A general framework for computational soundness proofs – or – The computational soundness of the applied pi-calculus

Michael Backes[1,3], Dennis Hofheinz[2], and Dominique Unruh[1]
[1]Saarland University [2]CWI [3]MPI-SWS

Abstract

We provide a general framework for conducting computational soundness proofs of symbolic models. Our framework considers arbitrary sets of constructors, deduction rules, and computational implementations, and it abstracts away from many details that are not core for proving computational soundness such as message scheduling, corruption models, and even the internal structure of a protocol. We identify several properties of a so-called simulator such that the existence of a simulator with all these properties already entails computational soundness in the sense of preservation of trace properties in our framework. This simulator-based characterization allows for proving soundness results in a conceptually modular and generic way.

We exemplify the usefulness of our framework by proving the first computational soundness result for the full-fledged applied π-calculus under active attacks. Concretely, we embed the applied π-calculus into our framework and give a sound implementation of public-key encryption.

1 Introduction

Proofs of security protocols are known to be error-prone and, owing to the distributed-system aspects of multiple interleaved protocol runs, awkward for humans to make. Hence work towards the automation of such proofs started soon after the first protocols were developed. From the start, the actual cryptographic operations in such proofs were idealized into so-called Dolev-Yao models, following [19, 20, 26], e.g., see [22, 29, 2, 25, 28, 9]. This idealization simplifies proof construction by freeing proofs from cryptographic details such as computational restrictions, probabilistic behavior, and error probabilities. While it was initially not clear whether Dolev-Yao models are a sound abstraction from real cryptography with its computational security definitions, a large number of results in the last years helped to establish a general understanding which cryptographic primitives can or cannot be proven computationally sound in which adversarial settings under which assumptions. (A far from being complete list of these works includes [3, 7, 5, 6, 24, 27, 18, 10, 21, 15, 17, 4, 23, 16].)

A careful inspection of this series of results, however, reveals that the soundness theorems stated in these works, and even more so the frameworks that underly these theorems, differ considerably in many respects. These differences range from various ways of syntactically expressing security protocols and corresponding restrictions on the set of permitted protocol classes, to different semantics for modelling protocol communication and communication with the adversary, to different (often incomparable) notions of computational soundness, to different assumptions on the adversary's capabilities, etc.[1] Moreover, many of these frameworks were freshly invented in the respective papers; they hence lack support of suitable verification tools and are more likely to suffer from idiosyncracies than more established frameworks for reasoning about security protocols.

The lack of a common framework that underlies computational soundness results complicates the thorough comparison of their strengths and limitations. Even worse, it often even remains unclear if restrictions in computational soundness results, e.g., to require additional randomization in the cryptographic implementation or to require the absence of key cycles, stem from idiosynracies of the underlying framework, or if they constitute conceptual limitations for establishing the desired computational soundness result for the prevalent cryptographic definitions. Moreover, framework-specific assumptions complicate the extension of these results to other frameworks, or to more comprehensive settings, e.g., a more expressive set of cryptographic primitives or a stronger adversary. Actually, such results are often proven from scratch again (for an extended new framework).

The present situation calls for a general unified framework

[1]To get a small impression of the diversity of the statements: [7, 23, 16] all establish (among others results) the computational soundness of symbolic encryption (either symmetric or asymmetric): First, [7] expresses protocols as probabilistic input-output automata, exploits the communication model offered by the Reactive Simulatability (RSIM) framework [8], and shows computational soundness in the sense of reactive simulatability for IND-CCA2-secure, additionally randomized encryption schemes. Second, [23] expresses protocols and their communication using a newly introduced concept called abstract algebras, and shows computational soundness as preservation of static equivalence in the presence of an adaptive, but passive adversary for IND-CPA-secure encryption schemes that are additionally length-concealing. Third, [16] expresses protocols and their communication within a small fragment of the applied π-calculus, and shows computational soundness as preservation of observational equivalence in the presence of an active adversary for IND-P1-C1-secure encryption schemes.

for computational soundness results. This framework should be designed in a modular, extensible way and allow for conveniently enbedding existing frameworks, thereby leveraging existing soundness results. Moreover, the framework should be tightly linked to a formal calculus that is well understood and accepted by the scientific community, that is expressive enough for expressing and reasoning about state-of-the-art protocols, and that is accessible to state-of-the-art verification tools.

1.1 Our contribution

A general framework for computational soundness proofs. We provide a general framework for conducting computational soundness proofs of symbolic models that allows to formulate soundness results in a unified and comparable manner. The framework comprises a general definition of symbolic protocols, their symbolic and computational execution, as well as a definition of computational soundness for trace properties.[2]

The framework does not put constraints on the symbolic model; in particular, it permits arbitrary sets of constructors, deduction rules, and computational implementations, and it is specifically tailored at establishing soundness results in that it abstracts away from many details that are not core for proving computational soundness such as message scheduling, corruption models, and even the internal structure of a protocol. Instead, we treat a protocol as one entity that interacts with an attacker. This allows for a unified treatment of different symbolic models.

In order to simplify conducting soundness proofs in this framework, we identify several properties of a so-called simulator such that the existence of a simulator with all these properties already entails computational soundness in the sense of preservation of trace properties in our framework. Intuitively, a simulator interacts with the symbolic protocol in a way that mimics the interaction of a computational adversary with the implementation. It thus might remind of the simulation-based proofs of computational soundness [7, 5, 6, 15], but it does not depend on framework-specific details such as scheduling, polynomial runtime restrictions, etc. This simulator-based characterization allows for proving soundness results in a conceptually modular and generic way, as it holds for arbitrary models that are embedded in the framework.

Computational soundness of the applied π-calculus As the second contribution of this paper, we show how to use our framework to establish the first computational soundness result for a fully-fledged applied π-calculus under active attacks. We consider the process calculus proposed in [14] additionally augmented with events; the calculus in [14] itself is a combination of the original applied π-calculus [1] with one of its dialects [13]. This combination offers the richness of the original applied π-calculus while additionally being accessible to state-of-the-art verification tool such as ProVerif [12]. We first syntactically embed the applied π-calculus into our framework. This embedding is particularly instructive because the calculus' syntax vastly differs from the framework's syntax. We then show that computational soundness of the embedding entails computational soundness of the applied π-calculus (in the sense of preservation of trace properties). Second, we provide a computational implementation of the embedding, and we prove it sound within our framework by constructing a suitable simulator. We stress that this result not only implies that our embedding is computationally sound; it also proves the applied π-calculus itself sound under active attacks.

As an example, we used ProVerif to analyze the entity authentication property of the Needham-Schroeder-Lowe protocol. Using the aforementioned results, this yields an implementation of this protocol within the applied π-calculus that is provably secure under active attacks.

1.2 Outline of the paper

Section 2 introduces our framework for computational soundness proofs. Section 3 introduces the notion of a simulator, and it identifies which properties a simulator need to have to entail a computational soundness result. Section 4 contains a case study: how to establish the computational soundness of public-key encryption within the general framework by constructing a suitable simulator. Section 5 establishes the computational soundness of the applied π-calculus. Section 6 concludes and outlines future work.

2 A general framework for computational soundness proofs

2.1 Preliminaries

We first introduce basic notations that are used in this paper, as well as central concepts such as constructors, destructors, and deduction relations.

Notation Given a term t and a substitution φ, we denote by $t\varphi$ the result of applying φ to t. Given a function f, $f(x := y)$ is the function f' with $f'(x) = y$ and $f'(z) = f(z)$ for $z \neq x$. We abbreviate $x_1, \ldots, x_n$ with $\underline{x}$ if n is clear from the context. We call a set M *efficiently decidable* if there is a deterministic polynomial-time algorithm deciding membership in M. We call M *prefix-closed* if $x \in M$ implies $x' \in M$

[2] We currently only consider the preservation of trace properties in the framework. Existing definitions of the preservation of more sophisticated properties such as observational equivalence [16] can be easily cast in our framework. However, deriving a sufficient criterion for computational soundness by means of the existence of a good simulator, see below, requires conceptual future work.

for all prefixes x' of x. A non-negative function f is *negligible* if for every c and sufficiently large n, $f(n) < n^{-c}$. f is *overwhelming* if $1 - f$ is negligible.

Definition 1 (Constructors and destructors) *A* constructor C *is a symbol with an arity. For a (possibly infinite) set of constructors* $\mathbf{C}$*, we denote the set of all terms over these constructors (respecting the arities) by* $T(\mathbf{C})$*. We write* $C/n \in \mathbf{C}$ *to denote that* $\mathbf{C}$ *contains a constructor* C *with arity* n*.*

A destructor D *of arity* n*, written* D/n*, over a set of constructors* $\mathbf{C}$ *is a partial map* $T(\mathbf{C})^n \to T(\mathbf{C})$*. If* D *is undefined on* $\underline{t}$*, we write* $D(\underline{t}) = \bot$*.*

In the following, we only consider sets of constructors $\mathbf{C}$ such that the same constructors cannot have different arities, i.e., $C/n, C/m \in \mathbf{C}$ implies $n = m$. (This restriction simplifies notion and is without loss of generality, as one can simulate multi-arity constructors by adding the arity to the name of the constructor.) We moreover assume that constructors have symbols that are bitstrings, and similarly for destructors and node identifiers in symbolic protocols as introduced below.

A predicate P of arity n over a set of constructors $\mathbf{C}$ is a subset of $T(\mathbf{C})^n$. Since each predicate P can be realized using a destructor D by defining $D(t_1, \ldots, t_n) := t_1$ if $P(t_1, \ldots, t_n) = \mathit{true}$ and $D(t_1, \ldots, t_n) := \bot$ otherwise, predicates however do not require an explicit treatment. Predicates can be used to describe arbitrary tests that a protocol may perform. In particular, they can describe the equality test $=$ which is the diagonal on T^2 for free equational theories and the equivalence relation between terms in non-free equational theories.

We now define which terms can be deduced from other terms; this is formalized using a deduction relation $\vdash$ over a set of terms T. The intuition of $S \vdash m$ for $S \subseteq T$ and $m \in T$ is that the term m can be deduced from the terms in S.

Definition 2 (Deduction relation) *A* deduction relation $\vdash$ *over a set of constructors* $\mathbf{C}$ *is a relation between* $2^{T(\mathbf{C})}$ *and* $T(\mathbf{C})$*.*

In most cases, the adversary can apply all constructors and destructors. This can be modelled by defining $S \vdash \underline{t} \Rightarrow S \vdash C(\underline{t})$ for every constructor C and $S \vdash \underline{t} \wedge D(\underline{t}) \neq \bot \Rightarrow S \vdash D(\underline{t})$ for every destructor D, respectively. However, our model does not assume this in general, i.e., it supports private constructors as used by, e.g., ProVerif.

2.2 Symbolic protocols

We define a symbolic protocol as a tree with a distinguished root and with labels on both nodes and edges. Intuitively, the node labels correspond to different protocol actions: *Constructor and destructor nodes* produce terms (using a constructor or destructor); *communication nodes* correspond to receive and send operations; *nondeterministic nodes* encode nondeterministic choices in the protocol. Moreover, the node labels contain unique identifiers for each node. The edge labels intuitively allow for distinguishing branches in the protocol execution, e.g., destructor nodes have two outgoing edges labelled with yes and no, corresponding to the two cases that the destructor is defined on the input term or not; hence we can, e.g., speak about the yes-successor of a destructor node.

Definition 3 (Symbolic protocol) *A* symbolic protocol *is a tree with a distinguished root and labels on both edges and nodes. Each node has a unique identifier* N *and one of the following types (labels):*

- Constructor nodes *are annotated with a constructor* C/n *together with the identifiers of* n *(not necessarily distinct) nodes. Constructor nodes have exactly one successor; the corresponding edge is labeled with* yes*.*
- Destructor nodes *are annotated with a destructor* D/n *together with the identifiers of* n *(not necessarily distinct) nodes. Destructor nodes have exactly two successors; the corresponding edges are labeled with* yes *and* no*, respectively.*
- Communication nodes *are annotated with the identifier of* $n \geq 0$ *nodes and with a bitstring* l *called out-metadata (out-metadata can be used to, e.g., model additional information leakage to the adversary). A communication node can have countably many successors; the corresponding edges are labeled with bitstrings called in-metadata (in-metadata allows the adversary to control the protocol flow).*
- Nondeterministic nodes *have no further annotation. Nondetermininistic nodes have at least one and at most finitely many successors; the corresponding edges are labeled with bitstrings.*

If a node N *contains an identifier* N' *in its label, then* N' *has to be on the path from the root to* N *(including the root, excluding* N*), and* N' *must be a constructor node, destructor node, or communication node. In case* N' *is a destructor node, then the path from* N' *to* N *has to additionally go through the outgoing edge of* N' *with label* yes*.*

Assigning each nondeterministic node a probability distribution over its successors yields the notion of a probabilistic symbolic protocol.

Definition 4 (Probabilistic symbolic protocol) *A* probabilistic symbolic protocol *is a symbolic protocol, where each nondeterministic node is additionally annotated with a probability distribution on the labels of the outgoing edges.*

In the following, we assume that such a probability distribution is encoded as a list of pairs, consisting of a label

and a rational probability. Any probabilistic symbolic protocol can be transformed canonically into a corresponding symbolic protocol by erasing the probability distributions.

Probabilistic symbolic protocols will be crucial in the definition of computational soundness. Moreover, they often constitute an intermediate technical step within a larger proof. For instance, reasoning about implementations of symbolic protocols is difficult since they do not have a unique such implementation if nondeterministic nodes are present, in contrast to probabilistic symbolic protocols. With the notion of probabilistic symbolic protocols at hand, one can instead consider the set of all implementations of all probabilistic symbolic protocols whose corresponding symbolic protocol is Π.

Definition 5 (Efficient protocol) *We call a probabilistic symbolic protocol* efficient *if:*

- *There is a polynomial p such that for any node N, the length of the identifier of N is bounded by $p(m)$ where m is the length (including the total length of the edge-labels) of the path from the root to N.*
- *There is a deterministic polynomial-time algorithm that, given the identifiers of all nodes and the edge labels on the path to a node N computes the label of N.*

We finally provide the notions of a symbolic model and of a symbolic execution of a protocol.

Definition 6 (Symbolic model) *A* symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{D}, \vdash)$ *consists of a set of constructors $\mathbf{C}$, a subset $\mathbf{N} \subseteq \mathbf{C}$ containing only constructors of arity 0 (called* nonces*), a set of destructors $\mathbf{D}$ over $\mathbf{C}$, and a deduction relation $\vdash$ over $\mathbf{C}$.*

The symbolic execution of a protocol for a given symbolic model consists of a sequence of triples (S, ν, f) where S represents the knowledge of the adversary, ν represents the current node identifier in the protocol, and f represents a partial function mapping already processed node identifiers to messages.

Definition 7 (Symbolic execution) *Let a symbolic model $(\mathbf{C}, \mathbf{N}, \mathbf{D}, \vdash)$ and a symbolic protocol Π be given. A* full trace *is a (finite) list of tuples (S_i, ν_i, f_i) such that the following conditions hold:*

- Correct start*: $S_1 = \varnothing$, ν_1 is the root of Π, f_1 is a totally undefined partial function mapping node identifiers to terms.*
- Valid transition*: For every two consecutive tuples (S, ν, f) and (S', ν', f') in the list, let $\underline{\tilde{\nu}}$ be the node identifiers in the label of ν and $\tilde{t}_k := f(\tilde{\nu}_k)$. We have:*
  - *If ν is a constructor node with constructor C, then $S' = S$, ν' is the successor of ν in Π, and $f' = f(\nu := m)$ for $m := C(\underline{\tilde{t}})$.*
  - *If ν is a destructor node with destructor D, then $S' = S$. Let $m := D(\underline{\tilde{t}})$. If $m \neq \bot$, then ν' is the yes-successor of ν in Π; if $m = \bot$, then ν' is the no-successor of ν in Π. We have $f' = f(\nu := m)$.*
  - *If ν is a communication node, then $S' = S \cup \{\tilde{\underline{t}}\}$, ν' is the successor of ν in Π, and there exists an m with $S' \vdash m$ and $f' = f(\nu := m)$.*
  - *If ν is a nondeterministic node, then $S' = S$, ν' is some successor of ν in Π, and $f' = f$.*

A list of node identifiers (ν_i) is a node trace *if there is a full trace with these node identifiers.*

2.3 Computational model

We now define the computation implementation of a symbolic model as a family of functions that provide computation interpretations to constructors and destructors.

Definition 8 (Computational implementation) *Let a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{D}, \vdash)$ be given. A* computational implementation of $\mathbf{M}$ *is a family of functions $A = (A_x)_{x \in \mathbf{C} \cup \mathbf{D}}$ such that A_C for $C/n \in \mathbf{C} \setminus \mathbf{N}$ is a total deterministic function $\mathbb{N} \times (\{0,1\}^*)^n \to \{0,1\}^*$, A_D for $D/n \in \mathbf{D}$ is a partial deterministic function $\mathbb{N} \times (\{0,1\}^*)^n \to \{0,1\}^*$, and A_N for $N \in \mathbf{N}$ is a probabilistic function with range $\{0,1\}^*$ (i.e., it specifies a probability distribution that depends on its argument).*

All functions A_C, A_D have to be computable in deterministic polynomial-time, and A_N has to be computable in probabilistic polynomial-time.[3]

Requiring A_C and A_D to be deterministic is without loss of generality, since one can always add an explicit randomness argument that takes a nonce as input.

