006.
16. Céline Chevalier, Stéphanie Delaune, and Steve Kremer. Transforming password protocols to compose. In *31st Conference on Foundations of Software Technology and Theoretical Computer Science*, Leibniz International Proceedings in Informatics, pages 204–216. Leibniz-Zentrum für Informatik, 2011.
17. Véronique Cortier, Jérémie Delaitre, and Stéphanie Delaune. Safely composing security protocols. In V. Arvind and Sanjiva Prasad, editors, *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science*

(FSTTCS'07), volume 4855 of *Lecture Notes in Computer Science*, pages 352–363. Springer, December 2007.
18. Véronique Cortier and Stéphanie Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, February 2009.
19. 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.
20. 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.
21. L. de Alfaro. The Verification of Probabilistic Systems under Memoryless Partial Information Policies is Hard. In *PROBMIV*, 1999.
22. 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. IEEE Computer Society Press, June 2008.
23. Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.
24. F.D. Garcia, P. van Rossum, and A. Sokolova. Probabilistic Anonymity and Admissible Schedulers. *CoRR*, abs/0706.1019, 2007.
25. David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Onion routing. *Commun. ACM*, 42(2):39–41, 1999.
26. Jean Goubault-Larrecq, Catuscia Palamidessi, and Angelo Troina. A probabilistic applied pi-calculus. In *5th Asian Symposium on Programming Languages and Systems (APLAS'07)*, volume 4807 of *Lecture Notes in Computer Science*, pages 175–290. Springer, 2007.
27. Carl A. Gunter, Sanjeev Khanna, Kaijun Tan, and Santosh S. Venkatesh. Dos protection for reliably authenticated broadcast. In *NDSS*, 2004.
28. Joshua D. Guttman. Authentication tests and disjoint encryption: A design method for security protocols. *Journal of Computer Security*, 12(3-4):409–433, 2004.
29. Joshua D. Guttman. Cryptographic protocol composition via the authentication tests. In Luca de Alfaro, editor, *the Foundations of Software Science and Computational Structures, 12th International Conference (FOSSACS 2009)*, volume 5504 of *LNCS*, pages 303–317. Springer, 2009.
30. Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C. Mitchell. A modular correctness proof of ieee 802.11i and TLS. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *the 12th ACM Conference on Computer and Communications Security, (CCS 2005)*, pages 2–15. ACM, 2005.
31. 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*, ASIA CCS '14, pages 435–446, New York, NY, USA, 2014. ACM.
32. Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92, 1998.
33. 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.