The computational execution of a probabilistic symbolic protocol defines an overall probability distribution on all possible node traces that the protocol proceeds through. In contrast to symbolic executions, we do not aim at defining the notion of a full trace: the adversary's symbolic knowledge S has no formal counterpart in the computational setting, and the function f occuring in the computational executions will not be needed in our later results.

Definition 9 (Computational execution) *Let a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{D}, \vdash)$, a computational implementation A of $\mathbf{M}$, and a probabilistic symbolic protocol Π be given. Let a probabilistic polynomial-time interactive machine E (the adversary) be given (polynomial-time in the sense that the number of steps in all activations are bounded in the*

[3] More precisely, there has to exist a single uniform probabilistic polynomial-time algorithm A that, given the name of C, D, or N, together with an integer k and the inputs $\underline{m}$, computes the output of A_C, A_D, and A_N or determines that the output is undefined. This algorithm must run in polynomial-time in $k + |m|$ and may not use random coins when computing A_C and A_D.

length of the first input of E), and let p be a polynomial. We define a probability distribution $Nodes^p_{M,A,\Pi,E}(k)$ on (finite) lists of node identifiers (ν_i) according to the following probabilistic algorithm (both the algorithm and E are run on input 1^k):

- *Initial state: $\nu_1 := \nu$ is the root of Π. Let f be an initially empty partial function from node identifiers to bitstrings, and n an initially empty partial function from $\mathbf{N}$ to bitstrings.*
- *For $i = 1, \ldots$ do the following:*
  - *Let $\tilde{\underline{\nu}}$ be the node identifiers in the label of ν. Let $\tilde{m}_k := f(\tilde{\nu}_k)$.*
  - *Proceed depending on the type of node ν:*
    - *If ν is a constructor node with constructor $C \in \mathbf{C} \setminus \mathbf{N}$, then $m' := A_C(k, \tilde{\underline{m}})$ let ν' be the successor of ν, and let $f' = f(\nu := m')$. Let $f := f'$ and $\nu := \nu'$.*
    - *If ν is a constructor node with constructor $N \in \mathbf{N}$: Let $m' := n(N)$ if $n(N) \neq \bot$ and sample m' according to $A_N(k)$ otherwise. Let ν' be the successor of ν, $f' := f(\nu := m')$, and $n' := n(N := m')$. Let $\nu := \nu'$, $f := f'$ and $n := n'$.*
    - *If ν is a destructor node with destructor D, then $m' := A_D(k, \tilde{\underline{m}})$. If $m' \neq \bot$, then ν' is the yes-successor of ν, if $m' = \bot$, then ν' is the no-successor of ν. Let $f' := f(\nu := m')$. Let $\nu := \nu'$ and $f := f'$.*
    - *If ν is a communication node labeled with a string l, give $(l, \tilde{\underline{m}})$ to E and get an answer (l', m'). Abort the loop if E halts. Let ν' be the successor of ν along the edge labeled l' (or the lexicographically smallest edge if there is no edge with label l'). Let $f := f(\nu := m')$ and $\nu := \nu'$.*
    - *If ν is a nondeterministic node, let $\mathcal{D}$ be the probability distribution on the label of ν. Pick ν' according to the distribution $\mathcal{D}$, and let $\nu := \nu'$.*
  - *Let $\nu_i := \nu$.*
  - *Let len be the number of nodes from the root to ν plus the total length of all bitstrings in the range of f. If $len > p(k)$, stop.*

2.4 Computational Soundness

We first define trace properties and their fulfillment by a (probabilistic) symbolic protocol. After that, we provide the definition of computational soundness for trace properties.

Definition 10 (Trace property) *A* trace property *$\mathcal{P}$ is an efficiently decidable and prefix-closed set of (finite) lists of node identifiers.*

Let $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{D}, \vdash)$ be a symbolic model and Π' a symbolic protocol. Then Π' symbolically satisfies *a trace property $\mathcal{P}$ in $\mathbf{M}$ iff every node trace of Π' is contained in $\mathcal{P}$. Let A be a computational implementation of $\mathbf{M}$ and let Π be a probabilistic symbolic protocol. Then* (Π, A) computationally satisfies *a trace property $\mathcal{P}$ in $\mathbf{M}$ iff for all probabilistic polynomial-time interactive machines E and all polynomials p, the probability is overwhelming that $Nodes^p_{\mathbf{M},A,\Pi,E}(k)$ is contained in $\mathcal{P}$.*

Definition 11 (Computational soundness) *A computational implementation A of a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{D}, \vdash)$ is* computationally sound *for a class P of symbolic protocols iff for every trace property $\mathcal{P}$ and for every efficient probabilistic symbolic protocol Π, we have that (Π, A) computationally satisfies $\mathcal{P}$ if the corresponding symbolic protocol Π' of Π symbolically satisfies $\mathcal{P}$ and $\Pi' \in P$.*

3 On simulators that entail computational soundness proofs

In this section, we introduce the notion of a simulator and identify several properties a simulator might enjoy. We show that the existence of a simulator that enjoys all of these properties already suffices to establish computational soundness in the sense of Definition 11. Future soundness proofs can thus concentrate on the construction of a suitable simulator.

In the following, we fix a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{D}, \vdash)$ and a computational implementation A of $\mathbf{M}$. Let $T_x(\mathbf{C})$ denote the set of terms over the constructors $\mathbf{C}$ that may contain variables, i.e, $T(\mathbf{C})$ are the ground terms in $T_x(\mathbf{C})$. Similarly, let $T_x(\mathbf{C}, \mathbf{D})$ denote the set of terms over constructors $\mathbf{C}$ and destructors $\mathbf{D}$ that contain variables, and let $T(\mathbf{C}, \mathbf{D})$ denote the corresponding ground terms. By the definition of destructors, any $t \in T(\mathbf{C}, \mathbf{D})$ evaluates to some $t' \in T(\mathbf{C}) \cup \{\bot\}$. We write $\mathrm{eval}(t)$ to denote this term t'. In the following, we moreover assume that whenever a machine sends a term or a node, the term / node is suitably encoded as bitstring.

Definition 12 (Question, answer, valid substitution) *A* question *is a term in $T_x(\mathbf{C}, \mathbf{D})$. An* answer *$A$ to a question Q is a term $Q = b$ where $b \in \{true, false\}$. An answer A is* correct *for a question $Q \in T(\mathbf{C}, \mathbf{D})$ iff $b = (\mathrm{eval}(Q) \neq \bot)$. A substitution φ from variables to $T(\mathbf{C})$ is* valid for a set $\mathbf{A}$ of answers *if $A\varphi$ is correct for all $A \in \mathbf{A}$.*

We proceed by introducing the notion of a simulator, essentially by imposing syntactic constraints on the set of all interactive machines.

Definition 13 (Simulator) *A* simulator *is an interactive machine Sim that satisfies the following syntactic requirements:*

- *If it is activated with a question Q, it sends an answer to Q.*

- *If it is activated with a term $t \in T_x(\mathbf{C}, \mathbf{D})$, it replies with a term $m \in T_x(\mathbf{C}, \mathbf{D})$.*
- *If it is activated with $(info, \nu, t)$ where ν is a node identifier and $t \in T(\mathbf{C})$, it either replies with $(proceed)$, or with $(terminate)$.*
- *At any point (in particular instead of sending a reply), it may terminate and output either $fail$ or a substitution φ from variables to $T(\mathbf{C})$.*

A simulator Sim is intuitively expected to constitute a translation routine that transforms a computational attack into a corresponding symbolic attack. Thus Sim essentially translates bitstrings to terms, and vice versa. Granting Sim the ability to process terms with variables (i.e., unknown subterms) frees Sim from providing a final translation at the time a term is sent; instead, Sim can lazily complement translations if it is subsequently queried with a corresponding question when the value of such a variable determines the flow of the protocol.

We proceed by defining the hybrid execution of a probabilistic symbolic protocol. We call this execution hybrid because it is a mixture of the symbolic and the computational execution. Roughly, we define a hybrid protocol machine Π^C that is associated to Π. Intuitively, Π^C behaves as Π but incoming communication terms are allowed to have variables that Π^C lazily instantiates by asking questions to a simulator.

Definition 14 (Hybrid execution) *Let Π be a probabilistic symbolic protocol, and let Sim be a simulator. We define a probability distribution $H\text{-}Trace_{\mathbf{M},\Pi,Sim}(k)$ on (finite) lists of tuples (S_i, ν_i, f_i) called the* full hybrid trace *according to the following probabilistic algorithm Π^C, run on input 1^k, that interacts with Sim. (Π^C is called the hybrid protocol machine associated with Π and internally runs a symbolic simulation of Π as follows:)*

- Start*: $S_1 := S := \varnothing$, $\nu_1 := \nu$ is the root of Π, and $f_1 := f$ are totally undefined partial functions mapping node identifiers to $T_x(\mathbf{C}, \mathbf{D})$. Run Π on ν.*
- Transition*: For $i = 1, \ldots$ do the following:*
  - *Let $\tilde{\underline{\nu}}$ be the node identifiers in the label of ν. Let $\tilde{m}_k := f(\tilde{\nu}_k)$.*
  - *Proceed depending on the type of ν:*
    - *If ν is a constructor node with constructor C, let $m := C(\tilde{\underline{t}})$, ν' be the successor of ν, and $f' := f(\nu := m)$. Let $\nu := \nu'$ and $f := f'$.*
    - *If ν is a destructor node with destructor D, then let $m := D(\tilde{\underline{t}})$. If m is ground, then let ν' be the yes- or no-successor of ν, depending on whether $\mathrm{eval}(m) \neq \bot$ or not, and let $f' := f(\nu := \mathrm{eval}(m))$. If m is not ground, then ask Sim the question m. If the answer is $m = true$, let $f' := f(\nu := m)$ and let ν' be the yes-successor of ν. Otherwise let ν' be the no-successor of ν and $f' := f$. Let $\nu := \nu'$ and $f := f'$.*
    - *If ν is a communication node with out-metadata l, send $(l, \tilde{\underline{t}})$ to Sim. Upon receiving (l', m) with $l \in \{0,1\}^*$ and $m \in T_x(\mathbf{C}, \mathbf{D})$ from Sim, let $f' := f(\nu := m)$, let ν' be the successor of ν with in-metadata l' (or the lexicographically smallest successor, if l' does not occur), and let $S' := S \cup \{m\}$. Let $S := S'$, $\nu := \nu'$, and $f := f'$.*
    - *If ν is a nondeterministic node, sample ν' according to the probability distribution specified in ν. Let $\nu := \nu'$.*
  - *Send $(info, \nu, t)$ to Sim. When receiving an answer $(proceed)$ from Sim, continue. When receiving $(terminate)$, stop.*
  - *If Sim has output a substitution φ or $fail$, stop. Otherwise let $(S_i, \nu_i, f_i) := (S, \nu, f)$.*

We write $Sim + \Pi^C$ to denote the execution of Sim and Π^C. We denote the probability on node traces of this execution by $H\text{-}Nodes_{\mathbf{M},\Pi,Sim}(k)$. By $H\text{-}Subst_{\mathbf{M},\Pi,Sim}(k)$ we denote the probability on the substitution that is output by Sim at the end of this execution (or $\bot$ if Sim outputs $fail$). By $H\text{-}Answers_{\mathbf{M},\Pi,Sim}(k)$ we denote the probability on the list of answers sent by Sim in the execution.

We proceed by defining several properties of a simulator, such as never outputting $fail$ and thus causing the hybrid execution to abort, correctly answering all queries, or adhering to a Dolev-Yao style deduction relation. Later we will show that simulators that satisfy all these properties entail computational soundness results. Treating these properties separately instead of immediately conjoining them into a general soundness criterion allows us to more careful identify where these individual properties are exploited in computational soundness proofs.

The first property – abort-freeness – ensures that Sim only outputs $fail$ with negligible probability.

Definition 15 (Abort-free simulator) *A simulator Sim is* abort-free *for $\mathbf{M}$ and Π, if the probability that Sim outputs $fail$ in the hybrid execution $Sim + \Pi^C$ is negligible.*

The next property – consistency – captures that the node traces obtained by interacting with Sim adhere to the rules from Definition 7 that determine the successor node. More precisely, whenever Sim answers a question that will determine the successor node in a hybrid execution, this answer will be consistent with the rules of Definition 7 for the full hybrid trace (with substituted variables).

Definition 16 (Consistent simulator) *A simulator Sim is* consistent *for $\mathbf{M}$ and Π, if with overwhelming probability there exists a substitution φ that is valid for $H\text{-}Answers_{\mathbf{M},\Pi,Sim}(k)$.*

The next property – Dolev-Yao-style – captures that Sim adheres to the deduction relation $\vdash$ in Definition 7 for communication nodes. More precisely, the communication terms

that Sim sends *to* the symbolic protocol have to be derivable from Sim's symbolic view so far. (However, this is formally only determined *after* Sim's substitution φ is applied.)

Definition 17 (Dolev-Yao style simulator) *A simulator Sim is* Dolev-Yao style *(short:* DY*) for $\mathbf{M}$ and Π, if with overwhelming probability the following holds:*

Whenever Sim outputs a valid substition φ for $H\text{-}Answers_{\mathbf{M},\Pi,Sim}(k)$ in a given execution of $Sim + \Pi^C$, let t_ℓ be the ℓ-th term sent from Π^C to Sim during the processing of a communication node in a given execution of $Sim + \Pi^C$, and let m_ℓ be the response sent from Sim to Π^C in that execution (i.e., the term sent from Sim to Π^C directly after receiving t_ℓ).

Then for all ℓ, we have $t_1\varphi, \ldots, t_\ell\varphi \vdash m_\ell\varphi$.

The final property – indistinguishability – captures that the hybrid node traces are computationally indistinguishable from real node traces, i.e., the corresponding random variables cannot be distinguished by any probabilistic algorithm that runs in polynomial time in the security parameter. We write $\overset{c}{\approx}$ to denote computational indistinguishability.

Definition 18 (Indistinguishable simulator) *A simulator Sim is* indistinguishable *for $\mathbf{M}$, Π, an implementation A, an adversary E, and a polynomial p, if*

$$Nodes^p_{\mathbf{M},A,\Pi,E}(k) \overset{c}{\approx} H\text{-}Nodes_{\mathbf{M},\Pi,Sim}(k),$$

i.e., if the node trace and the hybrid node trace are computationally indistinguishable.

We define the following abbreviation.

Definition 19 (Good simulator) *A simulator is* good *for $\mathbf{M}$, Π, A, E, and p if it is abort-free, consistent, and Dolev-Yao style for $\mathbf{M}$, and Π,and indistinguishable for $\mathbf{M}$, Π, A, E, and p.*

We can now formally state and prove the main result of this section: the existence of a good simulator implies computational soundness.

Theorem 1 (Good simulator implies soundness) *Let $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{D}, \vdash)$ be a symbolic model, let P be a class of symbolic protocols, and let A be a computational implementation of $\mathbf{M}$. Assume that for every efficient probabilistic symbolic protocol Π (whose corresponding symbolic protocol is in P), every probabilistic polynomial-time adversary E, and every polynomial p, there exists a good simulator for $\mathbf{M}$, Π, A, E, and p. Then A is computationally sound for protocols in P.*

Proof. We have to show that for every probabilistic symbolic protocol Π, we have that (Π, A) computationally satisfies $\mathcal{P}$ whenever Π' symbolically satisfies a property $\mathcal{P}$ (where Π' is the corresponding symbolic protocol of Π). Thus, for every E and p, $Nodes^p_{\mathbf{M},A,\Pi,E}(k)$ has to be contained in $\mathcal{P}$ with overwhelming probability. Fix Π, E, and p, and let Sim be a good simulator for $\mathbf{M}$, Π, A, E, and p. Let $A_\mathcal{P}$ denote a polynomial-time algorithm that decides property $\mathcal{P}$.

We first show a lemma on the hybrid node traces and then proceed with the overall proof; the proof of the lemma is postponed to Appendix C.

Lemma 1 *Consider a hybrid execution of $Sim + \Pi^C$ in which Sim is abort-free, consistent and DY, i.e.,*

- *Sim does not output $fail$,*
- *Sim finally outputs a substition φ that is valid for Sim's answers in this execution,*
- *Sim behaves as a Dolev-Yao adversary, i.e., we have $\{t_1, \ldots, t_\ell\} \vdash m_\ell$ for all t_i and m_ℓ as in Definition 17 and all ℓ.*

Let tr be the full hybrid trace of that execution. Then $tr' := tr\varphi$ is a full symbolic trace of Π'.

Lemma 1 immediately entails that the probability is overwhelming that $H\text{-}Nodes_{\mathbf{M},\Pi,Sim}(k)$ is a symbolic node trace of Π', and hence that $H\text{-}Nodes_{\mathbf{M},\Pi,Sim} \in \mathcal{P}$. Since $A_\mathcal{P}$ decides $\mathcal{P}$, this means that

$$\mathsf{Pr}\left[A_\mathcal{P}(H\text{-}Nodes_{\mathbf{M},\Pi,Sim}(k)) = 1\right] \text{ is overwhelming.} \quad (1)$$

By Sim's indistinguishability property, we know that

$$Nodes^p_{\mathbf{M},A,\Pi,E}(k) \overset{c}{\approx} H\text{-}Nodes_{\mathbf{M},\Pi,Sim}(k).$$

Since $A_\mathcal{P}$ is polynomial-time in its input, and $Nodes^p_{\mathbf{M},A,\Pi,E}(k)$ is polynomially-sized in k by construction, this implies that

$$\mathsf{Pr}\left[A_\mathcal{P}(Nodes^p_{\mathbf{M},A,\Pi,E}(k)) = 1\right] \quad \text{is overwhelming,}$$

and hence that $Nodes^p_{\mathbf{M},A,\Pi,E}(k) \in \mathcal{P}$ with overwhelming probability. This concludes the proof of Theorem 1. □

4 Case study: computational soundness of public-key encryption

In this section, we provide a symbolic model that allows for expressing encryption, decryption and pairs, and we derive criteria under which a computational execution of that model is computationally sound.

The symbolic model. We first specify the symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{D}, \vdash)$:

- Constructors: We have $\mathbf{C} := \{E/3, pk/1, sk/1, pair/2, garbage/1, garbageE/2\} \cup \mathbf{N}$ with $\mathbf{N} = \mathbf{N}_P \cup \mathbf{N}_E$. Here $\mathbf{N}_P$ and $\mathbf{N}_E$ are countably infinite sets representing protocol and adversary nonces, respectively. Intuitively, $E(pk(r'), m, r)$ encrypts m using the public key $pk(r')$ and randomness r. $garbage$ and $garbageE$ are constructors necessary to express certain invalid terms the adversary may send.

- Destructors: $\mathbf{D} := \{ispk/1, isenc/1, D/2, fst/1, snd/1, pkof/1, equals/2\}$. The destructors $ispk$ and $isenc$ realize predicates to test whether a term is a public key or a ciphertext, respectively. $pkof$ extracts the public key from a ciphertext. $D(sk(r), c)$ decrypts the ciphertext c.
  The behavior of the destructors is given by the following rules; an application matching none of these rules evaluates to $\perp$:

$$\begin{aligned} D(sk(t_1), E(pk(t_1), m, t_2)) &= m \\ ispk(pk(t)) &= pk(t) \\ isenc(E(pk(t_1), t_2, t_3)) &= E(pk(t_1), t_2, t_3) \\ isenc(garbageE(pk(t_1), t_2)) &= garbageE(pk(t_1), t_2) \\ fst(pair(x, y)) &= x \\ snd(pair(x, y)) &= y \\ pkof(E(pk(t_1), m, t_2)) &= pk(t_1) \\ pkof(garbageE(pk(t_1), t_2)) &= pk(t_1) \end{aligned}$$

- Deduction relation: $\vdash$ is the smallest relation such that $m \in S \Rightarrow S \vdash m$, $N \in \mathbf{N}_E \Rightarrow S \vdash N$, and such that for any constructor or destructor $f \notin \mathbf{C} \cup \mathbf{D} \setminus \mathbf{N}$ and for any $t_1, \ldots, t_n$ with $S \vdash \underline{t}$ and $f(\underline{t}) \neq \perp$ we have $S \vdash f(\underline{t})$.

The computational implementation. Obtaining a computational soundness result for the symbolic model $\mathbf{M}$ requires its implementation to use an IND-CCA2 secure encryption scheme. More precisely, we require that (A_{pk}, A_{sk}), A_E, and A_D form the key generation, encryption and decryption algorithm of an IND-CCA2-secure scheme. Let $A_{ispk}(m) = m$ and $A_{isenc}(m) = m$ iff m is a public key or a ciphertext, respectively. (Only a syntactic check is performed; it is not necessary to check whether m was correctly generated.) A_{pkof} extracts the public key from a ciphertext, i.e., we assume that ciphertexts are tagged with their public key. Nonces are implemented as random k-bit strings. A_{pair}, A_{fst}, and A_{snd} construct and destruct pairs. We require that the implementation of the constructors are length regular, i.e., the length of the result of applying a constructor depends only on the lengths of the arguments. No restrictions are put on $A_{garbage}$ and $A_{garbageE}$ as these are never actually used.

Protocol conditions. The computational soundness result we derive in this section requires that the symbolic protocol satisfies certain constraints. In a nutshell, these constraints require that encryption and key generation always use fresh randomness, that decryption only uses honestly generated secrets keys, and that the protocol does not produce garbage terms. We call protocols satisfying these conditions *encryption-safe*. For an exact characterization of the assumptions concerning the implementation and of the protocol conditions, we refer to Appendix D.

Theorem 2 *A is a computationally sound implementation of $\mathbf{M}$ for encryption-safe protocols.*

Proof sketch: We construct a simulator Sim that internally runs the adversary E and forwards the messages between E and the protocol to the protocol Π^C in the hybrid model. Sim translates all terms sent by the protocol into bitstrings by evaluating all constructors C using their implementation A_C. The values of protocol nonces are chosen by Sim. In the other direction, Sim converts bitstrings to terms by applying the destructors A_D, A_{fst}, A_{snd}, and A_{pkof}. Since Sim chooses the values of all protocol nonces, it knows the secret keys needed to parse ciphertexts encrypted with respect to honestly generated public keys. Ciphertexts c with respect to other public keys are considered as invalid encryptions $garbageE(\ldots)$. This is possible since the protocol only attempts to decrypt using honestly generated secret keys. Since the translation between bitstrings and terms is done using the implementation of constructors and destructors, Sim is indistinguishable. Consistency and abort-freeness follow directly from the construction. To show that Sim is DY, we construct another simulator Sim_f that behaves like Sim but uses fake encryptions: Instead of applying A_E to the plaintext m, it instead applies it to an all-zero string of the same length. From the IND-CCA property we get that the executions of Sim and Sim_f are indistinguishable. We show that if Sim_f is not DY, then a certain term t_{bad} is sent by Sim_f. The term t_{bad} is shown to contains a nonce that is never accessed (as Sim_f does not have to compute the plaintexts of encryptions) and hence the bitstring m_{bad} corresponding to t_{bad} is information-theoretically hidden. However, m_{bad} is easy to extract from the message sent by E using destructor applications. Hence we have a contradiction and Sim_f is DY. Since the executions of Sim_f and Sim are indistinguisable, Sim is DY. Thus Sim is good and the theorem follows. A detailed proof is given in Appendix D. □

5 Computational soundness of the applied π-calculus

In this section we show how to use our framework to establish the first computational soundness result for the applied π-calculus. Strictly speaking, we consider the process calculus proposed in [14] additionally augmented with events. The calculus in [14] itself is a combination of the original applied pi-calculus [1] with one of its dialects [13]. This combination offers the richness of the original applied pi-calculus while additionally being accessible by ProVerif [12]. The embedding of this calculus into our general formal model is particularly instructive because the calculus' syntax vastly differs from the framework's syntax, e.g., the applied π-calculus models secrecy of nonces via restrictions, it does not rely on a labeled transition system, but it considers an equational theory.

$M, N ::=$	terms
x, y, z	variables
a, b, c	names
$f(M_1, \ldots, M_n)$	constructor application
$D ::=$	destructor terms
M	terms
$d(D_1, \ldots, D_n)$	destructor application
$f(D_1, \ldots, D_n)$	constructor application
$P, Q ::=$	processes
$\bar{M}\langle N\rangle.P$	output
$M(x).P$	input
0	nil
$P \mid Q$	parallel composition
$!P$	replication
$\nu a.P$	restriction
$let\ x = D$ $in\ P\ else\ Q$	destructor application
$event(e).P$	event

Figure 1. Syntax of the process calculus.

5.1 Overview of this section

We briefly outline the structure of this section, since it can be seen as a general guideline on how to embed other calculi into our framewoek, and how to derive computational soundness guarantees for them.

We first review the syntax and the semantics of the applied π-calculus in Section 5.2. In Section 5.3, we define a computational execution of the calculus (this is only necessary since the applied π-calculus does not come with an a-priori defined computational execution), called computational π-execution, as well as trace properties in the applied π-calculus, called π-trace properties. In Section 5.4, we establish the actual soundness result using our framework: We first define a symbolic model of the applied π-calculus (in the sense of Definition 6) as well as a computational interpretation of this model (in the sense of Definition 8). The final theorem then asserts that if this computational implementation is computationally sound with respect to this symbolic model, then every π-calculus process that symbolically fulfills a π-trace property also computationally fulfills this property.

5.2 Review of the calculus' syntax and semantics

The syntax of the process calculus that we consider is provided in Figure 1. (We do not explicitly include an if-statement, but instead emulate it using destructor applications, see below.) Technically, it corresponds to the one considered in [14], except that we add processes of the form $event(e).P$ for a string e. The intuitive meaning of such a process is that it raises an event e and then proceeds to execute P.

In the following, we often call terms in the process calculus π-terms and terms in the framework, i.e., in the sense of Section 2.2 fw-terms, in order to avoid ambiguities. We proceed similarly for other homonyms, such as π-constructors, π-traces, etc. The set of ground π-terms is denoted T_π. By $fn(P)$ we denote the set of free names of P, i.e., the names n not protected by a restriction. By $fv(P)$ we denote the free variables of P, i.e., the variables that are not protected by a destructor application or an input. We call a process closed if it has no free variables (but it may have free names).

The calculus is parametrized over a (possibly infinite) set of π-constructors $\mathbf{C}_\pi$, a (possibly infinite) set of π-destructors $\mathbf{D}_\pi$, and an equivalence relation $\approx$ over ground π-terms (called the equational theory). A destructor d of arity n is a partial function $T_\pi^n \to T_\pi$. We require that the equational theory is compatible with the π-destructors and π-constructors in the following sense: For all π-constructors f and π-destructors d of arity n, for all ground π-terms $M_1, \ldots, M_n, M_1', \ldots, M_n'$ with $M_i \approx M_i'$ for $i = 1, \ldots, n$, we have that $f(\underline{M}) \approx f(\underline{M'})$, that $d(\underline{M}) = \bot$ iff $d(\underline{M'}) = \bot$, and that $d(\underline{M}) \approx d(\underline{M'})$. We also require $d(\underline{M}\tau) = d(\underline{M})\tau$ for any renaming τ of names.

We did not explicitly include an if-statement in the syntax of the calculus since such a statement can be expressed using an additional destructor $equals$: Let $equals(x, y) = x$ for $x \approx y$ and define $if\ M = N\ then\ P\ else\ Q$ as $let\ x = equals(M, N)\ in\ P\ else\ Q$ for some $x \notin fv(P)$. In the following, we will assume $equals \in \mathbf{D}_\pi$.

Given a ground destructor π-term D, we can evaluate it to a ground π-term $\mathrm{eval}^\pi D$ by evaluating all π-destructors. If one of the π-destructors returns $\bot$, we set $\mathrm{eval}^\pi D := \bot$.

The semantics of the calculus is standard and corresponds to the one defined in [14] except for the addition of events. The semantics hence consists of two possible transitions: $\to$ and $\xrightarrow{e}$. The latter denotes that the event e occurred, and we can define trace properties as properties over the sequence of events occurring in an execution of a process. Again, we prefix some notions with π to distinguish them from their corresponding notions in Section 2.2. The semantics is formally defined in Figure 4 in Appendix A.

Definition 20 (π-Trace properties) *A list of strings $e_1, \ldots, e_n$ is an* event trace *of P if there is a process Q that does not contain events such that $P \mid Q \to^* \xrightarrow{e_1} \to^* \xrightarrow{e_2} \to^* \cdots \to^* \xrightarrow{e_n}$. A* π-trace property

is an efficiently decidable and prefix-closed set of strings. A process P symbolically satisfies a π-trace property $\wp$ *if we have $e \in \wp$ for all event traces e of P.*

5.3 Defining a computational execution

A computational π-implementation assigns a total deterministic polynomial-time algorithm A_f^π to each π-constructor f, and a partial deterministic polynomial-time algorithm A_d^π to each π-destructor d. The formal definition is fully analogous to Definition 8, except that we do not require an implementation for nonces (they will be chosen uniformly at random); we hence omit this definition due to space constraints. We require that $A^\pi_{equals}(1^k, x, x) = x$ and $A^\pi_{equals}(1^k, x, y) = \bot$ for $x \neq y$ (i.e., the computational interpretation of $\approx$ is the identity on bitstrings). Given an assignment μ from names to bitstrings and an assignment η from variables to bitstrings for names and variables occurring in a destructor term D, we can (computationally) evaluate D to a bitstring $\mathrm{ceval}_{\eta,\mu} D$. (Formally, the security parameter k is an additional input to ceval, but we omit k for readability). We set $\mathrm{ceval}_{\eta,\mu} D := \bot$ if the application of one of the algorithms A_d^π fails.

Given a computational implementation of the constructors and destructors, the computational execution of a process P is already determined, except for resolving nondeterminism and which messages the adversary is allowed to observe. To resolve the non-determinism in the calculus, we uniformly randomly select the next action to take. This is the conceptually simplest approach and helps to increases the readability of the proof of computational soundness of the applied π-calculus. We stress that our proof does not exploit this specific way of resolving nondeterminism, but that the proof can easily be adapted to more sophisticated scheduling mechanisms. Furthermore, we have to reflect that the calculus allows the adversary to receive messages on any channel in his knowledge. Since the knowledge of the adversary is not well-defined in a computational setting, we cannot model this directly. Instead, we require that the adversary explicitly registers for any channel c he wants to eavesdrop on, by sending the message $(listen, c)$.

The computational implementation of a process is then defined using evaluation contexts: An evaluation context is a context with either one hole, or with two (distinguished) holes. In the case of two holes, we write $E[P][Q]$ to denote the replacement of the first hole by P and of the second hole by Q.

Definition 21 (Step contexts and input contexts) *Let P be a process, η a function from variables to bitstrings, μ a function from names to bitstrings, and pub a set of bitstrings. An evaluation context E is a* step context *for P if one of the following structural conditions holds true:*

- $P = E[\nu a.P_1]$,
- $P = E[\overline{M_1}\langle N\rangle.P_1][M_2(x).P_2]$ *with* $\mathrm{ceval}_{\eta,\mu} M_1 = \mathrm{ceval}_{\eta,\mu} M_2$,
- $P = E[let\ x = D\ in\ P_1\ else\ P_2]$,
- $P = E[event(e).P_1]$,
- $P = E[!P_1]$,
- $P = E[\overline{M}\langle N\rangle.P_1]$ *with* $\mathrm{ceval}_{\eta,\mu} M \in pub$.

An input context *for P is an evaluation context E such that $P = E[M(x).P_1]$ and $\mathrm{ceval}_{\eta,\mu} M = c$.*

Intuitively, a step context specifies all possible actions of a process that do not involve input from the adversary, assuming that the adversary can listen only on channels in pub. We consider the restriction $\nu a.P_1$ as an executable action in the computational setting because it involves choosing a random bitstring for the corresponding nonce. Input contexts represent all possible positions in the process that can receive inputs on a channel c.

The computational π-execution of a process is now defined as an interactive machine that executes the process and communicates with an adversary.

Definition 22 (Computational π-execution) *Let P_0 be a closed process, and let C be an interactive machine called the adversary. We define the computational π-execution as an interactive machine $\mathrm{Exec}_{P_0}(1^k)$ that takes a security parameter k as argument and interacts with C:*

- Start*: Let $P := P_0$ (where we rename all bound variables and names such that they are pairwise distinct and distinct from all unbound ones). Let η be a totally undefined partial function mapping variables to bitstrings, let μ be a totally undefined partial function mapping names to bitstrings, and let pub be an empty set of bitstrings. Let $a_1, \ldots, a_n$ denote the free names in P_0. For each i, pick $r_i \in \{0,1\}^k$ at random. Set $\mu := \mu(a_1 := r_1, \ldots, a_n := r_n)$. Send $(public, r_1, \ldots, r_n)$ to C.*[4]
- Transition*: Proceed depending on the type of message received from C as follows:*
  - *When receiving $(listen, c)$ from C, set $pub := pub \cup \{c\}$ and send (ok) to C.*
  - *When receiving $(input, c, m)$ from C, choose an input context E for (P, η, μ, c) uniformly at random. If no such input context exists, send $(stuck)$ to C. If such an input context exists (where P then is of the form $E[M(x).P_1]$), set $\eta := \eta(x := m)$, $P := E[P_1]$, and send (ok) to C.*
  - *When receiving $(step)$ from C, choose a step context E for (P, η, μ, pub) uniformly at random. If no step context exists, send $(stuck)$ to C. Otherwise, proceed as follows depending on the structure of P:*
    - *$P = E[\nu a.P_1]$: pick $r \in \{0,1\}^k$ at random, set $P := E[P_1]$ and $\mu := \mu(a := r)$. Send (ok) to C.*
    - *$P = E[\overline{M_1}\langle N\rangle.P_1][M_2(x).P_2]$: Set $P := E[P_1][P_2]$ and $\eta := \eta(x := \mathrm{ceval}_{\eta,\mu} N)$. Send (ok) to C.*

[4] In the applied π-calculus, free names occurring in the initial process represent nonces that are honestly chosen but known to the attacker.

  - $*$ $P = E[let\ x = D\ in\ P_1\ else\ P_2]$*: If* $m := \mathrm{ceval}_{\eta,\mu} D \neq \bot$*, set* $\eta := \eta(x := m)$ *and* $P := E[P_1]$*; Otherwise set* $P := E[P_2]$*. Send* (ok) *to* C*.*
  - $*$ $P = E[event(e).P_1]$*: Let* $P := E[P_1]$ *and send* $(event, e)$ *to* C*.*
  - $*$ $P = E[!P_1]$*: Rename all bound variables of* P_1 *such that they are pairwise distinct and distinct from all variables and names in* P *and in the domains of* η *and* μ*, yielding a process* $\tilde{P}_1$*. Set* $P := E[\tilde{P}_1 \mid !P_1]$*. Send* (ok) *to* C*.*
  - $*$ $P = E[\overline{M}\langle N\rangle.P_1]$*: Set* $P := E[P_1]$*. Send* $(output, \mathrm{ceval}_{\eta,\mu} M, \mathrm{ceval}_{\eta,\mu} N)$ *to* C*.*
- *When receiving anything else, send* $(stuck)$ *to* C*.*

The execution of $\mathrm{Exec}_{P_0}(1^k)$ maintains the invariant that all bound variables and names in P are pairwise distinct and that they are distinct from all variables and names in P and in the domains of η and μ. Moreover, all events occuring in the process can be extracted from the messages sent by Exec_{P_0}. For a given polynomial-time interactice machine C, a closed process P_0, and a polynomial p, we let $Events_{C,P,p}(k)$ denote the list of the strings e output by $\mathrm{Exec}_{P_0}(1^k)$ (as part of $(event, e)$-messages) within the first $p(k)$ computation steps (jointly counted for $C(1^k)$ and $\mathrm{Exec}_{P_0}(1^k)$).

We finally define the computational fulfillment of π-trace properties.

Definition 23 (Computational π-trace properties) *Let E be a polynomial-time interactive machine, P_0 a closed process, and p a polynomial. We say that P_0* computationally satisfies *a π-trace property $\wp$ if for all polynomial-time interactive machines C and all polynomials p, we have that $\Pr[Events_{C,P,p}(1^k) \in \wp]$ is overwhelming in k.*

5.4 Computational soundness of the calculus

We will now derive the computational soundness of the applied π-calculus, i.e., we will show that if its computational implementation is computationally sound in the sense of Definition 11, then every symbolically satisfied π-trace property is also computationally satisfied. Applying Definition 11 first requires us to specify a symbolic model of the applied π-calculus (in the sense of Definition 6) and a computational implementation of this model (in the sense of Definition 8).

The symbolic model of the applied π-calculus contains all the π-constructors and π-destructors from the process calculus. We additionally add an infinite number of adversary nonces $\mathbf{N}_E$ and protocol nonces $\mathbf{N}_P$ to represent free and bound names. The deduction relation allows the adversary to derive all adversary nonces and everything derivable by application of constructors and destructors.

Definition 24 (Symbolic model of the applied π-calculus) *For a π-destructor d, we define d' by $d'(\underline{t}) := d(\underline{t}\rho)\rho^{-1}$ where ρ is an injective map from the nonces in $\underline{M}$ to names.*[5] *Let $\mathbf{N}_E$ and $\mathbf{N}_P$ be countably infinite sets.*

The symbolic model of the applied π-calculus *is given by $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{D}, \vdash)$, where $\mathbf{N} := \mathbf{N}_E \cup \mathbf{N}_P$, $\mathbf{C} := \mathbf{C}_\pi \cup \mathbf{N}$, $\mathbf{D} := \{d' : d \in \mathbf{D}_\pi\}$, and where $\vdash$ is defined by the rules in Figure 2.*

In the following, we consider $\mathbf{M}, \mathbf{C}, \mathbf{N}, \mathbf{D}, \vdash$ as in Definition 24. In particular, the destructor $equals'$ thus induces an equivalence relation $\cong$ on the set of fw-terms by $x \cong y$ iff $equals'(x, y) \neq \bot$. The relation $\cong$ is the analogue to the equational theory $\approx$.

The computational implementation of this symbolic model is now specified by the computational π-implementations A_f and A_d of the π-constructors and π-destructors, with nonces being chosen uniformly at random.

Definition 25 (Computational implementation of Def. 24) *The computational implementation A of the symbolic model of the applied π-calculus $\mathbf{M}$ is given by $A_f := A^\pi_f$ for all $f \in \mathbf{C} \setminus \mathbf{N}$ and $A_d := A^\pi_d$ for all $d \in \mathbf{D}$. A_N for $N \in \mathbf{N}$ picks $r \in \{0,1\}^k$ uniformly at random and returns r.*

In order to relate the symbolic and the computational semantics of a process, we define an additional symbolic execution for closed processes as a technical tool. This new semantics constitutes a safe approximation of the original semantics of the process calculus while at the same time being a direct analogue of the computational semantics presented in Definition 22. The semantics is defined by means of an interactive non-deterministic machine SExec_{P_0}, analogous to the machine Exec_{P_0} from Definition 22. Intuitively, the only difference between Exec_{P_0} and SExec_{P_0} is that the latter operates immediately on terms whenever the former operates on computational implementations of these terms. We postpone the formal definition of SExec_{P_0} as well as further explanatory comments to Appendix B. There, it is also shown that the machine SExec_{P_0} can be realized as a symbolic protocol in the sense of Definition 3 by encoding protocol steps as nodes; we call this protocol Π_{P_0}. We obtain a probabilistic symbolic protocol Π'_{P_0} by annotating each non-deterministic node in Π_{P_0} with the uniform distribution on its successors. The nodes in Π_{P_0} (and Π'_{P_0}) that output $(event, e)$ for some string e, we call event nodes and say that they raise the event e. (Since both $event$ and e are hard-coded in the node, this is well-defined.) For a sequence of node identifiers ν, let $events(\nu)$ denote the sequence of the events raised by the event nodes in ν.

Definition 26 *A non-deterministic interactive machine C is a* Dolev-Yao adversary *if the following holds in an interaction with any interactive machine M in each step of the interaction: Let S be the set of all* fw*-terms sent by M up to*

[5] This is well-defined and independent of ρ since for any renaming of names τ, we have $d(\underline{M}\tau) = d(\underline{M})\tau$; intuitively d' behaves as d except that it uses nonces instead of names.

$$\frac{m \in S}{S \vdash m} \qquad \frac{N \in \mathbf{N}_E}{S \vdash N} \qquad \frac{S \vdash \underline{M} \qquad f \in \mathbf{C} \setminus \mathbf{N}}{S \vdash f(\underline{M})} \qquad \frac{S \vdash \underline{M} \qquad d \in \mathbf{D} \qquad d(\underline{M}) \neq \bot}{S \vdash d(\underline{M})}$$

Figure 2. Deduction rules for the symbolic model of the applied π-calculus

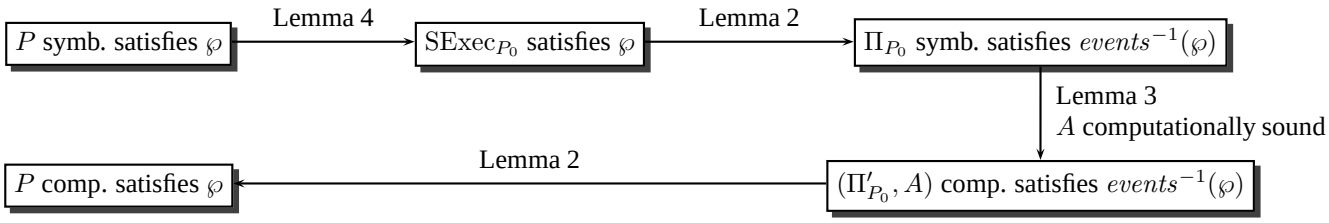

Figure 3. Overview of the proof of Theorem 3.

the current step. Let m be the term sent by C in the current step. Then $S \vdash m$.

SExec_{P_0} satisfies a π-trace property $\wp$ if in interaction with any Dolev-Yao adversary, the sequence of events output by SExec_{P_0} is contained in $\wp$.

Before we finally state and prove the soundness of the applied π-calculus, we provide three lemmas that are used to relate P_0, SExec_{P_0}, Π_{P_0}, and (Π'_{P_0}, A), and to assert the efficiency of the protocol Π'_{P_0}. Figure 3 illustrates the use of these lemmas in the overall proof.

Lemma 2 *Let $\wp$ be a trace property. Then SExec_{P_0} satisfies $\wp$ iff Π_{P_0} symbolically satisfies $events^{-1}(\wp)$ (in the sense of Definition 10). Moreover, P_0 computationally satisfies $\wp$ iff (Π'_{P_0}, A) computationally satisfies $events^{-1}(\wp)$ (in the sense of Definition 10).*

Lemma 3 *The protocol Π'_{P_0} is efficient.*

Lemma 4 *If a closed process P_0 symbolically satisfies a π-trace property $\wp$, then SExec_{P_0} satisfies $\wp$.*

The proof of the lemmas is postponed to Appendix C. With these lemmas at hand, we are finally ready to state and prove the computational soundness of the applied π-calculus as the main result of this section.

Theorem 3 (Comp. soundness in the applied π-calculus) *Assume that the computational implementation of the applied π-calculus (Definition 25) is a computationally sound implementation (in the sense of Definition 11) of the symbolic model of the applied π-calculus (Definition 24).*

If a process P_0 symbolically satisfies a π-trace property $\wp$, then P_0 computationally satisfies $\wp$.

Proof. By Lemma 4, SExec_{P_0} satisfies $\wp$. By Lemma 2, Π_{P_0} symbolically satisfies $events^{-1}(\wp)$. Furthermore, since $\wp$ is an efficiently decidable set, so is $events^{-1}(\wp)$. Using Lemma 3, we have that Π'_{P_0} is an efficient protocol. By assumption, the computational implementation of the applied π-calculus is computationally sound; hence Π'_{P_0} computationally satisfies $events^{-1}(\wp)$. Using Lemma 2, we obtain that P_0 computationally satisfies $\wp$. □

Soundness of encryption in the applied π-calculus Combining the results from this section with those from Section 4 immediately entails a computational soundness result for the applied π-calculus for public-key encryption. The protocol conditions from Section 4 translate into syntactic conditions for the process. As an illustrating example, we used ProVerif to analyze the entity authentication property of the Needham-Schroeder-Lowe protocol. Using the aforementioned results, this yields an implementation of this protocol within the applied π-calculus that is provably secure under active attacks. Further details are provided in Appendix E.

6 Conclusion and future work

We have provided a general framework for conducting computational soundness proofs of symbolic models that abstracts away from many details that are not core for proving computational soundness such as message scheduling, corruption models, and even the internal structure of a protocol. Computational soundness in this framework is shown to be entailed by a novel simulation-based criterion, which allows for proving soundness results in a conceptually modular and generic way. We finally have shown how to use our framework to establish the first computational soundness result for the full-fledged applied π-calculus under active attacks.

The framework currently only considers computational soundness as the preservation of trace properties. Existing definitions of the preservation of more sophisticated properties such as static or observational equivalence [23, 16] can be easily cast in our framework. However, deriving a corresponding simulation-based criterion that entails such stronger soundness results requires conceptual future work. Moreover, we plan to derive the computational soundness of additional calculi, especially those ones that strive for analyzing security protocols in more realistic settings. Calculi for reasoning about implementations of security protocols such as RCF [11] are hence particularly promising targets for this future work.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL)*, pages 104–115, 2001.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 36–47, 1997.

[3] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[4] Pedro Adão and Cédric Fournet. Cryptographically sound implementations for communicating processes. In *Proc. 32nd International Confererence on Automata, Languages and Programming (ICALP)*, pages 83–94, 2006.

[5] Michael Backes and Birgit Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218, 2004.

[6] Michael Backes and Birgit Pfitzmann. Relating symbolic and cryptographic secrecy. In *Proc. 26th IEEE Symposium on Security & Privacy*, pages 171–182, 2005. Extended version in IACR Cryptology ePrint Archive 2004/300.

[7] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003, `http://eprint.iacr.org/`.

[8] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

[9] David Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 2004.

[10] M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 652–663. Springer, 2005.

[11] Jesper Bengtson, Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. Refinement types for secure implementations. In *Proc. 21st IEEE Security Foundations Symposium (CSF)*, pages 17–32, 2008.

[12] Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–96, 2001.

[13] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 86–100, 2004.

[14] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75:3–51, 2008. Online available at `http://www.di.ens.fr/~blanchet/publications/BlanchetAbadiFournetJLAP07.pdf`.

[15] Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proc. 3rd Theory of Cryptography Conference (TCC)*, volume 3876 of *Lecture Notes in Computer Science*, pages 380–403. Springer, 2006.

[16] Hubert Comon-Lundh and Véronique Cortier. Computational soundness of observational equivalence. In *Proc. ACM Conference on Computer and Communications Security*, pages 109–118, 2008.

[17] Véronique Cortier, Steve Kremer, Ralf Küsters, and Bogdan Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. In *Proc. 26th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 176–187, 2006.

[18] Véronique Cortier and Bogdan Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. 14th European Symposium on Programming (ESOP)*, pages 157–171, 2005.

[19] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[20] Shimon Even and Oded Goldreich. On the security of multi-party ping-pong protocols. In *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 34–39, 1983.

[21] Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *Proc. 14th European Symposium on Programming (ESOP)*, pages 172–185, 2005.

[22] Richard Kemmerer, Catherine Meadows, and Jon Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[23] Steve Kremer and Laurent Mazaré. Adaptive soundness of static equivalence. In *Proc. 12th European Symposium On Research In Computer Security (ESORICS)*, pages 610–625, 2007.

[24] Peeter Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.

[25] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[26] Michael Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, 1983.

[27] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.

[28] Lawrence Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.

[29] Steve Schneider. Security properties and CSP. In *Proc. 17th IEEE Symposium on Security & Privacy*, pages 174–187, 1996.

A Semantics of the applied π-calculus (with events)

The semantics is formally defined in Figure 4.

B Postponed definitions

In this section, we provide the postponed definitions.

B.1 Symbolic execution of a π-process

For relating the symbolic and the computational semantics of a π-process, we introduce an additional symbolic execution for closed π-processes. To formulate these semantics, we define a variant of the notion of step and input contexts for the case that η and μ map to fw-terms instead of bitstrings.

Let pub be a set of fw-terms and c a fw-term: An evaluation context E is a *step context for* (P, η, μ, pub) if one of the following structural condictions holds true (we mark the differences to Definition 22 in boldface to increase readability):

- $P = E[\nu a.P_1]$ or
- $P = E[\overline{M_1}\langle N\rangle.P_1][M_2(x).P_2]$ with $\mathbf{eval^{fw}}\, \boldsymbol{M_1\eta\mu} \cong \mathbf{eval^{fw}}\, \boldsymbol{M_2\eta\mu}$ or
- $P = E[let\ x = D\ in\ P_1\ else\ P_2]$ or
- $P = E[event(e).P_1]$ or
- $P = E[!P_1]$ or
- $P = E[\overline{M}\langle N\rangle.P_1]$ with $\mathbf{eval^{fw}}\, \boldsymbol{M\eta\mu} \cong \boldsymbol{M'}$ **for some** $\boldsymbol{M' \in pub}$.

An *input context for* (P, η, μ, c) is an evaluation context E such that $P = E[M(x).P_1]$ and $\mathbf{eval^{fw}}\, \boldsymbol{M\eta\mu} \cong c$. For π-terms M, we hence have that $\mathrm{eval}^{\mathrm{fw}}\, M\eta\mu = M\eta\mu$ (this does not hold for destructor terms D). In these cases, we write the redundant $\mathrm{eval}^{\mathrm{fw}}$ anyway to emphasis the analogy to Definition 22.

Definition 27 (Symbolic execution of a π-process) *Let P_0 be a closed process, and let C be an interactive machine called the adversary. We define the computational π-execution as an interactive machine $\mathrm{Exec}_{P_0}(1^k)$ that takes a security parameter k as argument and interacts with C:*

- Start*: Let $P := P_0$ (where we rename all bound variables and names such that they are pairwise distinct and distinct from all unbound ones). Let η be a totally undefined partial function mapping variables to* ***terms****, let μ be a totally undefined partial function mapping names to* ***terms****, and let pub be an empty set of bitstrings. Let $a_1, \ldots, a_n$ denote the free names in P_0. For each i,* ***choose a different $r_i \in \mathbf{N_P}$*** *Set $\mu := \mu(a_1 := r_1, \ldots, a_n := r_n)$. Send $(public, r_1, \ldots, r_n)$ to C.*[6]
- Transition*: Proceed depending on the type of message received from C as follows:*
  - *When receiving $(listen, c)$ from C* ***where c is a fw-term****, set $pub := pub \cup \{c\}$ and send (ok) to C.*
  - *When receiving $(input, c, m)$ from C* ***where c, m are terms, non-deterministically*** *choose an input context E for (P, η, μ, c) uniformly at random. If no such input context exists, send $(stuck)$ to C. If such an input context exists (where P then is of the form $E[M(x).P_1]$), set $\eta := \eta(x := m)$, $P := E[P_1]$, and send (ok) to C.*
  - *When receiving $(step)$ from C, choose a step context E for (P, η, μ, pub) uniformly at random. If no step context exists, send $(stuck)$ to C. Otherwise, proceed as follows depending on the structure of P:*
    - *$P = E[\nu a.P_1]$:* ***Choose $r \in \mathbf{N_P} \setminus \mathbf{range}\,\mu$****, set $P := E[P_1]$ and $\mu := \mu(a := r)$. Send (ok) to C.*
    - *$P = E[\overline{M_1}\langle N\rangle.P_1][M_2(x).P_2]$: Set $P := E[P_1][P_2]$ and $\eta := \eta(x := \mathbf{eval^{fw}}\, \boldsymbol{N\eta\mu})$. Send (ok) to C.*

[6] In the π-calculus, free names occurring in the initial process represent nonces that are honestly chosen but known to the attacker.

$$\frac{}{P \mid 0 \equiv P} \qquad \frac{}{P \equiv P} \qquad \frac{}{P \mid Q \equiv Q \mid P} \qquad \frac{}{(P \mid Q) \mid R \equiv P \mid (Q \mid R)} \qquad \frac{P \equiv Q \qquad Q \equiv R}{P \equiv R}$$

$$\frac{}{\nu a.\nu b.P \equiv \nu b.\nu a.P} \qquad \frac{P \equiv Q}{P \mid R \equiv Q \mid R} \qquad \frac{a \notin fn(P)}{\nu a.(P \mid Q) \equiv P \mid \nu a.Q} \qquad \frac{P \equiv Q}{\nu a.P \equiv \nu a.Q}$$

$$\frac{N \approx N'}{\overline{N}\langle M\rangle.Q \mid N'(x).P \to Q \mid P\{M/x\}} \qquad \frac{\mathrm{eval}^{\pi}\ D \neq \bot}{let\ x = D\ in\ P\ else\ Q \to P\{\mathrm{eval}^{\pi}\ D/x\}}$$

$$\frac{\mathrm{eval}^{\pi}\ D = \bot}{let\ x = D\ in\ P\ else\ Q \to Q} \qquad \frac{}{!P \to P \mid !P} \qquad \frac{P \to Q}{P \mid R \to Q \mid R \qquad \nu a.P \to \nu a.Q}$$

$$\frac{P' \equiv P \qquad P \to Q \qquad Q \equiv Q'}{P' \to Q'} \qquad \frac{}{event(e).P \xrightarrow{e} P} \qquad \frac{P \xrightarrow{e} Q}{P \mid R \xrightarrow{e} Q \mid R \qquad \nu a.P \xrightarrow{e} \nu a.Q}$$

$$\frac{P' \equiv P \qquad P \xrightarrow{e} Q \qquad Q \equiv Q'}{P' \xrightarrow{e} Q'}$$

Figure 4. Semantics of the applied π-calculus with events.

- $P = E[let\ x = D\ in\ P_1\ else\ P_2]$*: If* $m := \mathbf{eval^{fw}}\ \boldsymbol{D\eta\mu} \neq \bot$*, set* $\eta := \eta(x := m)$ *and* $P := E[P_1]$*; Otherwise set* $P := E[P_2]$*. Send* (ok) *to* C*.*
- $P = E[event(e).P_1]$*: Let* $P := E[P_1]$ *and send* $(event, e)$ *to* C*.*
- $P = E[!P_1]$*: Rename all bound variables of* P_1 *such that they are pairwise distinct and distinct from all variables and names in* P *and in the domains of* η *and* μ*, yielding a process* $\tilde{P_1}$*. Set* $P := E[\tilde{P_1} \mid !P_1]$*. Send* (ok) *to* C*.*
- $P = E[\overline{M}\langle N\rangle.P_1]$*: Set* $P := E[P_1]$*. Send* $(output, \mathbf{eval^{fw}}\ \boldsymbol{M\eta\mu}, \mathbf{eval^{fw}}\ \boldsymbol{N\eta\mu})$ *to* C*.*

- *When receiving anything else, send* $(stuck)$ *to* C*.*

The only differences between Definition 22 and Definition 27 are that the latter operates on fw-terms instead of bitstrings, it computes $\mathrm{eval}^{\mathsf{fw}}\ X\eta\mu$ instead of $\mathrm{ceval}_{\eta,\mu}\ X$, it compares fw-terms using $\cong$ instead of checking for equality of bitstrings, it performs a non-deterministic choice instead of choosing step or input contexts uniformly, and it chooses a fresh fw-nonce $r \in \mathbf{N}_P$ instead of choosing a random bitstring r as value for a restricted name.

The interactive machine SExec_{P_0} performs only the following operations on fw-terms: Applying fw-constructors (this includes nonces) and fw-destructors, comparing using $\cong$ (which can be realized by an application of the destructor $equals'$), and sending and receiving terms. Hence this interactive machine can be realized as a symbolic protocol in the sense of Definition 3: The state of the SExec_{P_0} is used as a node identifier. However, fw-terms are not encoded directly into the node identifier; instead, the node in which they were created (or received) is referenced instead.[7] This is due to the fact that a symbolic protocol allows to treat fw-terms only as black boxes. Note that the process P and the π-terms occurring within P will be encoded in the node identifier (encoded as bitstrings). Operations on fw-terms can then be performed by using constructor and destructor nodes, and the input and output of fw-terms is handled using communication nodes.[8] We call the resulting protocol Π_{P_0}.

C Postponed proofs

This section contains the postponed proofs.

C.1 Proof of Lemma 1

We show that tr' fulfills the conditions on full traces of Definition 7.

This is clear for constructor nodes, since the processing of constructor nodes in the hybrid setting of Definition 14 matches the one in the symbolic setting of Definition 7, with the exception that terms with variables can occur in the hybrid setting. However, since we apply the substitution φ con-

[7] For technical reasons, we do not reference the nodes by its identifier, but instead by the its index in the path from the root to the referring node. Otherwise, the size of node identifiers would grow exponentially.

[8] All communication steps in SExec_{P_0} send π-terms that are tagged with bitstrings. These can be encoded by using communication nodes since the latter allow to attach metadata l to inputs and outputs. Inputs to SExec_{P_0} that consist only of a bitstring without a term encoded by ignoring the attached π-term. Inputs to SExec_{P_0} with containing two terms can be implemented by chaining two communication nodes.

sistently to all terms in tr (i.e., $C(\underline{t})\varphi = C(\underline{t\varphi})$ holds), constructor nodes in the substituted trace $tr' = tr\varphi$ fulfill the symbolic requirement of Definition 7.

Destructor nodes are treated similarly, except that there may be several possible successor nodes. We thus have to ensure that the respective successor node in tr' fulfills the requirements in Definition 7. This immediately follows from the validity of φ, which ensures that Sim's answers (which determine the respective successor node) are consistent (in the sense of Definition 7) with tr'.

Finally, communication nodes in tr' consist of a term $t \in T_x(\mathbf{C}, \mathbf{D})$ sent from Π^C to Sim, and a term t' sent back from Sim to Π^C. By the DY property of Sim, we know that $S\varphi \vdash t'\varphi$, where S denotes all terms (including t) sent from Π^C to Sim so far. Hence, the node satisfies the requirement for communication nodes from Definition 7. This completes the proof of the lemma.

C.2 Proof of Lemma 2

The symbolic case is immediate from the construction of Π_{P_0}. For the computational case, note the fact that the computational implementation of P_0 is defined like the symbolic one, except that it uses the implementations of the fw-constructors and fw-destructors instead of the operating on abstractly on terms (and the implementation of $equals$ uses the identity on bitstrings).

C.3 Proof of Lemma 3

By construction, there are efficient algorithms for computing the labels and successors of a node given its node identifier. It is left to show that the length of the node identifier of a node p is polynomial in the length of the path leading to that node. This is equivalent to showing that the state of SExec_{P_0} is of polynomial-length (when not counting the length of the representations of the fw-terms). For the variables η, μ, and pub, this is immediately satisfied because they grow by at most one entry in each activation of SExec_{P_0}. To show that the length of P is polynomially bounded, note the following facts: In each activation of SExec_{P_0}, P either gets smaller, or we have $P = E[!P_1]$ and P grows by the size of P_1. If $P = E[!P_1]$, then $!P_1$ is also a subterm of P_0 (up to renaming of names and variables). Hence in each activation, P grows at most by the size of P_0. Thus the size of P is linear in the number of activations of SExec_{P_0}.

C.4 Proof of Lemma 4

To show this lemma, it is sufficient to show that if SExec_{P_0} outputs events $e_1, \ldots, e_n$, then $\underline{e}$ is an event π-trace of P_0. Hence, for the following we fix an execution of SExec_{P_0} in interaction with a Dolev-Yao adversary E in which SExec_{P_0} outputs the events $e_1, \ldots, e_n$.

For a given activation of SExec_{P_0}, let P, η, μ, pub denote the corresponding variables from the state of SExec_{P_0} at the beginning of that activation. Let E denote the step or input context chosen in that activation. Let $\underline{n}$ be the domain of μ without the names $r_1, \ldots, r_n$ sent in the message $(public, r_1, \ldots, r_n)$ in the very beginning of the execution of SExec_{P_0}. $P', \eta', \mu', pub', \underline{n}'$ are the corresponding values *after* that activation. Let in be the input and out be the output of SExec_{P_0} in that activation. By $\bar{P}_0, \eta_0, \mu_0, pub_0, \underline{n}_0$ we denote the corresponding values before the first activation but after the sending of the message $(public, r_1, \ldots, r_n)$,[9] and by $P_*, \eta_*, \mu_*, pub_*, \underline{n}_*$ the values after the last activation. We call a name or variable *used* if it occurs in the domain of μ_* or η_*, respectively. Note that $\mu_0 = (a_1 \mapsto r_1, \ldots, a_n \mapsto r_1)$ where $\underline{a}$ are the free names in P_0, but $\underline{n}_0 = \varnothing$. Note that P will never contain unused free variables or names.

Let S denote the list of all fw-terms output by SExec_{P_0} up to the current activation. We encode $S = (s_1, \ldots, s_n)$ as a substitution φ mapping $x_i \mapsto s_i$ where x_i are arbitrary unused variables. We denote by S', φ' and S_0, φ_0 and S_*, φ_* the values of S, φ after the current activation, before the first activation (but after sending $(public, r_1, \ldots, r_n)$), and after the last activation, respectively. Note that $S_0 = (r_1, \ldots, r_n)$.

Let γ be an injective partial function that maps every $N \in \mathbf{N}_E$ to an unused name, and every $N \in \operatorname{range}\mu_*$ to $\mu_*^{-1}(N)$. (Note that γ is well-defined because $\operatorname{range}\mu_* \subseteq \mathbf{N}_P$ and μ_* is injective.) We additionally require that all unused names are in $\operatorname{range}\gamma$. (This is possible since both $\mathbf{N}_E$ and the set of unused names are countably infinite.)

Note that for any π-destructor d and any π-terms $\underline{M}$ with $fv(\underline{M}) \subseteq \operatorname{dom}\eta$ and $fn(M) \subseteq \operatorname{dom}\mu$, we have $d'(\underline{M}\eta\mu)\gamma = d(\underline{M}\eta\mu\gamma)$ (where d' is as in Definition 24). Hence for a destructor term D with $fv(D) \subseteq \operatorname{dom}\eta$ and $fn(D) \subseteq \operatorname{dom}\mu$, we have $\mathrm{eval}^{\mathsf{fw}}(D\eta\mu)\gamma = \mathrm{eval}^{\pi}(D\eta\mu\gamma)$. Since $a\mu\gamma = a$ for all names $a \in \operatorname{dom}\mu$, $D\eta\mu\gamma = D\eta\gamma$. Since $\mathrm{eval}^{\mathsf{fw}}(D\eta\mu)$ does not contain variables, $\mathrm{eval}^{\mathsf{fw}}(D\eta\mu) = \mathrm{eval}^{\mathsf{fw}}(D\eta\mu)\eta$. Thus

$$\mathrm{eval}^{\mathsf{fw}}(D\eta\mu)\eta\gamma = \mathrm{eval}^{\pi}(D\eta\gamma) \tag{2}$$

where the left hand side is defined iff the right hand side is.

Similarly to (2), if $fv(D) \subseteq \operatorname{dom}\varphi$ and $fn(D) \subseteq \operatorname{dom}\gamma^{-1}$, we have $\mathrm{eval}^{\mathsf{fw}}(D\varphi\gamma^{-1})\gamma = \mathrm{eval}^{\pi}(D\varphi\gamma)$. For a fw-term t with $S \vdash t$, from the definition of $\vdash$ is follows that $t = \mathrm{eval}^{\mathsf{fw}}\, D_t\varphi\gamma^{-1}$ for some destructor π-term D_t containing only unused names and variables in $\operatorname{dom}\varphi$ (note that every $N \in \mathbf{N}_E$ can be expressed as $a\gamma^{-1}$ for some unused a). Since all unused names are in $\operatorname{dom}\gamma^{-1}$, we have

$$t\gamma = \mathrm{eval}^{\mathsf{fw}}(D_t\varphi\gamma^{-1})\gamma = \mathrm{eval}^{\pi}(D_t\varphi\gamma). \tag{3}$$

Given two fw-terms $t \cong u$ such that t, u only contains nonces $N \in \mathbf{N}_E \cup \operatorname{range}\mu_*$, we have that $equals'(t, u) \neq \bot$. By definition of $equals'$ (Definition 24) and using that γ is injective and defined on $\mathbf{N}_E \cup \operatorname{range}\mu_*$, we have $equals'(t, u) = equals(t\gamma, u\gamma)\gamma^{-1}$ and hence

[9] We use the variable name $\bar{P}_0$ because P_0 is already used for the input of SExec_{P_0}. Note however that $\bar{P}_0 \equiv P_0$.

$equals(t\gamma, u\gamma) \neq \bot$. Hence, for t, u only containing nonces $N \in \mathbf{N}_E \cup \operatorname{range}\mu_*$, we have

$$t \cong u \qquad \Longrightarrow \qquad t\gamma \approx u\gamma \tag{4}$$

We call a process Q valid for φ if it does not contain events, all its free names are unused names, and all its free variables are in the domain of φ.

Claim: For all Q' valid for φ', there is a Q valid for φ such that $\nu\underline{n}.(Q\varphi\gamma|P\eta\gamma) \rightsquigarrow \nu\underline{n}'.(Q'\varphi'\gamma|P'\eta'\gamma)$. Here $\rightsquigarrow$ denotes $\xrightarrow{e}$ if $out = (event, e)$, and $\rightarrow^*$ otherwise.

Assuming that we have shown this claim, it follows that for all Q_* valid for φ_*, there is a Q_0 valid for φ_0 such that $\nu\underline{n}_0.(Q_0\varphi_0\gamma|P_0\eta_0\gamma) \rightarrow^* \xrightarrow{e_1} \rightarrow^* \xrightarrow{e_2} \rightarrow^* \ \ldots \ \rightarrow^* \xrightarrow{e_n} \rightarrow^* \nu\underline{n}.(Q_*\varphi_*\gamma|P_*\eta_*\gamma)$. Since $\eta_0 = \varnothing$ and since $\bar{P}_0$ does not contain $N \in \mathbf{N}$ (being a π-term) and since $\bar{P}_0$ is a renaming of P_0, we have $\bar{P}_0\eta_0\gamma = \bar{P}_0\gamma = \bar{P}_0 \equiv P_0$. Then, with $\tilde{Q} := Q_0\varphi_0\gamma$ and using $\underline{n}_0 = \varnothing$ we have $\tilde{Q}|P_0 \equiv \nu\underline{n}_0.(Q_0\varphi_0\gamma|\bar{P}_0\eta_0\gamma) \rightarrow^* \xrightarrow{e_1} \rightarrow^* \xrightarrow{e_2} \rightarrow^* \ \ldots \ \rightarrow^* \xrightarrow{e_n} \rightarrow^*$. Since $\tilde{Q}$ does not contain events, this implies that $\underline{e}$ is an event π-trace of P_0. This shows the lemma.

It is left to prove the claim. We distinguish the following cases:

- $out = stuck$ or $in = (listen, c)$: Then $P = P'$, $\varphi = \varphi'$, $\eta = \eta'$, and $\underline{n} = \underline{n}'$. Hence with $Q := Q'$, we have $\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma) = \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma)$.

- $(in, out) = (step, ok)$ and $P = E[\nu a.P_1]$: Then $P' = E[P_1]$, $\varphi' = \varphi$, $\eta = \eta'$, and $\underline{n}' = \underline{n}\|a$ for some $r \in \mathbf{N}_P \setminus \operatorname{range}\mu$, and $\mu' = \mu(a := r)$. Since $a \in \operatorname{dom}\mu' \subseteq \dim\mu_*$, a is used. Since Q' is valid for $\varphi = \varphi'$, this implies $a \notin fn(Q')$. Set $Q := Q'$. Then Q is valid for φ. Since we maintain the invariant that all bound names in P are pairwise distinct and distinct from all other names in P or $\operatorname{dom}\mu$, we have $r \notin fn(E)$ and $r \notin \underline{n}$. Furthermore, by the same invariant, we have $r \notin \operatorname{dom}\mu$. Note that the execution of SExec_{P_0} maintains the following invariant: Any nonce $N \in \mathbf{N}_P$ occurring (as a subterm) in the range of η or φ is also in the range of μ. (This uses the fact that the Dolev-Yao adversary cannot derive protocol nonces that have never been sent.) Hence $r \in \mathbf{N}_P \setminus \operatorname{range}\mu$ does not occur in the range of η or φ. By definition of γ and since a is used, this implies that for $N \neq r$, $\gamma(N) \neq a$. Thus for all variables x, $x\eta\gamma$ and $x\varphi\gamma$ do not contain a, and hence $a \notin fn(Q\varphi\gamma) \cup fn(E\eta\gamma)$. Together with $r \notin \underline{n}$ we get $\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma) = \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[\nu a.P_1\eta\gamma]) \equiv \nu\underline{n}.\nu a.(Q\varphi\gamma \mid (E\eta\gamma)[P_1\eta\gamma]) = \nu\underline{n}'.(Q'\varphi'\gamma \mid (E\eta'\gamma)[P_1\eta'\gamma]) = \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma)$.

- $(in, out) = (step, ok)$ and $P = E[\overline{M_1}\langle N\rangle.P_1][M_2(x).P_2]$ with $\mathrm{eval}^{\mathsf{fw}} M_1\eta\mu \cong \mathrm{eval}^{\mathsf{fw}} M_2\eta\mu$: Then $P' = E[P_1][P_2]$ and $\underline{n}' = \underline{n}$ and $\varphi' = \varphi$ and $\eta' = \eta(x := t)$ with $t := \mathrm{eval}^{\mathsf{fw}} N\eta\mu$. Since we maintain the invariant that all bound variables in P are distinct from all other names in P or $\operatorname{dom}\eta$, we have $x \notin fv(E) \cup fv(P_1)$ and $x \notin \operatorname{dom}\eta$. Hence $E\eta\gamma = E\eta'\gamma$ and $P_1\eta\gamma = P_1\eta'\gamma$. Furthermore, we have $P_2\eta\gamma\{N\eta\gamma/x\} = P_2\eta\gamma\{N\eta\mu\gamma/x\} = P_2\eta\{N\eta\mu/x\}\gamma = P_2\eta\{t/x\}\gamma = P_2\eta'\gamma$. Since a Dolev-Yao adversary will never derive protocol nonces that have never been sent, we have that only nonces $N \in \mathbf{N}_E \cup \operatorname{range}\mu_*$ occur in $M_1\eta\mu$ and $M_2\eta\mu$. With $M_1\eta\mu = \mathrm{eval}^{\mathsf{fw}} M_1\eta\mu \cong \mathrm{eval}^{\mathsf{fw}} M_2\eta\mu = M_2\eta\mu$ and Equation 4 we get $M_1\eta\gamma = M_1\eta\mu\gamma \approx M_2\eta\mu\gamma = M_2\eta\gamma$.

  Hence with $Q := Q'$, we have

  $$\begin{aligned}
  &\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma)\\
  &= \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[\overline{M_1\eta\gamma}\langle N\eta\gamma\rangle.P_1\eta\gamma][M_2\eta\gamma(x).P_2\eta\gamma])\\
  &\rightarrow \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[P_1\eta\gamma][P_2\eta\gamma\{N\eta\gamma/x\}]\\
  &= \nu\underline{n}'.(Q\varphi'\gamma \mid (E\eta'\gamma)[P_1\eta'\gamma][P_2\eta'\gamma])\\
  &= \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma).
  \end{aligned}$$

  Since Q' is valid for $\varphi' = \varphi$, $Q = Q'$ is valid for φ.

- $(in, out) = (step, ok)$ and $P = E[let\ x = D\ in\ P_1\ else\ P_2]$ and $\mathrm{eval}^{\mathsf{fw}} D\eta\mu = \bot$: Then $P' = E[P_2]$ and $\varphi' = \varphi$, and $\underline{n}' = \underline{n}$ and $\eta' = \eta$. Set $Q := Q'$. Then Q is valid for $\varphi = \varphi'$. By (2), $\mathrm{eval}^{\pi}(D\eta\gamma) = \bot$. Hence

  $$\begin{aligned}
  &\nu\underline{n}.(Q\varphi\gamma|P\eta\gamma)\\
  &= \nu\underline{n}.(Q\varphi\gamma|(E\eta\gamma)[let\ x = D\eta\gamma\ in\ \ldots\ else\ P_2\eta\gamma])\\
  &\rightarrow \nu\underline{n}.(Q\varphi\gamma|(E\eta\gamma)[P_2\eta\gamma])\\
  &= \nu\underline{n}'.(Q'\varphi'\gamma|P'\eta'\gamma).
  \end{aligned}$$

- $(in, out) = (step, ok)$ and $P = E[let\ x = D\ in\ P_1\ else\ P_2]$ and $\mathrm{eval}^{\mathsf{fw}} D\eta\mu \neq \bot$: Then $P' = E[P_1]$ and $\varphi' = \varphi$ and $\underline{n}' = \underline{n}$ and $\eta' = \eta(x := \mathrm{eval}^{\mathsf{fw}} D\eta\mu)$. Set $Q := Q'$. Then Q is valid for $\varphi = \varphi'$. By (2), $t := \mathrm{eval}^{\pi}(D\eta\gamma) = \mathrm{eval}^{\mathsf{fw}}(D\eta\mu)\eta\gamma \neq \bot$. Since we maintain the invariant that all bound variables in P are distinct from all other names in P or $\operatorname{dom}\eta$, we have $x \notin fv(E)$ and $x \notin \operatorname{dom}\eta$. Hence $P\eta\gamma = (E\eta\gamma)[let\ x = D\eta\gamma\ in\ P_1\eta\gamma\ else\ \ldots] \rightarrow (E\eta\gamma)[P_1\eta\gamma\{t/x\}])$. Furthermore, $P_1\eta\gamma\{t/x\} = P_1\eta\gamma\{\mathrm{eval}^{\mathsf{fw}}(D\eta\mu)\eta\gamma/x\} = P_1\{\mathrm{eval}^{\mathsf{fw}}(D\eta\mu)/x\}\eta\gamma = P_1\eta'\gamma$. Since $x \notin fv(E) \cup \operatorname{dom}\eta$, $(E\eta\gamma)[P_1\eta'\gamma] = E[P_1]\eta'\gamma = P'\eta'\gamma$. Hence $P\eta\gamma \rightarrow P'\eta'\gamma$. Since $Q = Q'$, $\varphi = \varphi'$, and $\underline{n} = \underline{n}'$, it follows that $\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma) \rightarrow \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma)$.

- $(in, out) = (step, ok)$ and $P = E[!P_1]$: Then $P' = E[!P_1 \mid \tilde{P}_1]$ for some $\tilde{P}_1 \equiv P_1$ and $\varphi' = \varphi$ and $\underline{n}' = \underline{n}$ and $\eta' = \eta$. Set $Q := Q'$. Then Q is valid for $\varphi = \varphi'$.

Hence

$$\begin{aligned}&\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma)\\&= \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[!P_1\eta\gamma])\\&\to \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[P_1\eta\gamma \mid !P_1\eta\gamma]\\&\equiv \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[\tilde{P}_1\eta\gamma \mid !P_1\eta\gamma]\\&= \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma).\end{aligned}$$

- $(in, out) = (step, (output, t_M, t_N))$ and $P = E[\overline{M}\langle N\rangle.P_1]$ with $t_M := \mathrm{eval}^{\mathsf{fw}} M\eta\mu$ and $t_N := \mathrm{eval}^{\mathsf{fw}} N\eta\mu$ and $t_M \cong M'$ for some $M' \in pub$: Then $P' = E[P_1]$ and $S' = S\|t_M\|t_N$ and $\varphi' = \varphi(x_{n+1} := t_M, x_{n+2} := t_N)$ where $x_{n+1}, x_{n+2} \notin \mathrm{dom}\,\varphi$ are unused and $\underline{n}' = \underline{n}$ and $\eta' = \eta$. Since $M' \in pub$, the adversary must have sent M', hence $S \vdash M$. By Equation 3, there is a destructor π-term $D_{M'}$ containing only unused names and variables in $\mathrm{dom}\,\varphi$ such that $M'\gamma = \mathrm{eval}^{\pi}(D_{M'}\varphi\gamma)$.

  Since a Dolev-Yao adversary will never derive protocol nonces that have never been sent, we have that only nonces $N \in \mathbf{N}_E \cup \mathrm{range}\,\mu$ occur in M' and $M\eta\mu$. Hence with (4), from $t_M \cong M'$ it follows that $M\eta\gamma = M\eta\mu\gamma \approx M'\gamma$.

  Let $Q := let\ x_{n+1} = D_{M'}\ in\ x_{n+1}(x_{n+2}).Q'\ else\ 0$. Then $Q\varphi\gamma \to M'\gamma(x_{n+2}).Q'\varphi\gamma\{M'\gamma/x_{n+1}\}$. Then

$$\begin{aligned}&Q\varphi\gamma \mid P\eta\gamma = Q\varphi\gamma \mid (E\eta\gamma)[\overline{M\eta\gamma}\langle N\eta\gamma\rangle.P_1\eta\gamma]\\&\to M'\gamma(x_{n+2}).Q'\varphi\gamma\{M'\gamma/x_{n+1}\}\\&\qquad\mid (E\eta\gamma)[\overline{M\eta\gamma}\langle N\eta\gamma\rangle.P_1\eta\gamma]\\&\to Q'\varphi\gamma\{M\eta\gamma/x_{n+1}\}\{N\eta\gamma/x_{n+2}\} \mid (E\eta\gamma)[P_1\eta\gamma]\\&= Q'\varphi\{M\eta\mu/x_{n+1}\}\{N\eta\mu/x_{n+2}\}\gamma|P'\eta\gamma\end{aligned}$$

  Since $x_{n+1}, x_{n+2} \notin \mathrm{dom}\,\varphi$, $Q'\varphi\{M\eta\mu/x_{n+1}\}\{N\eta\mu/x_{n+2}\}\gamma = Q'\varphi'\gamma$. Hence $Q\varphi \mid P\eta\gamma \to^* Q'\varphi'\gamma \mid P\eta\gamma = Q'\varphi'\gamma \mid P\eta'\gamma$. Since $\underline{n} = \underline{n}'$ we have $\nu\underline{n}.(Q\varphi \mid P\eta\gamma) \to^* \nu\underline{n}'.(Q'\varphi'\gamma \mid P\eta'\gamma)$.

  Since Q' is valid, Q does not contain events, and its free names are unused names, and $fv(Q) \subseteq \mathrm{dom}\,\varphi' = \mathrm{dom}\,\varphi \cup \{x_{n+1}, x_{n+2}\}$. Since x_{n+1} and x_{n+2} are bound on top level in Q, $x_{n+1}, x_{x+2} \notin fv(Q)$, $fv(Q) \subseteq \mathrm{dom}\,\varphi$. Hence Q is valid.

- $(in, out) = (step, (event, e))$ and $P = E[event(e).P_1]$: Then $P' = E[P_1]$ and $\varphi' = \varphi$ and $\underline{n}' = \underline{n}$ and $\eta' = \eta$. We have $\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma) = \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[event(e).P_1\eta\gamma]) \xrightarrow{e} \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[P_1\eta\gamma]) = \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma)$.

- $(in, out) = ((input, c, m), ok)$ and $P = E[M(x).P_1]$ and $\mathrm{eval}^{\mathsf{fw}} M\eta\mu \cong c$: Then $P' = E[P_1]$ and $\varphi' = \varphi$ and $\underline{n}' = \underline{n}$ and $\eta' = \eta(x := m)$. Furthermore, since SExec_{P_0} interacts with a Dolev-Yao adversary, $S \vdash c, m$. By Equation 3, there are destructor π-terms D_c, D_m containing only unused names and variables in $\mathrm{dom}\,\varphi$ such that $c\gamma = \mathrm{eval}^{\pi}(D_c\varphi\gamma)$ and $m\gamma = \mathrm{eval}^{\pi}(D_m\varphi\gamma)$. Since a Dolev-Yao adversary will never derive protocol nonces that have never been sent, we have that only nonces $N \in \mathbf{N}_E \cup \mathrm{range}\,\mu$ occur in c and in $M\eta\mu$. Hence with (4), from $M\eta\mu = \mathrm{eval}^{\mathsf{fw}} M\eta\mu \cong c$ it follows that $M\eta\gamma = M\eta\mu\gamma \approx c\gamma$.

  Pick some $y, z \notin fv(Q') \cup \mathrm{dom}\,\varphi'$. Let $Q := let\ y = D_c\ in\ let\ z = D_m\ in\ \overline{y}\langle z\rangle.Q'\ else\ 0\ else\ 0$. Then $Q\varphi\gamma \to \overline{c\gamma}\langle m\gamma\rangle.Q'\varphi\gamma$. Then

$$\begin{aligned}&Q\varphi\gamma \mid P\eta\gamma\\&= Q\varphi\gamma \mid (E\eta\gamma)[M\eta\gamma(x).P_1\eta\gamma]\\&\to \overline{c\gamma}\langle m\gamma\rangle.Q'\varphi\gamma \mid (E\eta\gamma)[M\eta\gamma(x).P_1\eta\gamma]\\&\to Q'\varphi\gamma \mid (E\eta\gamma)[P_1\eta\gamma\{m\gamma/x\}].\end{aligned}$$

  Since we maintain the invariant that all bound variables in P are distinct from all other names in P or $\mathrm{dom}\,\eta$, we have $x \notin fv(E)$ and $x \notin \mathrm{dom}\,\eta$. Hence $E\eta\gamma = E\eta'\gamma$ and $P_1\eta\gamma = P_1\eta\{m/x\}\gamma = P_1\eta'\gamma$. Furthermore since $\varphi = \varphi'$ we have $Q'\varphi\gamma = Q'\varphi'\gamma$. Thus $Q'\varphi\gamma \mid (E\eta\gamma)[P_1\eta\gamma\{m\gamma/x\}] = Q'\varphi'\gamma \mid (E\eta'\gamma)[P_1\eta'\gamma] = Q'\varphi'\gamma \mid P'\eta'\gamma$. Thus $Q\varphi\gamma \mid P\eta\gamma \to^* Q'\varphi'\gamma \mid P'\eta'\gamma$ and with $\underline{n} = \underline{n}'$ we have $\underline{n}.Q\varphi\gamma \mid P\eta\gamma \to^* \underline{n}'.Q'\varphi'\gamma \mid P'\eta'\gamma$.

  Since Q' is valid and all $fv(D_c, D_m) \subseteq \mathrm{dom}\,\varphi$, we have that Q is valid.

D Encryption (postponed details)

Implementation We require that the implementation A of the symbolic model $\mathbf{M}$ has the following properties:

- There are disjoint and efficiently recognizable sets of bitstrings representing the types nonces, ciphertexts, public keys, secret keys, and pairs. We require that the type nonces consists of all k-bit strings. (Here and in the following, k denotes the security parameters.)
- The functions A_E, A_{pk}, A_{sk}, and A_{pair} length-regular. We call an n-ary function f length regular if $|m_i| = |m_i'|$ for $i = 1, \ldots, n$ implies $|f(\underline{m})| = |f(\underline{m}')|$.
- A_N for $N \in \mathbf{N}$ returns a uniformly random $r \in \{0,1\}^k$.
- For all $m_1, m_2 \in \{0,1\}^*$ we have $A_{fst}(A_{pair}(m_1, m_2)) = m_1$ and $A_{snd}(A_{pair}(m_1, m_2)) = m_2$. Every m of type pair is in the range of A_{pair}.
- $A_{pkof}(A_E(p, x, y)) = p$ for all $p, x \in \{0,1\}^*$, $y \in \{0,1\}^k$. $A_{pkof}(e) \neq \bot$ for any e of type ciphertext and $A_{pkof}(e) = \bot$ for any e that is not of type ciphertext.
- $A_D(A_{sk}(r), m) = \bot$ if $r \in \{0,1\}^*$ and $A_{pkof}(m) \neq A_{pk}(r)$. (This implies that the public key is uniquely determined by the secret key.)
- $A_D(A_{sk}(r), A_E(A_{pk}(r), m, r')) = m$ for all $r, r' \in \{0,1\}^k$.

- $A_{ispk}(x) = x$ for any x of type public key. $A_{ispk}(x) = \bot$ for any x not of type public key.
- $A_{isenc}(x) = x$ for any x of type ciphertext. $A_{isenc}(x) = \bot$ for any x not of type ciphertext.
- We define an encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as follows: KeyGen picks a random $r \leftarrow \{0,1\}^k$ and returns $(A_{pk}(r), A_{sk}(r))$. $\mathsf{Enc}(p, m)$ picks a random $r \leftarrow \{0,1\}^k$ and returns $A_E(p, m, r)$. $\mathsf{Dec}(k, c)$ returns $A_D(k, c)$. We require that then $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is IND-CCA secure.

Protocol conditions. The computational soundness result we derive in this section requires that the symbolic protocol satisfies certain constraints. In a nutshell, these constraints require that encryption and key generation always use fresh nonces, that decryption only uses honestly generated secrets keys, and that the protocol does not produce garbage terms. We call protocols satisfying these conditions *encryption-safe*. In detail, the conditions are the following:

1. The argument of every pk-constructor node and of every sk-constructor node and the third argument of every E-constructor node is an N-constructor node with $N \in \mathbf{N}_P$. (Here and in the following, we call the nodes referenced by a protocol node its arguments.)
2. Every constructor node that is the argument of a pk-constructor node or of an sk-constructor node on some path p occurs only as argument to pk- and sk-constructor nodes on that path p.
3. Every constructor node that is the third argument of an E-constructor node on some path p occurs exactly once as an argument in that path p.
4. Every sk-constructor node occurs only as the first argument of a D-destructor node.
5. The first argument of a D-destructor node is an sk-constructor node.
6. The first argument of an E-constructor node is a pk-constructor node or an $ispk$-destructor node.
7. There are no constructor nodes with the constructors $garbage$, $garbageE$, or $N \in \mathbf{N}_E$.

Construction of the simulator. In the following, we define distinct nonces $N^m \in \mathbf{N}_E$ for each $m \in \{0,1\}^*$. In a hybrid execution, we call a term t *honestly generated* if it occurs as a subterm of a term sent by the protocol Π^C to the simulator before it has occurred as a subterm of a term sent by the simulator to the protocol Π^C.

For an adversary E and a polynomial p, we construct the simulator Sim as follows: In the first activation, it chooses $r_N \in \{0,1\}^k$ for every $N \in \mathbf{N}_P$. It maintains an integer len, initially 0. At any point in the execution, $\mathcal{N}$ denotes the set of all nonces $N \in \mathbf{N}_P$ that occurred in terms received from Π^C. Sim internally simulates the adversary E. When receiving a tuple $(l, \tilde{t}_1, \ldots, \tilde{t}_n)$ from Π^C, it passes $(l, \beta(\tilde{t}_1), \ldots, \beta(\tilde{t}_n))$ to E where the partial function $\beta : T(\mathbf{C}) \to \{0,1\}^*$ is defined below. When E answers with (l', m), the simulator sends $(l', \tau(m))$ to Π^C where the function $\tau : \{0,1\}^* \to T(\mathbf{C})$ is defined below. When the simulator receives $(info, \nu, t)$, the simulator increases len by $\ell(t) + 1$ where $\ell : T(\mathbf{C}) \to \{0,1\}^*$ is defined below. If $len > p(k)$, the simulator answers with $(timeout)$, otherwise with $(proceed)$. If the simulator receives a question, it fails. At the end, Sim outputs the substitution $\varphi = \varnothing$.

Translation functions. The partial function $\beta : T_x(\mathbf{C}) \to \{0,1\}^*$ is defined as follows (where the first matching rule is taken):

- $\beta(N) := r_N$
- $\beta(pk(N)) := A_{pk}(r_N)$ if $N \in \mathcal{N}$.
- $\beta(sk(N)) := A_{sk}(r_N)$ if $N \in \mathcal{N}$.
- $\beta(pk(N^m)) := m$.
- $\beta(pair(t_1, t_2)) := A_{pair}(\beta(t_1), \beta(t_2))$.
- $\beta(E(pk(N), t, M)) := A_E(A_{pk}(r_N), \beta(t), r_M)$ if $N, M \in \mathcal{N}$.
- $\beta(E(pk(t_1), t_1, M)) := A_E(\beta(pk(t_1)), \beta(t_1), r_M)$ if $M \in \mathcal{N}$.
- $\beta(E(pk(M), t, N^m)) := m$ if $M \in \mathcal{N}$.
- $\beta(garbage(N^c)) := c$.
- $\beta(garbageE(t, N^c)) := c$.
- $\beta(t) = \bot$ in all other cases.

The total function $\tau : \{0,1\}^* \to T(\mathbf{C})$ is defined as follows (where the first matching rule is taken):

- $\tau(r) := N$ if $r = r_N$ for some $N \in \mathcal{N}$.
- $\tau(r) := N^r$ if r is of type nonce.
- $\tau(e) := pk(N)$ if e has earlier been output by $\beta(pk(N))$ for some $N \in \mathcal{N}$.
- $\tau(e) := pk(N^e)$ where e is of type public key.
- $\tau(m) := pair(\tau(A_{fst}(m)), \tau(A_{snd}(m)))$ if m of type pair.
- $\tau(c) := E(pk(M), t, N)$ if c has earlier been output by $\beta_f(E(pk(M), t, N))$ for some $N, M \in \mathcal{N}$, $t \in T(\mathbf{C})$.
- $\tau(c) := E(pk(N), \tau(A_D(A_{sk}(r_N), c), N^c)$ if c is of type ciphertext and $\tau(A_{pkof}(c)) = pk(N)$ for some $N \in \mathcal{N}$. If the A_D returns $\bot$, $\tau(c) := garbageE(pk(N), N^c)$.
- $\tau(c) := garbageE(\tau(A_{pkof}(e)), N^c)$ if c is of type ciphertext.
- $\tau(m) := garbage(N^m)$ otherwise.

The function $\ell : T(\mathbf{C}) \to \{0,1\}^*$ is defined as $\ell(t) := |\beta(t)|$. Note that $\ell(t)$ does not depend on the values of r_N because of the length-regularity of A_{pk}, A_{sk}, A_{pair}, and A_E. Hence $\ell(t)$ can be computed without accessing r_N.

The faking simulator. The simulator Sim' is defined exactly like Sim, except that when computing $\beta(pk(N))$ with $N \in \mathcal{N}$, it picks a new public/secret key pair (pk_N, sk_N) using KeyGen (unless (pk_N, sk_N) are already defined) and returns pk_N. Analogously for $\beta(sk(N))$. When computing $\beta(E(pk(N), t, M))$ with $N, M \in \mathcal{N}$, the simulator invokes $\mathsf{Enc}(pk_N, \beta(t))$ instead of computing

$A_E(A_{pk}(r_N), \beta(t), r_M)$, and when computing $\tau(c)$ it invokes $\mathsf{Dec}(sk_N, c)$ instead of computing $A_D(A_{sk}(r_N), c)$.

The simulator Sim_f is defined like Sim', except that instead of invoking $Enc(pk_N, \beta(t))$, it invokes $Enc(pk_N, 0^{\ell(t)})$.

Properties of the simulator. We derive several properties of the simulators Sim and Sim_f that will finally allow to show that Sim is a good simulator for encryption-safe protocols. In the following, let Π' always denote an encryption-safe probabilistic symbolic protocol.

Lemma 5 *The simulators Sim, Sim' and Sim_f run in polynomial time. Sim is consistent and abort-free.*

Proof. By inspection of the construction of the simulators we see that they run in polynomial time. Since Sim never sends terms containing variables, Π^C never sends questions, so Sim is trivially consistent. Since Sim only aborts when receiving a question, it is abort-free. □

Lemma 6 *The full traces $H\text{-}Trace_{\mathbf{M},\Pi,Sim}$ and $H\text{-}Trace_{\mathbf{M},\Pi,Sim_f}$ are computationally indistinguishable.*

Proof. Note that the difference between Sim and Sim' is that the randomness for the key generation and the encryption is chosen by the algorithms KeyGen and Enc in Sim', while Sim uses nonces r_N instead. However, from protocol conditions 1, 2, 3, it follows that Sim never uses a given randomness r_N twice. Hence the full traces $H\text{-}Trace_{\mathbf{M},\Pi,Sim}$ and $H\text{-}Trace_{\mathbf{M},\Pi,Sim'}$ are indistinguishable.

Note that Sim' invokes $\mathsf{Dec}(sk_N, c)$ only for values c that have not been output by $\beta(E(pk(M), t, N))$. Thus $\mathsf{Dec}(sk_N, c)$ is invoked only for values c that have not been output by $\mathsf{Enc}(pk_N, \cdot)$. Since $|\beta(t)| = |0^{\ell(t)}|$ by definition of ℓ, the IND-CCA property of $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ implies that the full traces $H\text{-}Trace_{\mathbf{M},\Pi,Sim'}$ and $H\text{-}Trace_{\mathbf{M},\Pi,Sim_f}$ are indistinguishable. Using the transitivity of computational indistinguishability, the lemma follows. □

Lemma 7 *Sim is indistinguishable for $\mathbf{M}$, Π, A, and for every polynomial p.*

Proof. We will first show that when fixing the randomness of the adversary and the protocol, the node trace $Nodes^p_{\mathbf{M},A,\Pi,E}$ in the computational execution and the node trace $H\text{-}Nodes_{\mathbf{M},\Pi,Sim}$ in the hybrid execution are equal. Hence, fix the variables r_N for all $N \in \mathbf{N}_P$, fix a random tape for the adversary, and for each node ν fix a choice e_ν of an outgoing edge.

Assume that r_N, $A_{pk}(r_N)$, and $A_{sk}(r_N)$ are pairwise distinct for all $N \in \mathcal{N}^*$. (This happens with overwhelming probability when the randomness is uniformly chosen.)

In the following, we designate the values f_i and ν_i in the computational execution by f'_i and ν'_i, and in the hybrid execution by f_i^C and ν_i^C. Let s'_i denote the state of the adversary E in the computational model, and s_i^C the state of the simulated adversary in the hybrid model.

Claim 1: In the hybrid execution, for any $b \in \{0,1\}^*$, $\beta(\tau(b)) = b$.

This claim follows by induction over the length of b and by distinguishing the cases in the definition of τ.

Claim 2: In the hybrid execution, at any constructor or destructor node $\nu = \nu_i$ with constructor or destructor F and arguments $\bar{\nu}_1, \ldots, \bar{\nu}_n$ the following holds: Let t_i be the term stored at node $\bar{\nu}_i$ (i.e., $t_j = f'_i(\bar{\nu}_j)$). Then $\beta(F(\underline{t})) = A_F(\beta(t_1), \ldots, \beta(t_n))$. Here the left hand side is defined iff the right hand side is.

We show Claim 2. For nonces $F = N \in \mathbf{N}_P$, the claim holds because $\beta(N) = r_N = A_N$. For $F = pk$, note that by protocol condition 1, we have $t_1 \in \mathbf{N}_P$. Then $\beta(pk(t_1)) = A_{pk}(r_{t_1}) = A_{pk}(\beta(t_1))$. Analogously for $F = sk$. For $F = pair$, $F = fst$, $F = snd$ this follows directly from the definition of β. For $F = ispk$, if $t_1 = pk(t'_1)$, we have that $t'_1 = pk(N)$ with $N \in \mathcal{N}$ or $t'_1 = pk(N^m)$ where m is of type ciphertext (as other subterms of the form $pk(\cdot)$ are neither produced by the protocol nor by τ). In both cases, $\beta(t_1)$ is of type public key. Hence $\beta(ispk(t_1)) = \beta(pk(t'_1)) = A_{ispk}(\beta(pk(t'_1))) = A_{ispk}(\beta(t_1))$. If t_1 is not of the form $pk(\cdot)$, then $\beta(t_1)$ is not of type public key (this uses that τ only uses N^m with m of type public key inside a term $pk(N^m)$). Hence $\beta(ispk(t_1)) = \bot = A_{ispk}(\beta(t_1))$. Similarly for $isenc$, $pkof$, E (the latter using protocol condition 1), and D (using protocol condition 5). By construction of τ, for all terms $t_1 \neq t_2$ that may occur in the hybrid execution, $\beta(t_1) \neq \beta(t_2)$ (using the fact that all r_N, $pk(r_N)$, and $sk(r_N)$ are pairwise distinct). Hence $\beta(A_{equals}(t_1, t_2)) = \beta(t_1)$ iff $t_1 = t_2$ and $A_{equals}(\beta(t_1), \beta(t_2)) = \beta(t_1)$ iff $\beta(t_1) = \beta(t_2)$ iff $t_1 = t_2$. By protocol condition 7, the constructors $garbage$, $garbageE$, and $N \in \mathbf{N}_E$ do not occur in the protocol. This shows Claim 2.

We will now show that for the random choices fixed above, $Nodes^p_{\mathbf{M},A,\Pi,E} = H\text{-}Nodes_{\mathbf{M},\Pi,Sim}$.

To prove this, we show the following invariant: $f'_i = \beta \circ f_i^C$ and $\nu'_i = \nu_i^C$ and $s_i = s'_i$ for all i.

We have $f'_0 = f_0^C = \varnothing$ and $\nu'_0 = \nu_0^C$ is the root node, so the invariant is satisfied for $i = 0$. Assume that the invariant holds for some i. If ν'_i is a non-deterministic node, $\nu'_{i+1} = \nu_{i+1}^C$ is determined by $e_{\nu'_i} = e_{\nu_i^C}$. Since a non-deterministic node does not modify f and the adversary is not activated, $\tau \circ f'_{i+1} = f_{i+1}^C$ and $s_i = s'_i$. Hence the invariant holds for $i+1$ if ν'_i is a non-deterministic node.

If ν'_i is a constructor node with constructor C, we have that $f'_{i+1}(\nu'_i) = A_C(f'_i(\bar{\nu}_1), \ldots, f'_i(\bar{\nu}_n)) = A_C(\beta(f_i^C(\bar{\nu}_1)), \ldots, \beta(f_i^C(\bar{\nu}_n)))$ for some nodes $\bar{\nu}_s$ depending on the label of ν'_i. And $f_{i+1}^C(\nu'_i) = f_{i+1}^C(\nu_i^C) = C(f_i^C(\bar{\nu}_1), \ldots, f_i^C(\bar{\nu}_n))$. From Claim 2 it follows that $\beta(f_{i+1}^C(\nu'_i)) = f'_{i+1}(\nu'_i)$ and hence $\beta \circ f_{i+1}^C = f'_{i+1}$. The successor node of a constructor node is unique, hence $\nu'_{i+1} = \nu_{i+1}^C$, and the adversary E is not invoked, hence $s'_{i+1} = s_{i+1}^C$. Hence the invariant holds for $i+1$ if ν'_i is a constructor node.

If ν'_i is a constructor node with destructor D, $\beta \circ f^C_{i+1} = f'_{i+1}$ and $s'_{i+1} = s^C_{i+1}$ are shown like in the case of constructor nodes. Furthermore, from Claim 2 it also follows that $A_C(f'_i(\bar{\nu}_1), \ldots, f'_i(\bar{\nu}_n))$ is defined iff $C(f^C_i(\bar{\nu}_1), \ldots, f^C_i(\bar{\nu}_n))$ is. Hence the same successor node is taken in both executions, so the invariant holds for $i+1$ if ν'_i is a destructor node.

In the case of a communication node, the adversary E in the computational execution gets a tuple $m' := (l, f'_i(\bar{\nu}_1), \ldots, f'_i(\bar{\nu}_n))$ where l is the out-metadata of the node ν'_i and the nodes $\bar{\nu}_s$ depend on the label of ν'_i. In the hybrid execution, the simulator gets $(l, f^C_i(\bar{\nu}_1), \ldots, f^C_i(\bar{\nu}_n))$ and sends $m^C := (l, \beta(f^C_i(\bar{\nu}_1)), \ldots, \beta(f^C_i(\bar{\nu}_n)))$ to the simulated adversary E. By Claim 2 we then have $m' = m^C$, so the adversary gets the same input in both executions. Thus $s'_{i+1} = s^C_{i+1}$ and the adversary sends the same pair (l', b_1) in both executions. The successor nodes ν'_{i+1} and ν^C_{i+1} are determined by the in-metadata l', hence $\nu'_{i+1} = \nu^C_{i+1}$. Finally, we have $f'_{i+1}(\nu'_i) = b$ and $f^C_{i+1}(\nu'_i) = \tau(b)$ (because the simulator translates the bitstring b using τ before passing it to Π^C). Thus the invariant holds for $i+1$ in the case of a communication node.

From the invariant it follows, that the node trace is the same in both executions.

Since random choices with all r_N, $pk(r_N)$, and $sk(r_N)$ being pairwise distinct for $N \in \mathcal{N}^*$ occur with overwhelming probability, the node traces of the real and the hybrid execution are indistinguishable. □

Lemma 8 *In a given step of the hybrid execution with Sim_f, let S be the set of messages sent from Π^c to Sim_f. Let $u' \in T(\mathbf{C})$ be the message sent from Sim_f to in that step. Let $\mathcal{C}$ be a context and $u \in T(\mathbf{C})$ such that $u' = \mathcal{C}[u]$ and $S \nvdash u$.*

Then there exists a term t_{bad} and a context $\mathcal{D}$ such that $\mathcal{D}$ obeys the following grammar

$$\begin{aligned}\mathcal{D} ::= \Box \mid pair(\mathcal{D}, t) \mid pair(t, \mathcal{D}) \mid E(pk(N), \mathcal{D}, M) \\ \mid E(\mathcal{D}, t, M) \mid garbageE(\mathcal{D}, M) \\ \text{with } N \in \mathbf{N}_P, M \in \mathbf{N}_E, t \in T(\mathbf{C})\end{aligned}$$

and such that $u = \mathcal{D}[t_{bad}]$ and such that $S \nvdash t_{bad}$ and such that one of the following holds: $t_{bad} \in \mathbf{N}_P$, or $t_{bad} = E(p, m, N)$ with $N \in \mathbf{N}_P$, or $t_{bad} = pk(N)$ with $N \in \mathbf{N}_P$.

Proof. We prove the lemma by structural induction on M. We distinguish the following cases:

Case "$u = garbage(u_1)$": By protocol condition 7 the protocol does not contain $garbage$-constructor nodes. Thus u is not a honestly generated term. Hence it was produced by an invocation $\tau(m)$ for some $m \in \{0,1\}^*$, and hence $u = garbage(N^m)$. Hence $S \vdash u$ in contradiction to the premise of the lemma.

Case "$u = garbageE(u_1, u_2)$": By protocol condition 7 the protocol does not contain $garbageE$-constructor nodes. Thus u is not a honestly generated term. Hence it was produced by an invocation $\tau(c)$ for some $c \in \{0,1\}^*$, and hence $u = garbageE(u_1, N^m)$. Since $S \vdash N^m$ and $S \nvdash u$, we have $S \nvdash u_1$. Hence by the induction hypothesis, there exists a subterm t_{bad} of u_1 and a context $\mathcal{D}$ satisfying the conclusion of the lemma for u_1. Then t_{bad} and $\mathcal{D}' := garbageE(\mathcal{D}, N^m)$ satisfy the conclusion of the lemma for u.

Case "$u = sk(u_1)$": By protocol condition 4, any sk-constructor node occurs only as the first argument of a D-destructor node. The output of the destructor D only contains a subterm $sk(u_1)$ if its second argument already contained such a subterm. Hence a term $sk(u_1)$ cannot be honestly generated. But terms of the form $sk(\cdot)$ are not in the range of τ. Hence u cannot be a subterm of u'.

Case "$u = pk(u_1)$ with $u_1 \notin \mathbf{N}_P$": By protocol condition 1, the argument of a pk-constructor node is a N-constructor node with $N \in \mathbf{N}_P$. Hence u is not honestly generated. Hence it was produced by an invocation $\tau(e)$ for some $e \in \{0,1\}^*$, and hence $u = pk(N^e)$. Hence $S \vdash u$ in contradiction to the premise of the lemma.

Case "$u = pk(N)$ with $N \in \mathbf{N}_P$": The conclusion of the lemma is fulfilled with $\mathcal{D} := \Box$ and $t_{bad} := u$.

Case "$u = pair(u_1, u_2)$": Since $S \nvdash u$, we have $S \nvdash u_i$ for some $i \in \{1,2\}$. Hence by induction hypothesis, there exists a subterm t_{bad} of u_i and a context $\mathcal{D}$ satisfying the conclusion of the lemma for u_i. Then t_{bad} and $\mathcal{D}' = pair(\mathcal{D}, u_2)$ or $\mathcal{D}' = pair(u_1, \mathcal{D})$ satisfy the conclusion of the lemma for u.

Case "$u \in \mathbf{N}_P$": The conclusion of the lemma is fulfilled with $\mathcal{D} := \Box$ and $t_{bad} := u$.

Case "$u \in \mathbf{N}_E$": Then $S \nvdash u$ in contradiction to the premise of the lemma.

Case "$u = E(u_1, u_2, N)$ with $N \in \mathbf{N}_P$": The conclusion of the lemma is fulfilled with $\mathcal{D} := \Box$ and $t_{bad} := u$.

Case "$u = E(u_1, u_2, u_3)$ with $S \nvdash u_2$ and $u_3 \notin \mathbf{N}_P$": By protocol condition 1, the third argument of an E-constructor node is a N-constructor node with $N \in \mathbf{N}_P$. Hence u is not honestly generated. Hence it was produced by an invocation $\tau(c)$ for some $c \in \{0,1\}^*$, and hence $u = E(pk(N), u_2, N^c)$ for some $N \in \mathbf{N}_P$. Since $S \nvdash u_1$, by induction hypothesis, there exists a subterm t_{bad} of $pk(N)$ and a context $\mathcal{D}$ satisfying the conclusion of the lemma for $pk(N)$. Then t_{bad} and $\mathcal{D}' = E(\mathcal{D}, u_2, N^c)$ satisfy the conclusion of the lemma for u.

Case "$u = E(u_1, u_2, u_3)$ with $S \vdash u_2$ and $u_3 \notin \mathbf{N}_P$": Analogous to the previous case, $u = E(pk(N), u_2, N^c)$ for some $N \in \mathbf{N}_P$. From $S \vdash u_2$, $S \vdash N^c$, and $S \nvdash u$ we have $S \nvdash u_2$. Hence by induction hyposthesis, there exists a subterm t_{bad} of u_2 and a context $\mathcal{D}$ satisfying the conclusion of the lemma for u_2. Then Then t_{bad} and $\mathcal{D}' = E(pk(N), \mathcal{D}, N^c)$ satisfy the conclusion of the lemma for u. □

Lemma 9 *Sim_f is DY for M and Π.*

Proof. Let $a_1, \ldots, a_n$ be terms sent by the protocol to Sim_f. Let $u_1, \ldots, u_n$ be the terms sent by Sim_f to the protocol.

Let $S_i := \{a_1, \ldots, a_i\}$. If Sim_f is not DY, then with non-negligible probability there exists an i such that $S_i \not\vdash u_i$. Fix the smallest such i_0 and set $S := S_{i_0}$ and $u := u_{i_0}$. By Lemma 8 (with $u' := u$ and $C := \Box$), we have that there is a term t_{bad} and a context $\mathcal{D}$ obeying the grammar given in Lemma 8 and such that $u = \mathcal{D}[t_{bad}]$ and such that $S \not\vdash t_{bad}$ and such that one of the following holds: $t_{bad} \in \mathbf{N}_P$, or $t_{bad} = E(p, m, N)$ with $N \in \mathbf{N}_P$, or $t_{bad} = pk(N)$ with $N \in \mathbf{N}_P$.

By construction of the simulator, if the simulator outputs u, we know that the simulated adversary E has produced a bitstring m such that $\tau(m) = u = \mathcal{D}[t_{bad}]$. By definition of τ, during the computation of $\tau(m)$, some recursive invocation of τ has returned t_{bad}. Hence the simulator has computed a bitstring m_{bad} with $\tau(m_{bad}) = t_{bad}$.

We are left to show that such a bitstring m_{bad} can be found only with negligible probability.

Since $S \not\vdash t_{bad}$, we have that t_{bad} occurs in S only as a part of a subterm t' of a term $t'' \in S$ where $S \vdash t$ and $S \not\vdash \mathcal{E}[t_{bad}]$ for some context $\mathcal{E}$ and $t' = E(t_1, \mathcal{E}[t_{bad}], t_3)$ or $t' = E(t_1, t_2, \mathcal{E}[t_{bad}])$ or $t' = pk(\mathcal{E}[t_{bad}])$ or $t' = sk(\mathcal{E}[t_{bad}])$ or $t' = garbage(\mathcal{E}[t_{bad}])$ or $t' = garbageE(t_1, \mathcal{E}[t_{bad}])$.

For all $i < i_0$, we have that $S_i \vdash u_i$. With $S \not\vdash \mathcal{E}[t_{bad}]$, we have that t' is a subterm of u_i only if t' was a subterm of a_j with $j \leq i$. Hence t' must be honestly generated. Because of protocol conditions 1, 4, 7, this leaves only the possibilities $t' = E(t_1, \mathcal{E}[t_{bad}], N)$ with $N \in \mathbf{N}_P$, or $t' = E(t_1, t_2, \mathcal{E}[t_{bad}])$ with $\mathcal{E}[t_{bad}] \in \mathbf{N}_P$, or $t' = pk(\mathcal{E}[t_{bad}])$ with $\mathcal{E}[t_{bad}] \in \mathbf{N}_P$.

Note that in all three cases, the computation $\beta(t')$ in Sim_f has been defined not to invoke $\beta(\mathcal{E}[t_{bad}])$. (This was the modification of Sim_f with respect to Sim.) Hence $\beta(t_{bad})$ is never invoked. If $t_{bad} = N \in \mathbf{N}_P$, then $m_{bad} = r_N$ and r_N is never used. So the probability that $m_{bad} = r_N$ occurs as output of τ is negligible. If $t_{bad} = E(p, m, N)$ with $N \in \mathbf{N}_P$, then $\tau(m_{bad})$ returns t_{bad} only if m_{bad} was the output of an invocation of $\beta(E(p, m, N)) = \beta(t_{bad})$. But since $\beta(t_{bad})$ is never invoked, this case does not occur. Finally, if $t_{bad} = pk(N)$ with $N \in \mathbf{N}_P$, then $\beta(t_{bad})$ is never computed and pk_N is never used. Furthermore, sk_N is only accessed by Sim_f when evaluating $\tau(c)$ where c is a ciphertext tagged with pk_N. Hence the probability that pk_N occurs as output of τ is negligible. (The IND-CCA property implies that guessing an unknown public key has negligible probability.)

Summarizing, we have show that if the simulator Sim_f is not DY, then with non-negligible probability Sim_f performs the computation $\tau(m_{bad})$, but m_{bad} can only occur with negligible probability as an argument of τ. Hence we have a contradiction to the assumption that Sim_f is not DY. □

Theorem 4 *A is a computationally sound implementation of $\mathbf{M}$ for encryption-safe protocols.*

Proof. By Lemma 9, Sim_f is DY for encryption-safe protocols. Whether a full trace satisfies the conditions from Definition 17 can be efficiently verified (since $\vdash$ is efficiently decidable). Hence Lemma 6 implies that Sim is DY for encryption-safe protocol, too. By Lemma 5, Sim is consistent and abort-free. By Lemma 7, Sim is indistinguishable. Hence Sim is a good simulator for $\mathbf{M}$, encryption-safe Π, A, and polynomials p. By Theorem 1, the computational soundness of A for encryption-safe protocols follows. □

E Encryption in the applied π-calculus

Consider an instantiation of our process calculus with constructors $E/3, pk/1, sk/1, pair/2, garbage/1, garbageE/2$ and destructors $ispk/1, isenc/1, D/2, fst/1, snd/1, pkof/1, equals/2$.

The semantics of the destructors is given by $D(sk(t_1), E(pk(t_1), m, t_2)) = m$. $ispk(pk(t)) = pk(t)$. $isenc(E(pk(t_1), t_2, t_3)) = E(pk(t_1), t_2, t_3)$ $isenc(garbageE(pk(t_1), t_2)) = garbageE(pk(t_1), t_2)$. $fst(pair(x, y)) = x$. $snd(pair(x, y)) = y$. $pkof(E(pk(t_1), m, t_2)) = pk(t_1)$. In all other cases, the destructors return $\perp$.

Assume an implementation of the constructors or destructors satisfying the implementation conditions given in section D.

We call a process $\tilde{P}$ encryption-safe if it has the following grammar: Let x and s stand for two different sets of variables (general purpose and secret key variables). Let a and r stand for two sets of names (general purpose and randomness names). Then the allowed terms are $\tilde{M}, \tilde{N} ::= x \mid a \mid pair(\tilde{M}, \tilde{N})$, the allowed destructor terms are $\tilde{D} ::= \tilde{M} \mid ispk(\tilde{D}) \mid isenc(\tilde{D}) \mid D(s, \tilde{D}) \mid fst(\tilde{D}) \mid snd(\tilde{D}) \mid pkof(\tilde{D})$. The allowed processes are

$$\begin{aligned}\tilde{P}, \tilde{Q} ::= {} & \overline{\tilde{M}}\langle \tilde{N} \rangle.\tilde{P} \mid \tilde{M}(x).\tilde{P} \mid 0 \mid (\tilde{P} \mid \tilde{Q}) \mid !\tilde{P} \mid \nu a.\tilde{P} \mid \\ & let\ x = \tilde{D}\ in\ \tilde{P}\ else\ \tilde{Q} \mid event(e).\tilde{P} \mid \\ & \nu r.let\ x = pk(r)\ in\ let\ s = sk(r)\ in\ \tilde{P} \mid \\ & \nu r.let\ x = E(ispk(\tilde{D}_1), \tilde{D}_2, r)\ in\ \tilde{P}\ else\ \tilde{Q}\end{aligned}$$

(Note that in the last production rules for key generation and for encryption, all occurrences of r denote the same name.)

Then from Theorem 2 and Theorem 3, we immediately get the following soundness result in this process calculus:

Theorem 5 *If a closed encryption-safe process P symbolically satisfies a π-trace property $\wp$, then P computationally satisfies $\wp$.*

Analysis of Needham-Schroeder-Lowe The Needham-Schroeder-Lowe protocol can be written as follows in our

```
fun E/3. fun pk/1. fun sk/1.
fun pair/2. fun garbage/1. fun garbageEnc/2.
reduc D(sk(t1),E(pk(t1),m,t2)) = m.
reduc ispk(pk(t)) = pk(t).
reduc isenc(E(pk(t1),t2,t3)) = E(pk(t1),t2,t3);
      isenc(garbageEnc(pk(t1),t2)) = garbageEnc(pk(t1),t2).
reduc fst(pair(x,y)) = x.
reduc snd(pair(x,y)) = y.
reduc pkof(E(pk(t1),m,t2)) = pk(t1);
      pkof(garbageEnc(pk(t1),t2)) = pk(t1).
reduc equals(x,x) = x.

query evinj:endAB() ==> evinj:beginAB().

let A = !in(net,pkX); if pkX=pkB then event beginAB(); A' else A'.
let A' = new nA; out(net,nA); new r2; in(net,c); let m=D(skA,c) in
         if nA=fst(m) then if pkX=snd(snd(m)) then
         let c'=E(ispk(pkX),fst(snd(m)),r2) in out(net,c').
let B = !in(net,pkX); in(net,nA); new nB;
        new r1; let c=E(ispk(pkX), pair(nA,pair(nB,pkB)), r1) in
        out(net,c); in(net,c'); if nB=D(skB,c') then if pkX=pkA then event endAB().
process new rA; let pkA=pk(rA) in let skA=sk(rA) in out(net,pkA);
        new rB; let pkB=pk(rB) in let skB=sk(rB) in out(net,pkB); A|B
```

Figure 5. Needham-Schroeder-Lowe in ProVerif syntax

calculus (we use syntactic sugar (x, y) for $pair(x, y)$):

$$
\begin{aligned}
A &:= !net(pk_X).\\
&\quad \mathit{if}\ pk_X = pk_B\ \mathit{then}\ event(beginAB).A'\ \mathit{else}\ A'\\
A' &:= \nu n_A.\overline{net}\langle n_A\rangle.net(c).\mathit{let}\ m = D(sk_A, c)\ \mathit{in}\\
&\quad \mathit{if}\ n_A = fst(m)\ \mathit{then}\ \mathit{if}\ pk_X = snd(snd(m))\ \mathit{then}\\
&\quad \nu r_2.\mathit{let}\ c' = E(ispk(pk_X), fst(snd(m)), r_2)\ \mathit{in}\\
&\quad \overline{net}\langle c'\rangle.0\\
B &:= !net(pk_X).net(n_A).\nu n_B.\\
&\quad \nu r_1.\mathit{let}\ c = E(ispk(pk_X), (n_A, (n_B, pk_B)), r_1)\ \mathit{in}\\
&\quad \overline{net}\langle c\rangle.net(m).\mathit{if}\ n_B = D(sk_B, m)\ \mathit{then}\\
&\quad \mathit{if}\ pk_X = pk_A\ \mathit{then}\ event(endAB)\\
P &:= \nu r_A.\mathit{let}\ pk_A = sk(r_A)\ \mathit{in}\ \mathit{let}\ sk_A = sk(r_A)\ \mathit{in}\ \overline{net}\langle pk_A\rangle.\\
&\quad \nu r_B.\mathit{let}\ pk_B = sk(r_B)\ \mathit{in}\ \mathit{let}\ sk_B = sk(r_B)\ \mathit{in}\ \overline{net}\langle pk_B\rangle.\\
&\quad (A \mid B).
\end{aligned}
$$

We model the participants A and B as processes with an unbounded number of sessions that perform authentications with arbitrary participants (the adversary may control with which communication partner an authentication is performed by sending a public key pk_X over the public channel net). If A believes to perform an authentication with B ($pk_X = pk_B$), it raises the event $beginAB$. If B believes to have completed an authentication with A, it raises the event $endAB$. Entity authentication can be expressed by requiring that every $endAB$ event is preceded by a $beginAB$ event. The process P describing the whole protocol is an encryption-safe process (if r_1, r_2, r_A, r_B are declared as randomness names and sk_A, sk_B as secret key variables).

We can encode the processes and the equational theory in ProVerif as show in Figure 5. Note that we moved νr_2 in A' up in front of the input $net(c)$. Obviously, this leads to an equivalent process (in terms of event traces), but it helps ProVerif to terminate.[10] ProVerif verifies the entity authentication property with no noticeable delay.

[10] In general, when ProVerif does not terminate, it is helpful to move all restrictions upwards as far as possible.